/Volumes/DongyunLee/ESE280 Lab/Lab9/task3lab9.asm

```
;*****************************************************************
;**********        BASIC DOG LCD TEST PROGRAM           **********
;*****************************************************************
;
;DOG_LCD_BasicTest.asm
;  Simple test application to verify DOG LCD is properly
;  wired.  This test writes simple test messages to each
;  line of the display.
;
;Version - 2.0 For DOGM163W LCD operated at 3.3V
;

     .CSEG

     ; interrupt vector table, with several 'safety' stubs
     rjmp RESET      ;Reset/Cold start vector
     reti            ;External Intr0 vector
     reti            ;External Intr1 vector

;**********************************************************************
;************* M A I N   A P P L I C A T I O N   C O D E  *************
;**********************************************************************

RESET:

    sbi VPORTA_DIR, 7       ; set PA7 = output.
    sbi VPORTA_OUT, 7       ; set /SS of DOG LCD = 1 (Deselected)

    rcall init_lcd_dog    ; init display, using SPI serial interface
    rcall clr_dsp_buffs   ; clear all three SRAM memory buffer lines

    rcall update_lcd_dog        ;display data in memory buffer on LCD

    rcall start

// display setting line
    rcall clear_line

    rcall update_lcd_dog

    // keypad subroutine
    check_press:
        wait_for_1:
    sbis VPORTB_IN, 5   ;wait for PB5 being 1
        rjmp wait_for_1     ;skip this line if PE0 is 1

    rjmp output

    rjmp check_press

    end_loop:       ;infinite loop, program's task is complete
    rjmp end_loop

;********************************************************************
; start subroutine
;********************************************************************
start:
    sbi VPORTA_DIR, 4    //MOSI output

    sbi VPORTB_DIR, 4    // clear flip flop output
    sbi VPORTB_OUT, 4   // set clear to 1

    ldi r17, 0x00
    out VPORTC_DIR, r17 // input 4 dip switch + 16 keypads
    sbi VPORTD_DIR, 0   // pulse generator

    cbi VPORTB_DIR, 5    // check if the keypad is pressed

    ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
    ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.

    ldi r19, 0x00 // register for storing value

    ret
;********************************************************************
; keypad subroutine
;********************************************************************
table: .db $31, $32, $33, $46
       .db $34, $35, $36, $45
       .db $37, $38, $39, $44
       .db $41, $30, $42, $43

second_output:  // changing the pointer to the second line which is for T multiply
    ldi XH, high (dsp_buff_2+15) ; Load ZH and ZL as a pointer to 1st
    ldi XL, low (dsp_buff_2+15)  ; byte of buffer for line 1.
    rjmp output2

output:
cpi r19, 0          // if r19 is not 0, which means that the first enter has been pressed yet, so change the pointer to the next line
brne second_output

output2:
in r18, VPORTC_IN   // gets the input from DIP switch and keypad

lsr r18     // shifting  to right 4 bits
lsr r18
lsr r18
lsr r18


// lookup table from lecture
lookup:
    ldi ZH, high (table*2)
    ldi ZL, low (table*2)
    ldi r16, $00
    add ZL, r18
    adc ZH, r16
    lpm r18, Z

    st X, r18  // storing into SRAM buffer

    clear_flipflop:     // clear the flip flop for next input
    cbi VPORTB_OUT, 4
    sbi VPORTB_OUT, 4

    cpi r18, $41    // if the pressed key is clear
```

```
        breq push_clear

    cpi r18, $43    // if the pressed key is Enter
        breq check_which_push_enter

    rcall shift_by_1

    //rcall delay_break

    rcall update_lcd_dog

rjmp check_press    // go back to the start

;********************************************************************
; check_which_push_enter
;********************************************************************
check_which_push_enter:
    cpi r19, 0
    breq enter_clear
    rjmp second_enter_clear
;********************************************************************
; delay break
;********************************************************************
delay_break:            ;delay lable for break delay
    ldi r16, 80
    outer_loop_break:
        ldi r17, 133
        inner_loop_break:
            dec r17
    brne inner_loop_break
        dec r16
brne outer_loop_break

ret
;********************************************************************
; push_clear
;********************************************************************

push_clear:
    rjmp RESET

;********************************************************************
; error loop
;********************************************************************
line3_testmessage: .db 3, "ERROR, press CLEAR", 0  ; message for line #1.

error_loop:
    rcall clr_dsp_buffs
   ldi  ZH, high(line3_testmessage<<1)  ; pointer to line 1 memory buffer
   ldi  ZL, low(line3_testmessage<<1)   ;
   rcall load_msg         ; load message into buffer(s).
   rcall update_lcd_dog

wait_for_clear:
        sbis VPORTB_IN, 5   ;wait for PB5 being 1
        rjmp wait_for_clear    ;skip this line if PE0 is 1

output_error:
in r18, VPORTC_IN   // gets the input from DIP switch and keypad

```

```
lsr r18     // shifting  to right 4 bits
lsr r18
lsr r18
lsr r18


// lookup table from lecture
lookup_error:
    ldi ZH, high (table*2)
    ldi ZL, low (table*2)
    ldi r16, $00
    add ZL, r18
    adc ZH, r16
    lpm r18, Z

    cpi r18, $41    // if the pressed key is clear
        breq push_clear


rjmp output_error

;*******************************************************************
; push enter
; r19 is the storage
;*******************************************************************
100th_addition:
    dec r17
    ldi r16, 100
    mul r18, r16 // multiply by 100 for the 100th place value. Stores in r0
    add r19, r0 // and then add the next digit on 1st
    adiw ZH:ZL, $0001
rjmp lookup2

10th_addition:
    dec r17
    ldi r16, 10 // to multiply ; shift to the left on 10th
    mul r18, r16    //shift to the left on 10th  stores in r0
    add r19, r0
    adiw ZH:ZL, $0001
rjmp lookup2

enter_clear:
    // clear the flip flop for next input
    cbi VPORTB_OUT, 4
    sbi VPORTB_OUT, 4

push_enter: // error: clear button does not work once enter is pressed

    ldi r17, 3
    ldi r19, 0x00
    ldi ZH, high (dsp_buff_1+12) ; Load ZH and ZL as a pointer to 1st
    ldi ZL, low (dsp_buff_1+12)  ; byte of buffer for line 1.

    lookup2:
        ld r18, Z
        andi r18, 0x0F  // mask


    sbic VPORTB_IN, 5   ;wait for PB5 being 1
        rjmp output

    cpi r19, 101    // check if the value is over 100
    brge error_loop // branch if it is equal or greater than 101

    // now convert the percentage value into value out of 255, and generate pulse

    cpi r19, 100
    breq birghtness_full

    cpi r19, 0
    breq birghtness_zero

    ldi r16, 2
    mul r19, r16 // multiply r19 by 2 (r16)
    mov r19, r0

    rjmp check_press

;********************************************************************
; second push enter
;  should be range of 1 - 100
; r21 is the storage
;********************************************************************
100th_addition_2:
    dec r17
    ldi r16, 100
    mul r18, r16 // multiply by 100 for the 100th place value
    add r21, r0 // and then add the next digit on 1st
    adiw ZH:ZL, $0001
rjmp lookup3

10th_addition_2:
    dec r17
    ldi r16, 10 // to multiply ; shift to the left on 10th
    mul r18, r16    //shift to the left on 10th
    add r21, r0
    adiw ZH:ZL, $0001
rjmp lookup3

second_enter_clear:
    // clear the flip flop for next input
    cbi VPORTB_OUT, 4
    sbi VPORTB_OUT, 4

second_enter:   // error: clear button does not work once enter is pressed
    ldi r21, 0x00 // r21 is the storage for second enter which is T multiply
    ldi r17, 3
    ldi ZH, high (dsp_buff_2+12) ; Load ZH and ZL as a pointer to 1st
    ldi ZL, low (dsp_buff_2+12)  ; byte of buffer for line 1.
lookup3:
    ld r18, Z
    andi r18, 0x0F  // mask

    sbic VPORTB_IN, 5   ;if no key is pressed then skip next line
        rjmp output     ; if you see a key is pressed go to output

    cpi r17, 3
    breq 100th_addition

    cpi r17, 2
    breq 10th_addition

    // 1th addition
    add r21, r18

;********************************************************************
; execute
;********************************************************************
execute:
    ldi r16, 2
    mul r19, r16 // multiply r19 by 2 (r16)
    mov r19, r0
    mov r16, r19

    ldi r20, 255
    sub r20, r19

    mov r17, r21    // r17 and r21 is the t multiply
    rjmp highloop
timing_loop:
    mov r19, r16
    ldi r20, 255
    sub r20, r19

    highloop:
        sbi VPORTD_OUT, 0

    dec_loop:
        dec r19
        brne highloop
        dec r21
        brne timing_loop
        rjmp lowloop2
timing_loop2:
    mov r19, r16
    ldi r20, 255
    sub r20, r19

    lowloop2:
        cbi VPORTD_OUT, 0

    dec_loop2:
        dec r20
        brne loop2
        dec r17
        brne timing_loop2

rjmp push_enter
;********************************************************************
; shift_by_1
;********************************************************************
second_line_shift:
    ldi ZH, high (dsp_buff_2+15) ; Load ZH and ZL as a pointer to 1st
    ldi ZL, low (dsp_buff_2+15)  ; byte of buffer for line 1.
    rjmp shift_by_1_2
shift_by_1:
cpi r19, 0
brne second_line_shift
    ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
    ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
```

```
shift_by_1_2:

    sbiw ZH:ZL, $0002
    ld r20, Z

    sbiw ZH:ZL, $0001
    st Z, r20

    adiw ZH:ZL, $0002
    ld r20, Z

    sbiw ZH:ZL, $0001
    st Z, r20

    adiw ZH:ZL, $0002
    ld r20, Z

    sbiw ZH:ZL, $0001
    st Z, r18
ldi r20, 0x20   //r20 is blank
    adiw ZH:ZL, $0001
    st Z, r20

    ret // i am not sure if this return will still work since I have branched to somewhere in the middle.
;********************************************************************
; brightness full (100%)
;********************************************************************
birghtness_full:
    sbi VPORTD_OUT, 0

    rjmp push_enter

;********************************************************************
; brightness zero (0%)
;********************************************************************
birghtness_zero:
    cbi VPORTD_OUT, 0

    rjmp push_enter


;********************************************************************
;   clear line 1
;********************************************************************

line1_testmessage: .db 1, "Setting 1  :000 ", 0  ; message for line #1.
line2_testmessage: .db 2, "T multiply :000 ", 0

clear_line:
      ;load_line_1 into dbuff1:
   ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
   ldi  ZL, low(line1_testmessage<<1)   ;
   rcall load_msg          ; load message into buffer(s).

   ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 1 memory buffer
   ldi  ZL, low(line2_testmessage<<1)   ;
   rcall load_msg          ; load message into buffer(s).

   ret
```

```

;*******************
;NAME:      load_msg
;FUNCTION:  Loads a predefined string msg into a specified diplay
;           buffer.
;ASSUMES:   Z = offset of message to be loaded. Msg format is
;           defined below.
;RETURNS:   nothing.
;MODIFIES:  r16, Y, Z
;CALLS:     nothing
;CALLED BY:
;********************************************************************
; Message structure:
;   label:  .db <buff num>, <text string/message>, <end of string>
;
; Message examples (also see Messages at the end of this file/module):
;   msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
;   msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
;
; Notes:
;   a) The 1st number indicates which buffer to load (either 1, 2, or 3).
;   b) The last number (zero) is an 'end of string' indicator.
;   c) Y = ptr to disp_buffer
;      Z = ptr to message (passed to subroutine)
;********************************************************************
load_msg:
     ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
     ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
                               ; (dsp_buff_1 for now).
     lpm R16, Z+               ; get dsply buff number (1st byte of msg).
     cpi r16, 1                ; if equal to '1', ptr already setup.
     breq get_msg_byte         ; jump and start message load.
     adiw YH:YL, 16            ; else set ptr to dsp buff 2.
     cpi r16, 2                ; if equal to '2', ptr now setup.
     breq get_msg_byte         ; jump and start message load.
     adiw YH:YL, 16            ; else set ptr to dsp buff 2.

get_msg_byte:
     lpm R16, Z+               ; get next byte of msg and see if '0'.
     cpi R16, 0                ; if equal to '0', end of message reached.
     breq msg_loaded           ; jump and stop message loading operation.
     st Y+, R16                ; else, store next byte of msg in buffer.
     rjmp get_msg_byte         ; jump back and continue...
msg_loaded:
     ret

;---------------------------- SUBROUTINES ----------------------------

;************************
;NAME:      clr_dsp_buffs
;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
;RETURNS:   nothing.
;MODIFIES:  r25,r26, Z-ptr
;CALLS:     none
;CALLED BY: main application and diagnostics
;********************************************************************
clr_dsp_buffs:
     ldi R25, 48               ; load total length of both buffer.
     ldi R26, ' '              ; load blank/space into R26.
     ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
     ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.

    ;set DDRAM address to 1st position of first line.
store_bytes:
     st  Z+, R26       ; store ' ' into 1st/next buffer byte and
                       ; auto inc ptr to next location

;====================================
.include "lcd_dog_asm_driver_avr128.inc"  ; LCD DOG init/update procedures.
;====================================
```