```
;***************************************************************************
;*
;* Title: subroutine_isr_based_display.asm
;* Author: Jason Chen
;* Version: 1
;* Last updated: 11/08/2022
;* Target: AVR128DB48
;*
;* DESCRIPTION
;*          Design Task 4:
;*          This program polls the flag associated with the pushbutton. This flag
;*          is connected to PORTE. If the flag is set, the contents of the array
;*          bcd_entries is shifted left and the BCD digit set on the least
;*          significant 4 bits of PORTC_IN are stored in the least significant
;*          byte of the bcd_entries array. Then the corresponding segment values
;*          for each digit in bcd_entries display are written into the led_display.
;*
;*          Note: entry of a non-BCD value is ignored.
;*
;* This program also continually multiplexes the display so that the digits
;* entered are constantly seen on the display. Before any digits are
;* entered the display displays 0000.
;*
;* VERSION HISTORY
;* 1.0 Original version
;***************************************************************************

.dseg                           ; start of data segment
bcd_entries: .byte 4
led_display: .byte 4
digit_num: .byte 1

.cseg                           ; start of code segment
reset:
    jmp start

.org PORTE_PORT_vect
    jmp porte_isr               ;vector for all PORTE pin change IRQs

start:
; Configure I/O ports
    ldi r16, 0xFF
    out VPORTD_DIR, r16         ; VPORTD - all pins configured as output
    ldi r16, 0xF0
    out VPORTA_DIR, r16         ; PA4 - PA7 configured as output (gate of pnp transistor)
    ldi r16, 0x00
    out VPORTC_DIR, r16         ; VPORTC - all pins configured as input
    cbi VPORTE_DIR, 0           ; PE0 configured as input

; Enable pullup resistors and inven for PORTC
```

```
/*  ldi r16, 0x88               ; inven = bit 7, pullup_enable = bit 3 (1000 1000)
    sts PORTC_PINCONFIG, r16    ; write PINCONFIG
    ldi r17, 0xFF               ; specify which PINnCTRL registers to update (all)
    sts PORTC_PINCTRLUPD, r17   ; update specified PINnCTRL registers simultaneously*/

; Configure interrupt request
    lds r16, PORTE_PIN0CTRL     ; set ISC for PE0 to rising edge
    ori r16, 0x02               ; ISC = bit 1 for rising edge
    sts PORTE_PIN0CTRL, r16     ; update PIN0CTRL register (0000 0010)
    sei

; Set pointers for arrays
    ldi XH, HIGH(bcd_entries)
    ldi XL, LOW(bcd_entries)    ; X points to bcd_entries[0]
    ldi YH, HIGH(led_display)
    ldi YL, LOW(led_display)    ; Y points to led_display[0]

; Clear arrays
    ldi r16, 0                  ; load r16 with 0
    mov r18, r16
    rcall hex_to_7seg           ; load r18 with 7 segment bit pattern to show 0
    ldi r17, 4                  ; loop control variable
    clear_entries:
        st X+, r16
        st Y+, r18
        dec r17
        brne clear_entries

; Program loop
main_loop:
    rcall multiplex_display
    rcall mux_digit_delay
    nop
    rjmp main_loop





porte_isr:
    cli
    push r16
    in r16, CPU_SREG
    push r16
    rcall poll_PE0
    pop r16
    out CPU_SREG, r16
    pop r16
    sei
    reti





;***************************************************************************
;*
;* "poll_PE0" - Poll PE0 for IRQ
;*
;* Description:
;*          Checks PE0 for interrupt request. If IRQ is met, the request is
;*          cleared and then polls digit entry.
;*
;* Author:          Jason
;* Version:         1
;* Last updated:    11/7/2022
;* Target:          AVR128DB48
;* Number of words:
;* Number of cycles:
;* Low registers modified: none
;* High registers modified: r16
;*
;* Parameters:  PE0 is checked for flag.
;* Returns:     PE0's flag is cleared.
;*
;* Notes:
;*
;***************************************************************************
poll_PE0:
    lds r16, PORTE_INTFLAGS     ; Determine if PE0's INTFLAG is set
    sbrs r16, 0                 ; Check if PE0 IRQ flag is set
    ret                         ; return to caller (main_loop) if not set
    rcall clear_irq
    rcall digit_entry
    ret





;***************************************************************************
;*
;* "clear_irq" - Clear Interrupt request
;*
;* Description:
;*          Clears PORTE_INTFLAG register.
;*
;* Author:          Jason
;* Version:         1
;* Last updated:    11/7/2022
;* Target:          AVR128DB48
;* Number of words:
;* Number of cycles:
;* Low registers modified: none
;* High registers modified: none
;*
;* Parameters:  PE0_INTFLAG to be cleared
;* Returns:     PE0_INTFLAG is cleared.
;*
;* Notes:
;*
;***************************************************************************
clear_irq:
    ldi r16, PORT_INT0_bm
    sts PORTE_INTFLAGS, r16
    ret





;***************************************************************************
;*
;* "poll_digit_entry" - Polls Pushbutton for Conditional Digit Entry
;*
;* DESCRIPTION:
;*          Polls the flag associated with the pushbutton. This flag is
;*          connected to PE0. If the flag is set, the contents of the array
;*          bcd_entries is shifted left and the BCD digit set on the least
;*          significant 4 bits of PORTC_IN are stored in the least significant
;*          byte of the bcd_entries array. Then the corresponding segment
;*          segment values for each digit in the bcd_entries display are
;*          written into the led_display. Note: entry of a non=BCD value must
;*          be ignored.
;*
;* Author:          Jason Chen
;* Version:         1
;* Last updated:    10/25/2022
;* Target:          AVR128DB48
;* Number of words:     44
;* Number of cycles:    134
;* Low registers modified:  none
;* High registers modified: none
;*
;* Parameters:
;*      bcd_entries: a four byte array that holds a series of binary
;*          represented decimals.
;*      led_display: a four byte array that holds the bit pattern to turn ON
;*          the segments dp, a-g to represent the corresponding decimal of
;*          bcd_entries array.
;*
;* Returns: Outputs the led_display array containing the bit pattern to be
;*          displayed associated to the digit position
```

```
;*
;* Notes:
;***************************************************************************
digit_entry:
    push r16        ; depopulate registers by pushing to stack
    push r17
    push r18
    push XH
    push XL
    push YH
    push YL

    in r16, VPORTC_IN
    rcall reverse_bits          ; reverse bits for 90 degree board rotation
    cpi r16, 10                 ; check if r16 >= 10
    brge repopulate

    ldi XH, HIGH(bcd_entries)   ; X points to bcd_entries[0]
    ldi XL, LOW(bcd_entries)
    ldi YH, HIGH(led_display)   ; Y points to led_display[0]
    ldi YL, LOW(led_display)
    mov r18, r16
    ldi r17, 4                  ; loop control variable, 4 step counter

    left_shift_arrays:
        ld r16, X               ; save contents of bcd_entries[i]
        st X+, r18              ; bcd_entries[i] = r18, i++
        rcall hex_to_7seg
        st Y+, r18              ; led_display[i] = r18, i++
        mov r18, r16            ; preparing r18 for next step in loop
        dec r17                 ; decrement lcv
        brne left_shift_arrays  ; repeats 3 times

    repopulate:         ; repopulate all registers with
        pop YL          ; original contents from stack
        pop YH
        pop XL
        pop XH
        pop r18
        pop r17
        pop r16
    ret





;***************************************************************************
;*
;* "multiplex_display" - Multiplex the Four Digit LED Display
;*
;* DESCRIPTION
;*          Updates a single digit of the display and increments the
;*          digit_num to the digit position to be displayed next.
;* Author:          Jason Chen
;* Version:         1
;* Last Updated:    10/24/2022
;* Target:          AVR128DB48
;* Number of words:
;* Number of cycles:
;* Low registers modified:  none
;* High registers modified: none
;*
;* Parameters:
;*      led_display: a four byte array that holds the segment values
;*          for each digit of the display. led_display[0] holds the
;*          segment patter for digit 0 (the rightmost digit) and so on.
;*
;*      digit_num: byte variable, the least significant two bits are the
;*          index of the last digit displayed.
;*
;* Returns: Outputs segment pattern and turns on digit driver for the next
;*          position in the display to be turned ON.
;*
;* Notes:   The segments are controlled by PORTD - (dp, a through g), the
;*          digit drivers are controlled by PORTA (PA7 - PA4, digit 0 - 3).
;****************************************************************************
multiplex_display:
    push r16        ; depopulate registers by pushing to stack
    push r17
    push r18

    ldi r16, 0xFF
    out VPORTD_OUT, r16     ; turn all segments OFF
    in r16, VPORTA_OUT      ; get current value of VPORTA
    ori r16, 0xF0
    out VPORTA_OUT, r16     ; turn all digits OFF

    ldi XH, HIGH(led_display)   ; X points to start of led_display array
    ldi XL, LOW(led_display)

    lds r16, digit_num      ; get current display number
    inc r16
    andi r16, 0x03          ; mask for two least significant bits
    sts digit_num, r16      ; store next digit to be displayed

    add XL, r16             ; add digit number to offset to array pointer
    brcc PC + 2             ; if no carry, skip next instruction
    inc XH

    ld r17, X
    out VPORTD_OUT, r17     ; output to segment display driver port
    in r17, VPORTA_OUT      ; get current digit driver port value
    ldi r18, 0x10           ; for next PORTA value via bit shift

    digit_pos:
        cpi r16, 0          ; if digit number is 0, use pattern in r18
        breq digit_on
        lsl r18             ; r18 shifted left if not 0
        dec r16             ; decrement digit number offset
        rjmp digit_pos

    digit_on:
        eor r17, r18        ; complement digit driver position
        out VPORTA_OUT, r17 ; turn selected digit ON

    pop r18         ; repopulate all registers with
    pop r17         ; original contents from stack
    pop r16
    ret





;***************************************************************************
;*
;* "reverse_bits" - Reverse Bit Order in a Register
;*
;* Description:
;*          Reverse the order of bits register 17, which reads the input
;*          switches, into register 18.
;*
;* Author:          Jason Chen
;* Version:         1
;* Last updated:    10/13/2022
;* Target:          AVR128DB48
;* Number of words:
;* Number of cycles:    8
;* Low registers modified: none
;* High registers modified: r16
;*
;* Parameters: r16 containing original bit order
;*
;* Returns: r16 containing reversed reversed bits
;*
;* Notes:
;*
;***************************************************************************
reverse_bits:
    push r17        ; write contents of r17 and r18 to stack
    push r18

    ldi r18, 0x00
    ldi r17, 0x08   ; 8 step counter

    bits_loop:
        lsl r16         ; left shift r16, original register
        ror r18         ; rotate right r18, reversed register
        dec r17
        cpi r17, 0x00   ; ----- probably can delete
        brne bits_loop  ; repeats 7 times
        mov r16, r18    ; copy bit pattern into r16

    pop r18         ; retrieve original contents of r17 and r18 from stack
    pop r17
    ret





;***************************************************************************
;*
;* "mux_digit_delay" - Multiplex Digit Delay / Variable Delay
;*
;* Description: Delays r16 * 1ms (approx.)
;*
;* Author:          Jason Chen
;* Version:         1
;* Last updated:    10/13/2022
;* Target:          AVR128DB48
;* Number of words:     11
;* Number of cycles:
;* Low registers modified:  none
;* High registers modified: none
;*
;* Parameters:
;*
;* Returns:
;*
;* Notes:
;*
;***************************************************************************
mux_digit_delay:
    push r16        ; write contents of r16 and r17 to stack
    push r17

    ldi r16, 1      ; outer loop control variable

    outer_loop:
        ldi r17, 133    ; inner loop control variable

    inner_loop:
        dec r17
        brne inner_loop
        dec r16
        brne outer_loop

    pop r17         ; retrieve original contents of r16 and r17 from stack
    pop r16
    ret




;***************************************************************************
;*
;* "hex_to_7seg" - Hexadecimal to Seven Segment Conversion
;*
;* Description:
;*         Converts a right justified hexadecimal digit to the seven
;*         segment pattern required to display it. Pattern is right
;*         justified a through g. Pattern uses 0s to turn segments on ON.
;*
;* Author:          Ken Short
;* Version:         0.1
;* Last updated:    10/03/2022
;* Target:          AVR128DB48
;* Number of words:     1
;* Number of cycles:    1
;* Low registers modified: none
;* High registers modified: r18
;*
;* Parameters: r18: hex digit to be converted
;* Returns: r18: seven segment pattern. 0 turns segment ON
;*
;* Notes:
;*
;***************************************************************************
hex_to_7seg:
    push r16
    ldi ZH, HIGH(hextable * 2)  ; set Z to point to start of table
    ldi ZL, LOW(hextable * 2)

    ldi r16, $00        ; add offset to Z pointer
    andi r18, 0x0F      ; mask for low nibble
    add ZL, r18
    adc ZH, r16
    lpm r18, Z          ; load byte from table pointed to by Z
    pop r16
    ret

; Table of segment values to display digits 0 - F
; dp, a - g
hextable: .db $01, $4F, $12, $06, $4C, $24, $20, $0F, $00, $04;, $08, $60, $31,
  $42, $30, $38
; dp, g - a
;hextable: .db $40, $79, $24, $30, $19, $12, $02, $78, $00, $10
```