```

AVRASM ver. 2.2.7  E:\ESE_280\$MyDocuments$\Atmel Studio\7.0\lab_9\PE0_and_PE2_intrs\PE0_and_PE2_intrs\main.asm Tue Nov 03 19:29:58 2020

E:\ESE_280\$MyDocuments$\Atmel Studio\7.0\lab_9\PE0_and_PE2_intrs\PE0_and_PE2_intrs\main.asm(9): Including file 'C:/Program Files (x86)\Atmel\Studio\7.0\Packs\atmel\ATmega_DFP\1.3.300\avrasm\inc\m4809def.inc'
E:\ESE_280\$MyDocuments$\Atmel Studio\7.0\lab_9\PE0_and_PE2_intrs\PE0_and_PE2_intrs\main.asm(9): Including file 'C:/Program Files (x86)\Atmel\Studio\7.0\Packs\atmel\ATmega_DFP\1.3.300\avrasm\inc\m4809def.inc'


                                 ; PE0_and_PE2_intrs.asm
                                 ;
                                 ; Created: 10/30/2020 9:44:44 PM
                                 ; Author : hp
                                 ;

                                 .list

                                 .dseg
002800                           PB1_count: .byte 1  ;pushbutton 1 presses.
002801                           PB2_count: .byte 1  ;pushbutton 2 presses.


                                 .cseg                  ;start of code segment
                                 reset:
000000 940c 0048                    jmp start           ;reset vector executed a power on

                                 .org PORTE_PORT_vect
000046 940c 005c                    jmp porte_isr       ;vector for all PORTE pin change IRQs


                                 start:
                                     ; Configure I/O ports
000048 9880                         cbi VPORTE_DIR, 0   ;PE0 input- gets output from PB1
000049 9882                         cbi VPORTE_DIR, 2   ;PE2 input- gets output from PB2

00004a e000                         ldi r16, 0x00       ;make initial counts 0
00004b 9300 2800                    sts PB1_count, r16
00004d 9300 2801                    sts PB2_count, r16

                                    ;Configure interrupts
00004f 9100 0490                    lds r16, PORTE_PIN0CTRL  ;set ISC for PE0 to pos. edge
```

```
000051 6002                          ori r16, 0x02       ;set ISC for rising edge
000052 9300 0490                     sts PORTE_PIN0CTRL, r16

000054 9100 0492                     lds r16, PORTE_PIN2CTRL ;set ISC for PE2 to pos. edge
000056 6002                          ori r16, 0x02       ;set ISC for rising edge
000057 9300 0492                     sts PORTE_PIN2CTRL, r16

000059 9478                          sei         ;enable global interrupts

                                  main_loop:     ;main program loop
00005a 0000                          nop
00005b cffe                          rjmp main_loop


                                  ;Interrupt service routine for any PORTE pin change IRQ
                                  porte_ISR:
00005c 930f                          push r16        ;save r16 then SREG
00005d b70f                          in r16, CPU_SREG
00005e 930f                          push r16
00005f 94f8                          cli             ;clear global interrupt enable

                                     ;Determine which pins of PORTE have IRQs
000060 9100 0489                     lds r16, PORTE_INTFLAGS ;check for PE0 IRQ flag set
000062 fd00                          sbrc r16, 0
000063 d008                          rcall PB1_sub           ;execute subroutine for PE0

000064 9100 0489                     lds r16, PORTE_INTFLAGS ;check for PE2 IRQ flag set
000066 fd02                          sbrc r16, 2
000067 d00d                          rcall PB2_sub           ;execute subroutine for PE2

000068 910f                          pop r16         ;restore SREG then r16
000069 bf0f                          out CPU_SREG, r16
00006a 910f                          pop r16
00006b 9518                          reti            ;return from PORTE pin change ISR


                                  ;Subroutines called by porte_ISR
                                  PB1_sub:       ;PE0's task to be done
00006c 9100 2800                     lds r16, PB1_count      ;get current count
```

```
  for PB1
00006e 9503                         inc r16                 ;increment count
00006f 9300 2800                    sts PB1_count, r16      ;store new count
000071 e001                         ldi r16, PORT_INT0_bm   ;clear IRQ flag for
   PE0
000072 9300 0489                    sts PORTE_INTFLAGS, r16
000074 9508                         ret


                                 PB2_sub:       ;PE2's task to be done
000075 9100 2801                    lds r16, PB2_count      ;get current count
  for PB2
000077 9503                         inc r16                 ;increment count
000078 9300 2801                    sts PB2_count, r16      ;store new count
00007a e004                         ldi r16, PORT_INT2_bm   ;clear IRQ flag for
   PE2
00007b 9300 0489                    sts PORTE_INTFLAGS, r16
00007d 9508                         ret



RESOURCE USE INFORMATION
------------------------

Notice:
The register and instruction counts are symbol table hit counts,
and hence implicitly used resources are not counted, eg, the
'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

"ATmega4809" register use summary:
x  :   0 y  :   0 z  :   0 r0 :   0 r1 :   0 r2 :   0 r3 :   0 r4 :   0
r5 :   0 r6 :   0 r7 :   0 r8 :   0 r9 :   0 r10:   0 r11:   0 r12:   0
r13:   0 r14:   0 r15:   0 r16:  29 r17:   0 r18:   0 r19:   0 r20:   0
r21:   0 r22:   0 r23:   0 r24:   0 r25:   0 r26:   0 r27:   0 r28:   0
r29:   0 r30:   0 r31:   0
Registers used: 1 out of 35 (2.9%)

"ATmega4809" instruction use summary:
.lds  :   0 .sts  :   0 adc   :   0 add   :   0 adiw  :   0 and   :   0
andi  :   0 asr   :   0 bclr  :   0 bld   :   0 brbc  :   0 brbs  :   0
brcc  :   0 brcs  :   0 break :   0 breq  :   0 brge  :   0 brhc  :   0
brhs  :   0 brid  :   0 brie  :   0 brlo  :   0 brlt  :   0 brmi  :   0
brne  :   0 brpl  :   0 brsh  :   0 brtc  :   0 brts  :   0 brvc  :   0
```

```
brvs  :   0 bset  :   0 bst   :   0 call  :   0 cbi   :   2 cbr   :   0
clc   :   0 clh   :   0 cli   :   1 cln   :   0 clr   :   0 cls   :   0
clt   :   0 clv   :   0 clz   :   0 com   :   0 cp    :   0 cpc   :   0
cpi   :   0 cpse  :   0 dec   :   0 des   :   0 eor   :   0 fmul  :   0
fmuls :   0 fmulsu:   0 icall :   0 ijmp  :   0 in    :   1 inc   :   2
jmp   :   2 ld    :   0 ldd   :   0 ldi   :   3 lds   :   6 lpm   :   0
lsl   :   0 lsr   :   0 mov   :   0 movw  :   0 mul   :   0 muls  :   0
mulsu :   0 neg   :   0 nop   :   1 or    :   0 ori   :   2 out   :   1
pop   :   2 push  :   2 rcall :   2 ret   :   2 reti  :   1 rjmp  :   1
rol   :   0 ror   :   0 sbc   :   0 sbci  :   0 sbi   :   0 sbic  :   0
sbis  :   0 sbiw  :   0 sbr   :   0 sbrc  :   2 sbrs  :   0 sec   :   0
seh   :   0 sei   :   1 sen   :   0 ser   :   0 ses   :   0 set   :   0
sev   :   0 sez   :   0 sleep :   0 spm   :   0 st    :   0 std   :   0
sts   :   8 sub   :   0 subi  :   0 swap  :   0 tst   :   0 wdr   :   0

Instructions used: 19 out of 114 (16.7%)

"ATmega4809" memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x0000fc    116      0    116   49152   0.2%
[.dseg] 0x002800 0x002802      0      2      2    6144   0.0%
[.eseg] 0x000000 0x000000      0      0      0     256   0.0%

Assembly complete, 0 errors, 0 warnings
```

```
;***************************************************************************
;*
;* Title: BCD to Hex
;* Author:  Judah Ben-Eliezer
;* Version: 1.0
;* Last updated:
;* Target:          ;ATmega4809 @3.3MHz
;*
;* DESCRIPTION
;* This program polls the flag associated with pushbutton 1. 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
;* PORTA_IN are stored in the least significant byte of the bcd_entries array.
;* Then the corresponding segment values for each digit in the 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.
;*
;* This program also polls the flag associated with pushbutton 2. This flag
;* is connected to PE2. If the flag is set, the digits in the bcd_entries
;* array are read and passed to the prewritten subroutine BCD2bin16. This
;* subroutine performs a BCD to binary conversion. The binary result is
;* partitioned into hexadecimal and placed into the array hex_results. The
;* contents of the hex_results array is converted to seven segment values
;* and placed into the led_display array. The multiplexing then causes
;* the hexadecimal equivalent of the BCD value entered to be displayed in
;* hexadecimal.
;*
;* VERSION HISTORY
;* 1.0 Original version
;***************************************************************************

.nolist
.include "m4809def.inc"
.list

.dseg
bcd_entries: .byte 4
led_display: .byte 4
digit_num: .byte 1
hex_results: .byte 4

.cseg
start:
    cbi VPORTE_DIR, 0
    ldi r16, $00
    out VPORTA_DIR, r16
    com r16
    out VPORTD_DIR, r16
    out VPORTC_DIR, r16
    ldi XH, HIGH(bcd_entries)
    ldi XL, LOW(bcd_entries)
    ldi YH, HIGH(led_display)
    ldi YL, LOW(led_display)
    com r16
    st X+, r16
    inc r16
    st X+, r16
    inc r16
    st X+, r16
    inc r16
    st X, r16
    cbi VPORTE_OUT, 1
    sbi VPORTE_OUT, 1

main_loop:
    rcall multiplex_display
    rcall mux_digit_delay
    rcall poll_digit_entry
    rcall poll_bcd_hex
    rjmp main_loop


;***************************************************************************
;*
;* "poll_digit_entry" - Polls Pushbutton 1 for Conditional Digit Entry
;*
;* Description:
;* Polls the flag associated with pushbutton 1. 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 PORTA_IN are
;* stored in the least significant byte of the bcd_entries array. Then the
;* corresponding segment values for each digit in the bcd_entries display are
;* written into the led_display. Note: entry of a non-BCD value is ignored.
;* Author:
;* Version:
;* Last updated:
;* Target:
;* Number of words:
;* Number of cycles:
;* Low registers modified:
;* High registers modified:
;*
;* Parameters:
;* Returns:
;*
;* Notes:
;*
;***************************************************************************
poll_digit_entry:
    ldi XH, HIGH(bcd_entries)
    ldi XL, LOW(bcd_entries)
    sbis VPORTE_IN, 0
    rjmp poll_digit_entry
    in r16, VPORTA_IN
    rcall reverse_bits
    rcall check_for_non_bcd
    rcall shift_bcd_entries
    rcall bcd_to_led

;***************************************************************************
;*
;* "poll_bcd_hex" - Polls Pushbutton 2 for Conditional Conversion of BCD to
;* Hex.
;*
;* Description:
;* Polls the flag associated with pushbutton 2. This flag is connected to
;* PE2. If the flag is set, the digits in the bcd_entries array are read
;* and passed to the prewritten subroutine BCD2bin16. This subroutine
;* performs a BCD to binary conversion. The binary result is partitioned
;* into hexadecimal and placed into the array hex_results. The contents of
;* the hex_results array is converted to seven segment values and placed
;* into the led_display array.
;* Author:
;* Version:
;* Last updated:
;* Target:
;* Number of words:
;* Number of cycles:
;* Low registers modified:
;* High registers modified:
;*
;* Parameters:
;* Returns:
;*
```

```
;* Notes:
;*
;***************************************************************************
poll_bcd_hex:
    sbis VPORTE_IN, 2
    ret
    ldi XH, HIGH(bcd_entries)
    ldi XL, LOW(bcd_entries)
    ld r18, X+
    ld r17, X+
    swap r17
    ld r19, X+
    or r17, r19
    ld r16, X+
    swap r16
    ld r19, X+
    or r16, r19
    rcall BCD2bin16
    ldi XH, HIGH(hex_results)
    ldi XL, LOW(hex_results)
    ldi r19, $00
    or r19, r15
    andi r19, $F0
    swap r19
    st X+, r19
    lds r20, r15
    lds r21, r14
    andi r20, $0F
    st X+, r20
    ldi r19, $00
    or r19, r21
    andi r19, $F0
    swap r19
    st X+, r19
    andi r21, $0F
    st X, r21
    ret


;***************************************************************************
;*
;* "mux_digit_delay" - title
;*
;* Description: delays 0.1 * r23
;*
;* Author:  Judah Ben-Eliezer
;* Version: 1.0
;* Last updated:
;* Target:
;* Number of words:
;* Number of cycles:
;* Low registers modified:
;* High registers modified:
;*
;* Parameters:
;* Returns:
;*
;* Notes:
;*
;***************************************************************************
mux_digit_delay:
    ldi r23, $08 ; 0.1 * r23 = delay
outer_loop:
    ldi r24, $06
inner_loop:
    dec r24
    brne inner_loop
    dec r23
    brne outer_loop
    ret


;***************************************************************************
;*
;* "reverse_bits" - Reverse Bits
;*
;* Description: Reverses the bit positions in a byte passed in. Bit 0
;* becomes bit 7, bit 6 becomes bit 1, and so on.
;*
;* Author:                  Judah Ben-Eliezer
;* Version:                 1.0
;* Last updated:            101120
;* Target:                  ATmega4809
;* Number of words:         8
;* Number of cycles:
;* Low registers modified:  r16, r17, r18
;* High registers modified: none
;*
;* Parameters: r16: byte to be reversed.
;* Returns: r16: reversed byte
;*
;* Notes:
;*
;***************************************************************************
reverse_bits:
    ldi r18, $08
loop_8:
    ror r16
    rol r17
    dec r18
    brne loop_8
    ret

check_for_non_bcd:
    cpi r17, $0A
    brsh reset
    ret

reset:
    cbi VPORTE_OUT, 1
    sbi VPORTE_OUT, 1
    ret

shift_bcd_entries:
    ldi r18, $03
shift_loop:
    ldi XH, HIGH(bcd_entries)
    ldi XL, LOW(bcd_entries)
    dec r18
    add XL, r18
    ld r19, X+
    st X, r19
    brne shift_loop
    ret

bcd_to_led:
    ldi XL, LOW(bcd_entries)
    ldi XH, HIGH(bcd_entries)
    st X, r17
    ldi r20, $04
conversion_loop:
    dec r20
    ldi XH, HIGH(bcd_entries)
    ldi XL, LOW(bcd_entries)
    ldi YH, HIGH(led_display)
    ldi YL, LOW(led_display)
    add XL, r20
    add YL, r20
    ld r18, X
    rcall hex_to_7seg
    st Y, r18
    cpi r20, $00
    brne conversion_loop
    ret

multiplex_display:
    ldi r22, $00
    sts digit_num, r22
    ldi YL, LOW(led_display)
    lds r17, digit_num
    lds r20, digit_num
    andi r17, $03
    add XL, r17
    ld r18, Y
    ldi r21, $80
    inc r20
loop:
    lsr r21
    dec r20
    brne loop
    lsl r21
    com r21
    out VPORTC_OUT, r21
    out VPORTD_OUT, r18
    cbi VPORTE_OUT, 1
    sbi VPORTE_OUT, 1
    inc r17
    sts digit_num, r17
    ret

;***************************************************************************
;*
;* "BCD2bin16" - BCD to 16-Bit Binary Conversion
;*
;* This subroutine converts a 5-digit packed BCD number represented by
;* 3 bytes (fBCD2:fBCD1:fBCD0) to a 16-bit number (tbinH:tbinL).
;* MSD of the 5-digit number must be placed in the lowermost nibble of fBCD2.
;*
;* Let "abcde" denote the 5-digit number. The conversion is done by
;* computing the formula: 10(10(10(10a+b)+c)+d)+e.
;* The subroutine "mul10a"/"mul10b" does the multiply-and-add operation
;* which is repeated four times during the computation.
;*
;* Number of words  :30
;* Number of cycles :108
;* Low registers used   :4 (copyL,copyH,mp10L/tbinL,mp10H/tbinH)
;* High registers used  :4 (fBCD0,fBCD1,fBCD2,adder)
;*
;***************************************************************************

;***** "mul10a"/"mul10b" Subroutine Register Variables

.def    copyL   =r12        ;temporary register
.def    copyH   =r13        ;temporary register
.def    mp10L   =r14        ;Low byte of number to be multiplied by 10
.def    mp10H   =r15        ;High byte of number to be multiplied by 10
.def    adder   =r19        ;value to add after multiplication

;***** Code

mul10a: ;***** multiplies "mp10H:mp10L" with 10 and adds "adder" high nibble
    swap    adder
mul10b: ;***** multiplies "mp10H:mp10L" with 10 and adds "adder" low nibble
    mov copyL,mp10L ;make copy
    mov copyH,mp10H
    lsl mp10L       ;multiply original by 2
    rol mp10H
    lsl copyL       ;multiply copy by 2
    rol copyH
    lsl copyL       ;multiply copy by 2 (4)
    rol copyH
    lsl copyL       ;multiply copy by 2 (8)
    rol copyH
    add mp10L,copyL ;add copy to original
    adc mp10H,copyH
    andi    adder,0x0f  ;mask away upper nibble of adder
    add mp10L,adder ;add lower nibble of adder
    brcc    m10_1       ;if carry not cleared
    inc mp10H       ;   inc high byte
m10_1:  ret

;***** Main Routine Register Variables

.def    tbinL   =r14        ;Low byte of binary result (same as mp10L)
.def    tbinH   =r15        ;High byte of binary result (same as mp10H)
.def    fBCD0   =r16        ;BCD value digits 1 and 0
.def    fBCD1   =r17        ;BCD value digits 2 and 3
.def    fBCD2   =r18        ;BCD value digit 5

;***** Code

BCD2bin16:
    andi    fBCD2,0x0f  ;mask away upper nibble of fBCD2
    clr mp10H
    mov mp10L,fBCD2 ;mp10H:mp10L = a
    mov adder,fBCD1
    rcall   mul10a      ;mp10H:mp10L = 10a+b
    mov adder,fBCD1
    rcall   mul10b      ;mp10H:mp10L = 10(10a+b)+c
    mov adder,fBCD0
    rcall   mul10a      ;mp10H:mp10L = 10(10(10a+b)+c)+d
    mov adder,fBCD0
    rcall   mul10b      ;mp10H:mp10L = 10(10(10(10a+b)+c)+d)+e
    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:                     1.0
;* Last updated:                101620
;* Target:                      ATmega4809
;* Number of words:             8
;* Number of cycles:            13
;* Low registers modified:      none
;* High registers modified:     r16, r18, ZL, ZH
;*
;* Parameters: r18: right justified hex digit, high nibble 0
;* Returns: r18: segment values a through g right justified
;*
;* Notes:
;*
;***************************************************************************

hex_to_7seg:
    andi r18, 0x0F              ;clear ms nibble
    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
    add ZL, r18
    adc ZH, r16
    lpm r18, Z                  ;load byte from table pointed to by Z
    ret

    ;Table of segment values to display digits 0 - F
    ;!!! seven values must be added - verify all values
hextable: .db $01, $4F, $12, $06, $4C, $24, $20, $0F, $00, $04, $08, $60, $31, $42,
$30, $38
```