```
;***************************************************************************
;* Title:           temp_hum_sys_ext.asm
;* Author:          Jason Chen
;* Version:         1
;* Last updated:    11/28/2022
;* Target:          AVR128DB48
;*
;* DESCRIPTION
;*      Design Task 4:
;*          This program verifies the data measured from the DHT11 and
;*          decodes the information to display on the 4-Digit-7-Segment
;*          display. The pushbutton will switch between displaying humidity
;*          and temperature
;*
;* VERSION HISTORY
;* 1.0 Original version
;***************************************************************************

.equ PERIOD = 38        ; 38 for 2.5ms multiplex delay, 1 for 0.128ms

.dseg
bcd_entries:    .byte 4
led_display:    .byte 4
digit_num:      .byte 1
measured_data:  .byte 5
mode:           .byte 1

.cseg                          ; start of code segment
reset:
    jmp start

.org TCA0_OVF_vect
    jmp ovf_mux_isr            ; vector for overflow IRQ

.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
    ldi r16, 0xF0
    out VPORTA_DIR, r16
    ldi r16, 0x00
    cbi VPORTE_DIR, 0
    sbi VPORTE_DIR, 1
    sbi VPORTE_OUT, 1
    sbi VPORTE_DIR, 2
    sbi VPORTE_OUT, 2
    sbi VPORTE_DIR, 3
    sbi VPORTE_OUT, 3


```

```
; Configure TCA0
    ldi r16, TCA_SINGLE_WGMODE_NORMAL_gc    ; WGMODE normal
    sts TCA0_SINGLE_CTRLB, r16

    ldi r16, TCA_SINGLE_OVF_bm              ; enable overflow interrupt
    sts TCA0_SINGLE_INTCTRL, r16

    ldi r16, LOW(PERIOD)                    ; set the period
    sts TCA0_SINGLE_PER, r16
    ldi r16, HIGH(PERIOD)
    sts TCA0_SINGLE_PER + 1, r16

    ldi r16, TCA_SINGLE_CLKSEL_DIV256_gc | TCA_SINGLE_ENABLE_bm ; set clock and 
      start timer
    sts TCA0_SINGLE_CTRLA, r16

; 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:
    cli
    rcall send_start
    rcall wait_for_response_signal
    rcall get_measured_data
    sei
    rcall delay_2s
    rjmp main_loop

;***************************************************************************
;* "send_start" - Send start signal
;* Author:          Jason Chen
;* Description:
;*      Makes DATA line 0 for 18ms then a 1 for 20us (or 40us).
;***************************************************************************

send_start:
    rcall write_0_to_DHT11
    rcall delay_18ms
    rcall write_1_to_DHT11
    rcall delay_20us
    ret

write_0_to_DHT11:
    sbi VPORTB_DIR, 0
    ret

write_1_to_DHT11:
    cbi VPORTB_DIR, 0
    ret

;***************************************************************************
;* "read_DHT11" "wait_for_0" "wait_for_1" - Read DHT11 and Wait for signal
;* Author:          Jason Chen
;* Description:
;*      Read the current logic level of the DATA line. Wait and return once
;*      the DATA line is read to be either 0 or 1.
;***************************************************************************

read_DHT11:             ; saves contents to bit 0 of r16
    in r16, VPORTB_IN
    andi r16, 0x01
    ret

wait_for_0:
    rcall read_DHT11
    cpi r16, 0x00
    brne wait_for_0
    ret

wait_for_1:
    rcall read_DHT11
    cpi r16, 0x01
    brne wait_for_1
    ret

;***************************************************************************
;* "wait_for_response_signal" - Wait for response signal
;* Author:          Jason Chen
;* Description:
;*      Wait for the response from the DHT11 by detecting a 0 and 1 for 80us
;*      each.
;***************************************************************************

```

```
wait_for_response_signal:
    rcall wait_for_0
    rcall delay_80us
    rcall wait_for_1
    rcall delay_80us
    ret

;***************************************************************************
;* "read_DHT11_data_bit" "get_byte" "get_measured_data" - Reading data
;* Author:          Jason Chen
;* Description:
;*      Retrieve a the bit information from the DATA line. Get a full byte
;*      by reading 8 bits. Save 5 bytes of information in memory.
;***************************************************************************

read_DHT11_data_bit:    ; records bit in bit 0 of r16
    rcall wait_for_0
    rcall delay_20us
    rcall wait_for_1
    rcall delay_30us
    rcall read_DHT11            ; record logic level of DATA line as input
    ret

get_byte:               ; records byte in r18
    push r17
    ldi r17, 8
    read_bit:
        rcall read_DHT11_data_bit
        lsr r16
        rol r18
        dec r17
        brne read_bit
    pop r17
    ret

get_measured_data:
    push r17
    ldi r17, 5
    ldi XH, HIGH(measured_data)
    ldi XL, LOW(measured_data)
    read_data:
        rcall get_byte
        st X+, r18
        dec r17
        brne read_data
    pop r17
    ret

;***************************************************************************
;* "delay_N" - Delay
;* Author:          Jason Chen
;* Description:
```

```
;*      Delay 2s:   ~8 million clock cycles
;*      Delay 18ms: 72000 clock cycles accounting for 2 clocks from rcall
;*      Delay 50us: 200 clock cycles accounting for 2 clocks from rcall
;*      Delay 30us: 120 clock cycles accounting for 2 clocks from rcall
;*      Delay 20us: 80 clock cycles accounting for 2 clocks from rcall
;***************************************************************************

delay_2s:           ; 8,001,222 clocks (1222 extra)
    push r16
    ldi r16, 111
    loop_2s:
        rcall delay_18ms
        rcall delay_20us
        dec r16
        brne loop_2s
    pop r16
    ret

delay_18ms:         ; 72,000 clocks total (2 from rcall)
    push r16                ; 1 clock
    push r17                ; 1 clock
    ldi r16, 106            ; 1 clock (m)
    outer_18ms:             ; (4 + N)m - 1 = 71867 clocks
        ldi r17, 225            ; 1 clock (n)
        inner_18ms:             ; 3n - 1 = 674 clocks (N)
            dec r17                 ; 1 clock
            brne inner_18ms         ; 1/2 clocks
        dec r16                 ; 1 clock
        brne outer_18ms         ; 2/1 clocks
        rcall delay_30us    ; 120 clocks
    pop r17                 ; 2 clocks
    pop r16                 ; 2 clocks
    ret                     ; 4 clocks

delay_80us:         ; 320 clocks total (2 from rcall)
    push r16                ; 1 clock
    ldi r16, 103            ; 1 clock (n)
    loop_80us:              ; 3n - 1 = 308 clocks
        dec r16                 ; 1 clock
        brne loop_20us          ; 2/1 clocks
    nop nop                 ; 2 clock
    pop r16                 ; 2 clocks
    ret                     ; 4 clocks

delay_50us:         ; 200 clocks total (2 from rcall)
    push r16                ; 1 clock
    ldi r16, 63             ; 1 clock (n)
    loop_50us:              ; 3n - 1 = 188 clocks
        dec r16                 ; 1 clock
        brne loop_50us          ; 2/1 clocks
    nop nop                 ; 2 clock
    pop r16                 ; 2 clocks
    ret                     ; 4 clocks

delay_30us:         ; 120 clocks total (2 from rcall)
    push r16                ; 1 clock
    ldi r16, 34             ; 1 clock (n)
    loop_30us:              ; 3n - 1 = 110 clocks
        dec r16                 ; 1 clock
        brne loop_20us          ; 2/1 clocks
    pop r16                 ; 2 clocks
    ret                     ; 4 clocks

delay_20us:         ; 80 clocks total (2 from rcall)
    push r16                ; 1 clock
    ldi r16, 23             ; 1 clock (n)
    loop_20us:              ; 3n - 1 = 68 clocks
        dec r16                 ; 1 clock
        brne loop_20us          ; 2/1 clocks
    nop nop                 ; 2 clock
    pop r16                 ; 2 clocks
    ret                     ; 4 clocks

;****************************************************************************
;* "ovf_mux_isr" "porte_isr" - Interrupt subroutines
;* Author:          Jason Chen
;* Description:
;*          Interrupt subroutine jumps here
;****************************************************************************

ovf_mux_isr:
    push r16                        ; save registers
    in r16, CPU_SREG
    push r16
    rcall multiplex_display         ; multiplex display
    ldi r16, TCA_SINGLE_OVF_bm      ; clear OVF flag
    sts TCA0_SINGLE_INTFLAGS, r16
    pop r16                         ; restore registers
    out CPU_SREG, r16
    pop r16
    reti

porte_isr:
    cli                     ; clear interrupt
    push r16                ; save registers
    in r16, CPU_SREG
    push r16
    rcall poll_PE0          ; poll PE0
    pop r16                 ; restore registers
    out CPU_SREG, r16
    pop r16
    sei                     ; enable interrupt
    reti

```

```
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 switch_mode
    ret

clear_irq:
    ldi r16, PORT_INT0_bm
    sts PORTE_INTFLAGS, r16
    ret

switch_mode:
    push r16
    lds r16, mode
    inc r16
    /*andi r16, 0x01*/
    cpi r16, 3
    brne switch_here
    ldi r16, 0

    switch_here:
        sts mode, r16

    rcall change_led

    pop r16
    ret

change_led:
    push r17
    cpi r16, 0x01
    breq led_1
    cpi r16, 0x02
    breq led_2
    led_0:
        cbi VPORTE_OUT, 1
        sbi VPORTE_OUT, 2
        sbi VPORTE_OUT, 3
        rjmp return_from_change

    led_1:
        cbi VPORTE_OUT, 2
        sbi VPORTE_OUT, 1
        sbi VPORTE_OUT, 3
        rjmp return_from_change

    led_2:
        cbi VPORTE_OUT, 3
        sbi VPORTE_OUT, 2
        sbi VPORTE_OUT, 1

    return_from_change:
    pop r17
    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

    rcall check_mode
    rcall write_mode

    lds r16, led_display + 1    ; place decimal
    ldi r17, 0x80
    eor r16, r17
    sts led_display + 1, r16

    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

check_mode:
    push r16
    push r17
    push r18
    lds r18, mode           ; check mode
    cpi r18, 0x01
    breq mode_1_cel
    cpi r18, 0x02
    breq mode_2_fah

    mode_0_hum:
        ldi r16, 0
        sts bcd_entries + 3, r16
        lds r16, measured_data + 0
        rcall bin2bcd8
        sts bcd_entries + 2, r17
        sts bcd_entries + 1, r16
        lds r16, measured_data + 1
        sts bcd_entries + 0, r16
        rjmp return_from_check

    mode_1_cel:
        ldi r16, 0
        sts bcd_entries + 3, r16
        lds r16, measured_data + 2
        rcall bin2bcd8
        sts bcd_entries + 2, r17
        sts bcd_entries + 1, r16
        lds r16, measured_data + 3
        sts bcd_entries + 0, r16
        rjmp return_from_check

    mode_2_fah:lm
        lds r16, measured_data + 2
        ldi r17, 10
        rcall mpy8u
        lds r16, measured_data + 3
        ldi r19, 0
        add r17, r16
        adc r18, r19
        mov r16, r17
        mov r17, r18
        ldi r18, 18
        rcall mpy16u
        ldi r16, 0x80
        ldi r17, 0x0C
        add r16, r18
        adc r17, r19
        rcall bin2bcd16

        mov r16, r13
        mov r17, r14
        mov r18, r14
        mov r19, r15
        andi r16, 0xF0
        andi r17, 0x0F
        andi r18, 0xF0
        andi r19, 0x0F
        ldi r20, 4

        right_shift:
            lsr r16
            lsr r18
            dec r20
            brne right_shift

        sts bcd_entries + 3, r19
        sts bcd_entries + 2, r18
        sts bcd_entries + 1, r17
        sts bcd_entries + 0, r16

return_from_check:
    pop r18
    pop r17
    pop r16
    ret

write_mode:
    push r17
    push r18
    push XL
    push XH
    push YL
    push YH

    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)
    ldi r17, 4

    convert_display:
        ld r18, X+
        rcall hex_to_7seg
        st Y+, r18
        dec r17
        brne convert_display

    pop YH
    pop YL
    pop XH
    pop XL
    pop r18
    pop r17
    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
;hextable: .db $81, $CF, $92, $86, $CC, $A4, $A0, $8F, $80, $84
; dp, g - a
;hextable: .db $40, $79, $24, $30, $19, $12, $02, $78, $00, $10
hextable: .db $C0, $F9, $A4, $B0, $99, $92, $82, $F8, $80, $90

;***************************************************************************
;*
;* "bin2BCD8" - 8-bit Binary to BCD conversion
;*
;* This subroutine converts an 8-bit number (fbin) to a 2-digit
;* BCD number (tBCDH:tBCDL).
;*
;* Number of words  :6 + return
;* Number of cycles :5/50 (Min/Max) + return
;* Low registers used   :None
;* High registers used  :2 (fbin/tBCDL,tBCDH)
;*
;* Included in the code are lines to add/replace for packed BCD output.
;*
;***************************************************************************

;***** Subroutine Register Variables

.def    fbin    =r16        ;8-bit binary value
.def    tBCDL   =r16        ;BCD result MSD
```

```
.def    tBCDH   =r17        ;BCD result LSD

;***** Code

bin2bcd8:
    clr tBCDH       ;clear result MSD
bBCD8_1:subi    fbin,10     ;input = input - 10
    brcs    bBCD8_2     ;abort if carry set
    inc tBCDH       ;inc MSD
;---------------------------------------------------------------------------
;               ;Replace the above line with this one
;               ;for packed BCD output
;   subi    tBCDH,-$10  ;tBCDH = tBCDH + 10
;---------------------------------------------------------------------------
    rjmp    bBCD8_1     ;loop again
bBCD8_2:subi    fbin,-10    ;compensate extra subtraction
;---------------------------------------------------------------------------
;               ;Add this line for packed BCD output
;   add fbin,tBCDH
;---------------------------------------------------------------------------
    ret

;***************************************************************************
;*
;* "mpy16u" - 16x16 Bit Unsigned Multiplication
;*
;* This subroutine multiplies the two 16-bit register variables
;* mp16uH:mp16uL and mc16uH:mc16uL.
;* The result is placed in m16u3:m16u2:m16u1:m16u0.
;*
;* Number of words  :14 + return
;* Number of cycles :153 + return
;* Low registers used   :None
;* High registers used  :7 (mp16uL,mp16uH,mc16uL/m16u0,mc16uH/m16u1,m16u2,
;*                          m16u3,mcnt16u)
;*
;***************************************************************************

;***** Subroutine Register Variables

.def    mc16uL  =r16        ;multiplicand low byte
.def    mc16uH  =r17        ;multiplicand high byte
.def    mp16uL  =r18        ;multiplier low byte
.def    mp16uH  =r19        ;multiplier high byte
.def    m16u0   =r18        ;result byte 0 (LSB)
.def    m16u1   =r19        ;result byte 1
.def    m16u2   =r20        ;result byte 2
.def    m16u3   =r21        ;result byte 3 (MSB)
.def    mcnt16u =r22        ;loop counter

;***** Code

```

```
mpy16u: clr m16u3       ;clear 2 highest bytes of result
    clr m16u2
    ldi mcnt16u,16  ;init loop counter
    lsr mp16uH
    ror mp16uL

m16u_1: brcc    noad8       ;if bit 0 of multiplier set
    add m16u2,mc16uL    ;add multiplicand Low to byte 2 of res
    adc m16u3,mc16uH    ;add multiplicand high to byte 3 of res
noad8:  ror m16u3       ;shift right result byte 3
    ror m16u2       ;rotate right result byte 2
    ror m16u1       ;rotate result byte 1 and multiplier High
    ror m16u0       ;rotate result byte 0 and multiplier Low
    dec mcnt16u     ;decrement loop counter
    brne    m16u_1      ;if not done, loop more
    ret

;***************************************************************************
;*
;* "div16u" - 16/16 Bit Unsigned Division
;*
;* This subroutine divides the two 16-bit numbers
;* "dd8uH:dd8uL" (dividend) and "dv16uH:dv16uL" (divisor).
;* The result is placed in "dres16uH:dres16uL" and the remainder in
;* "drem16uH:drem16uL".
;*
;* Number of words  :19
;* Number of cycles :235/251 (Min/Max)
;* Low registers used   :2 (drem16uL,drem16uH)
;* High registers used  :5 (dres16uL/dd16uL,dres16uH/dd16uH,dv16uL,dv16uH,
;*              dcnt16u)
;*
;***************************************************************************

;***** Subroutine Register Variables

.def    drem16uL=r14
.def    drem16uH=r15
.def    dres16uL=r16
.def    dres16uH=r17
.def    dd16uL  =r16
.def    dd16uH  =r17
.def    dv16uL  =r18
.def    dv16uH  =r19
.def    dcnt16u =r20

;***** Code

div16u: clr drem16uL    ;clear remainder Low byte
    sub drem16uH,drem16uH;clear remainder High byte and carry
    ldi dcnt16u,17  ;init loop counter
d16u_1: rol dd16uL      ;shift left dividend
```

```
    rol dd16uH
    dec dcnt16u     ;decrement counter
    brne    d16u_2      ;if done
    ret         ;    return
d16u_2: rol drem16uL    ;shift dividend into remainder
    rol drem16uH
    sub drem16uL,dv16uL ;remainder = remainder - divisor
    sbc drem16uH,dv16uH ;
    brcc    d16u_3      ;if result negative
    add drem16uL,dv16uL ;    restore remainder
    adc drem16uH,dv16uH
    clc         ;    clear carry to be shifted into result
    rjmp    d16u_1      ;else
d16u_3: sec         ;    set carry to be shifted into result
    rjmp    d16u_1

;***************************************************************************
;*
;* "bin2BCD16" - 16-bit Binary to BCD conversion
;*
;* This subroutine converts a 16-bit number (fbinH:fbinL) to a 5-digit
;* packed BCD number represented by 3 bytes (tBCD2:tBCD1:tBCD0).
;* MSD of the 5-digit number is placed in the lowermost nibble of tBCD2.
;*
;* Number of words  :25
;* Number of cycles :751/768 (Min/Max)
;* Low registers used   :3 (tBCD0,tBCD1,tBCD2)
;* High registers used  :4(fbinL,fbinH,cnt16a,tmp16a)
;* Pointers used    :Z
;*
;***************************************************************************

;***** Subroutine Register Variables

.dseg
tBCD0: .byte 1  // BCD digits 1:0
tBCD1: .byte 1  // BCD digits 3:2
tBCD2: .byte 1  // BCD digits 4

.cseg
.def    tBCD0_reg = r13     ;BCD value digits 1 and 0
.def    tBCD1_reg = r14     ;BCD value digits 3 and 2
.def    tBCD2_reg = r15     ;BCD value digit 4

.def    fbinL = r16     ;binary value Low byte
.def    fbinH = r17     ;binary value High byte

.def    cnt16a  =r18        ;loop counter
.def    tmp16a  =r19        ;temporary value

;***** Code

```

```
bin2BCD16:
    push fbinL
    push fbinH
    push cnt16a
    push tmp16a


    ldi cnt16a, 16  ;Init loop counter
    ldi r20, 0x00
    sts tBCD0, r20 ;clear result (3 bytes)
    sts tBCD1, r20
    sts tBCD2, r20
bBCDx_1:
    // load values from memory
    lds tBCD0_reg, tBCD0
    lds tBCD1_reg, tBCD1
    lds tBCD2_reg, tBCD2

    lsl fbinL       ;shift input value
    rol fbinH       ;through all bytes
    rol tBCD0_reg       ;
    rol tBCD1_reg
    rol tBCD2_reg

    sts tBCD0, tBCD0_reg
    sts tBCD1, tBCD1_reg
    sts tBCD2, tBCD2_reg

    dec cnt16a      ;decrement loop counter
    brne bBCDx_2        ;if counter not zero

    pop tmp16a
    pop cnt16a
    pop fbinH
    pop fbinL
ret         ; return
    bBCDx_2:
    // Z Points tBCD2 + 1, MSB of BCD result + 1
    ldi ZL, LOW(tBCD2 + 1)
    ldi ZH, HIGH(tBCD2 + 1)
    bBCDx_3:
        ld tmp16a, -Z       ;get (Z) with pre-decrement
        subi tmp16a, -$03   ;add 0x03

        sbrc tmp16a, 3      ;if bit 3 not clear
        st Z, tmp16a        ;store back

        ld tmp16a, Z    ;get (Z)
        subi tmp16a, -$30   ;add 0x30

        sbrc tmp16a, 7  ;if bit 7 not clear
        st Z, tmp16a    ;   store back

        cpi ZL, LOW(tBCD0)  ;done all three?
    brne bBCDx_3
        cpi ZH, HIGH(tBCD0) ;done all three?
    brne bBCDx_3
rjmp bBCDx_1

;***************************************************************************
;*
;* "mpy8u" - 8x8 Bit Unsigned Multiplication
;*
;* This subroutine multiplies the two register variables mp8u and mc8u.
;* The result is placed in registers m8uH, m8uL
;*
;* Number of words  :9 + return
;* Number of cycles :58 + return
;* Low registers used   :None
;* High registers used  :4 (mp8u,mc8u/m8uL,m8uH,mcnt8u)
;*
;* Note: Result Low byte and the multiplier share the same register.
;* This causes the multiplier to be overwritten by the result.
;*
;***************************************************************************

;***** Subroutine Register Variables

.def    mc8u    =r16        ;multiplicand
.def    mp8u    =r17        ;multiplier
.def    m8uL    =r17        ;result Low byte
.def    m8uH    =r18        ;result High byte
.def    mcnt8u  =r19        ;loop counter

;***** Code


mpy8u:  clr m8uH        ;clear result High byte
    ldi mcnt8u,8    ;init loop counter
    lsr mp8u        ;rotate multiplier

m8u_1:  brcc    m8u_2       ;carry set
    add     m8uH,mc8u   ;   add multiplicand to result High byte
m8u_2:  ror m8uH        ;rotate right result High byte
    ror m8uL        ;rotate right result L byte and multiplier
    dec mcnt8u      ;decrement loop counter
    brne    m8u_1       ;if not done, loop more
    ret
```