```
1 ;*****
2 ;* Title:
                    temp_hum_sys_ext.asm
3 ;* Author:
                     Jason Chen
4 ;* Version:
5 ;* Last updated:
                    11/28/2022
                    AVR128DB48
6 ;* Target:
7;*
8 ;* DESCRIPTION
9 ;*
          Design Task 4:
10 ;*
             This program verifies the data measured from the DHT11 and
11 ;*
              decodes the information to display on the 4-Digit-7-Segment
12 ;*
             display. The pushbutton will switch between displaying humidity
13;*
             and temperature
14 ;*
15 ;* VERSION HISTORY
16 ;* 1.0 Original version
19 .equ PERIOD = 38
                    ; 38 for 2.5ms multiplex delay, 1 for 0.128ms
20
21 .dseg
22 bcd_entries:
                 .byte 4
23 led_display:
                 .byte 4
24 digit_num:
                 .byte 1
25 measured_data: .byte 5
26 mode:
                 .byte 1
27
28 .cseg
                            ; start of code segment
29 reset:
30
      jmp start
31
32 .org TCA0_OVF_vect
33
                           ; vector for overflow IRQ
      jmp ovf_mux_isr
34
35 .org PORTE_PORT_vect
36
      jmp porte_isr
                            ; vector for all PORTE pin change IRQs
37
38 start:
                     ; Configure I/O ports
39
      ldi r16, 0xFF
40
      out VPORTD DIR, r16
      ldi r16, 0xF0
41
42
      out VPORTA_DIR, r16
43
      ldi r16, 0x00
44
      cbi VPORTE_DIR, 0
45
      sbi VPORTE_DIR, 1
46
      sbi VPORTE_OUT, 1
47
      sbi VPORTE DIR, 2
48
      sbi VPORTE_OUT, 2
49
      sbi VPORTE_DIR, 3
50
      sbi VPORTE_OUT, 3
51
52
```

```
53 ; Configure TCA0
        ldi r16, TCA_SINGLE_WGMODE_NORMAL_gc
                                             ; WGMODE normal
 55
        sts TCA0 SINGLE CTRLB, r16
 56
 57
        ldi r16, TCA_SINGLE_OVF_bm
                                              ; enable overflow interrupt
 58
        sts TCA0_SINGLE_INTCTRL, r16
 59
        ldi r16, LOW(PERIOD)
 60
                                              ; set the period
 61
        sts TCA0 SINGLE PER, r16
 62
        ldi r16, HIGH(PERIOD)
        sts TCA0_SINGLE_PER + 1, r16
 63
 64
        ldi r16, TCA_SINGLE_CLKSEL_DIV256_gc | TCA_SINGLE_ENABLE_bm ; set clock and >
 65
          start timer
 66
        sts TCA0_SINGLE_CTRLA, r16
 67
 68 ; Configure interrupt request
 69
        lds r16, PORTE PINOCTRL
                                   ; set ISC for PE0 to rising edge
        ori r16, 0x02
                                   ; ISC = bit 1 for rising edge
 70
 71
        sts PORTE_PINOCTRL, r16
                                   ; update PINOCTRL register (0000 0010)
 72
        sei
 73
 74 ; Set pointers for arrays
        ldi XH, HIGH(bcd_entries)
 75
        ldi XL, LOW(bcd entries)
                                  ; X points to bcd entries[0]
 76
 77
        ldi YH, HIGH(led_display)
        ldi YL, LOW(led_display)
 78
                                 ; Y points to led_display[0]
 79
 80 ; Clear arrays
        ldi r16, 0
                                  ; load r16 with 0
 81
 82
        mov r18, r16
 83
        rcall hex_to_7seg
                                  ; load r18 with 7 segment bit pattern to show 0
        ldi r17, 4
                                  ; loop control variable
 84
 85
        clear_entries:
 86
            st X+, r16
            st Y+, r18
 87
 88
            dec r17
 89
            brne clear_entries
 90
 91 ; Program loop
 92 main_loop:
 93
        cli
 94
        rcall send_start
        rcall wait_for_response_signal
 95
        rcall get_measured_data
 96
 97
        sei
        rcall delay 2s
 98
99
        rjmp main_loop
100
102 ;* "send_start" - Send start signal
103 ;* Author:
                       Jason Chen
```

```
104 ;* Description:
       Makes DATA line 0 for 18ms then a 1 for 20us (or 40us).
107
108 send_start:
109
   rcall write_0_to_DHT11
    rcall delay 18ms
    rcall write_1_to_DHT11
111
112
    rcall delay_20us
113
     ret
114
115 write 0 to DHT11:
116
      sbi VPORTB DIR, 0
117
     ret
118
119 write_1_to_DHT11:
120 cbi VPORTB_DIR, 0
121
      ret
122
124 ;* "read_DHT11" "wait_for_0" "wait_for_1" - Read DHT11 and Wait for signal
125 ;* Author:
                Jason Chen
126 ;* Description:
127 ;*
      Read the current logic level of the DATA line. Wait and return once
     the DATA line is read to be either 0 or 1.
130
131 read_DHT11:
                   ; saves contents to bit 0 of r16
in r16, VPORTB IN
     andi r16, 0x01
133
134
     ret
135
136 wait_for_0:
137
    rcall read DHT11
138
    cpi r16, 0x00
139
    brne wait for 0
140
     ret
141
142 wait_for_1:
143 rcall read DHT11
    cpi r16, 0x01
144
    brne wait_for_1
145
146
    ret
147
;* "wait_for_response_signal" - Wait for response signal
150 ;* Author:
                Jason Chen
151 ;* Description:
152 ;*
       Wait for the response from the DHT11 by detecting a 0 and 1 for 80us
153 ;*
155
```

```
156 wait_for_response_signal:
157
       rcall wait_for_0
158
       rcall delay 80us
159
       rcall wait_for_1
160
      rcall delay_80us
161
       ret
162
164 ;* "read_DHT11_data_bit" "get_byte" "get_measured_data" - Reading data
165 ;* Author:
                    Jason Chen
166 ;* Description:
167 ;*
          Retrieve a the bit information from the DATA line. Get a full byte
168 ;*
          by reading 8 bits. Save 5 bytes of information in memory.
170
171 read_DHT11_data_bit:
                      ; records bit in bit 0 of r16
       rcall wait_for_0
172
173
       rcall delay 20us
174
       rcall wait_for_1
      rcall delay_30us
175
176
     rcall read_DHT11
                              ; record logic level of DATA line as input
177
       ret
178
179 get_byte:
                      ; records byte in r18
180
       push r17
181
       ldi r17, 8
182
       read_bit:
183
          rcall read_DHT11_data_bit
184
          1sr r16
          rol r18
185
          dec r17
186
187
          brne read_bit
188
       pop r17
189
       ret
190
191 get measured data:
192
       push r17
       ldi r17, 5
193
       ldi XH, HIGH(measured_data)
194
195
       ldi XL, LOW(measured data)
       read data:
196
          rcall get_byte
197
198
          st X+, r18
199
          dec r17
          brne read_data
200
201
       pop r17
202
203
205  ;* "delay_N" - Delay
206 ;* Author:
                   Jason Chen
207 ;* Description:
```

```
Delay 2s: ~8 million clock cycles
209 ;*
            Delay 18ms: 72000 clock cycles accounting for 2 clocks from rcall
210 ;*
            Delay 50us: 200 clock cycles accounting for 2 clocks from rcall
211 ;*
            Delay 30us: 120 clock cycles accounting for 2 clocks from rcall
212 ;*
            Delay 20us: 80 clock cycles accounting for 2 clocks from rcall
215 delay_2s:
                        ; 8,001,222 clocks (1222 extra)
216
        push r16
217
        ldi r16, 111
218
        loop_2s:
219
            rcall delay_18ms
220
            rcall delay_20us
221
            dec r16
222
            brne loop_2s
223
        pop r16
224
        ret
225
226 delay_18ms:
                        ; 72,000 clocks total (2 from rcall)
227
        push r16
                                ; 1 clock
228
        push r17
                                ; 1 clock
229
        ldi r16, 106
                                ; 1 clock (m)
        outer_18ms:
                                ; (4 + N)m - 1 = 71867 \text{ clocks}
230
231
            ldi r17, 225
                                   ; 1 clock (n)
                                    ; 3n - 1 = 674 \text{ clocks (N)}
232
            inner 18ms:
                dec r17
233
                                        ; 1 clock
                                        ; 1/2 clocks
234
                brne inner_18ms
235
            dec r16
                                    ; 1 clock
236
            brne outer 18ms
                                   ; 2/1 clocks
            rcall delay_30us
                                ; 120 clocks
237
238
        pop r17
                                ; 2 clocks
239
                                ; 2 clocks
        pop r16
240
        ret
                                ; 4 clocks
241
242 delay_80us:
                        ; 320 clocks total (2 from rcall)
243
        push r16
                                ; 1 clock
244
        ldi r16, 103
                                ; 1 clock (n)
                                ; 3n - 1 = 308 clocks
245
        loop 80us:
246
            dec r16
                                    ; 1 clock
247
            brne loop_20us
                                    ; 2/1 clocks
248
                                ; 2 clock
        nop nop
                                ; 2 clocks
249
        pop r16
250
                                ; 4 clocks
        ret
251
                        ; 200 clocks total (2 from rcall)
252 delay_50us:
253
                                ; 1 clock
        push r16
                                ; 1 clock (n)
254
        ldi r16, 63
255
        loop_50us:
                                ; 3n - 1 = 188 \text{ clocks}
256
            dec r16
                                    ; 1 clock
257
            brne loop_50us
                                    ; 2/1 clocks
                                ; 2 clock
258
        nop nop
                                ; 2 clocks
259
        pop r16
```

```
260
       ret
                            ; 4 clocks
261
                   ; 120 clocks total (2 from rcall)
262 delay_30us:
                           ; 1 clock
263
       push r16
264
       ldi r16, 34
                           ; 1 clock (n)
       loop_30us:
265
                           ; 3n - 1 = 110 \text{ clocks}
                              ; 1 clock
266
          dec r16
          brne loop_20us
                              ; 2/1 clocks
267
                            ; 2 clocks
268
       pop r16
269
                            ; 4 clocks
       ret
270
                  ; 80 clocks total (2 from rcall)
271 delay_20us:
272
       push r16
                         ; 1 clock
273
       ldi r16, 23
                           ; 1 clock (n)
274
      loop_20us:
                           ; 3n - 1 = 68 clocks
                             ; 1 clock
275
          dec r16
                            ; 2/1 clocks
276
          brne loop_20us
277
                            ; 2 clock
     nop nop
278
      pop r16
                           ; 2 clocks
279
                            ; 4 clocks
       ret
280
282 ;* "ovf_mux_isr" "porte_isr" - Interrupt subroutines
283 ;* Author:
                   Jason Chen
284 ;* Description:
285 ;* Interrupt subroutine jumps here
287
288 ovf mux isr:
289
       push r16
                                 ; save registers
290
       in r16, CPU_SREG
291
       push r16
292
      rcall multiplex_display
                                 ; multiplex display
293
       ldi r16, TCA_SINGLE_OVF_bm
                                  ; clear OVF flag
294
     sts TCA0_SINGLE_INTFLAGS, r16
295
       pop r16
                                  ; restore registers
296
       out CPU_SREG, r16
297
       pop r16
298
      reti
299
300 porte_isr:
301
       cli
                           ; clear interrupt
302
       push r16
                           ; save registers
303
       in r16, CPU_SREG
304
       push r16
305
      rcall poll_PE0
                           ; poll PE0
306
       pop r16
                            ; restore registers
307
       out CPU_SREG, r16
     pop r16
308
309
      sei
                            ; enable interrupt
310
       reti
311
```

```
312 poll_PE0:
         lds r16, PORTE_INTFLAGS
                                     ; Determine if PEO's INTFLAG is set
313
314
         sbrs r16, 0
                                     ; Check if PEO IRQ flag is set
315
         ret
                                     ; return to caller (main_loop) if not set
316
         rcall clear_irq
         rcall switch_mode
317
318
         ret
319
320 clear_irq:
321
         ldi r16, PORT_INT0_bm
         sts PORTE_INTFLAGS, r16
322
323
         ret
324
325 switch_mode:
326
         push r16
327
         lds r16, mode
328
         inc r16
329
         /*andi r16, 0x01*/
330
         cpi r16, 3
331
         brne switch_here
332
        ldi r16, 0
333
334
         switch_here:
335
             sts mode, r16
336
337
         rcall change_led
338
339
         pop r16
340
         ret
341
342 change_led:
343
         push r17
344
         cpi r16, 0x01
345
         breq led 1
346
         cpi r16, 0x02
347
         breq led_2
348
         led_0:
349
             cbi VPORTE_OUT, 1
350
             sbi VPORTE_OUT, 2
351
             sbi VPORTE OUT, 3
352
             rjmp return_from_change
353
354
         led_1:
             cbi VPORTE_OUT, 2
355
356
             sbi VPORTE_OUT, 1
357
             sbi VPORTE_OUT, 3
358
             rjmp return_from_change
359
360
         led_2:
361
             cbi VPORTE_OUT, 3
362
             sbi VPORTE OUT, 2
363
             sbi VPORTE_OUT, 1
```

```
364
365
        return_from_change:
366
        pop r17
367
        ret
368
369
371 ;*
372 ;* "multiplex_display" - Multiplex the Four Digit LED Display
373 ;*
374 ;* DESCRIPTION
375 ;*
               Updates a single digit of the display and increments the
376 ;*
               digit_num to the digit position to be displayed next.
377 ;* Author:
                      Jason Chen
378 ;* Version:
                      1
379 ;* Last Updated:
                      10/24/2022
380 ;* Target:
                      AVR128DB48
381 ;* Number of words:
382 ;* Number of cycles:
383 ;* Low registers modified: none
384 ;* High registers modified: none
385 ;*
386 ;* Parameters:
387 ;*
           led_display: a four byte array that holds the segment values
388 ;*
               for each digit of the display. led display[0] holds the
389 ;*
               segment patter for digit 0 (the rightmost digit) and so on.
390 :*
391 ;*
           digit_num: byte variable, the least significant two bits are the
392 ;*
               index of the last digit displayed.
393 ;*
394 ;* Returns: Outputs segment pattern and turns on digit driver for the next
395 ;*
               position in the display to be turned ON.
396 ;*
397 ;* Notes:
              The segments are controlled by PORTD - (dp, a through g), the
               digit drivers are controlled by PORTA (PA7 - PA4, digit 0 - 3).
398 ;*
400 multiplex_display:
401
        push r16
                      ; depopulate registers by pushing to stack
402
        push r17
403
        push r18
404
405
       ldi r16, 0xFF
406
        out VPORTD_OUT, r16
                             ; turn all segments OFF
407
        in r16, VPORTA_OUT
                             ; get current value of VPORTA
        ori r16, 0xF0
408
409
        out VPORTA OUT, r16
                             ; turn all digits OFF
410
411
       rcall check_mode
412
       rcall write_mode
413
414
        lds r16, led display + 1 ; place decimal
       ldi r17, 0x80
415
```

```
eor r16, r17
416
417
         sts led_display + 1, r16
418
419
         ldi XH, HIGH(led_display)
                                    ; X points to start of led_display array
420
         ldi XL, LOW(led_display)
421
422
         lds r16, digit num
                                 ; get current display number
423
         inc r16
424
         andi r16, 0x03
                                 ; mask for two least significant bits
425
         sts digit_num, r16
                                 ; store next digit to be displayed
426
         add XL, r16
427
                                 ; add digit number to offset to array pointer
428
         brcc PC + 2
                                 ; if no carry, skip next instruction
429
        inc XH
430
431
         ld r17, X
432
         out VPORTD_OUT, r17
                                 ; output to segment display driver port
433
         in r17, VPORTA OUT
                                 ; get current digit driver port value
434
         ldi r18, 0x10
                                 ; for next PORTA value via bit shift
435
436
         digit_pos:
437
             cpi r16, 0
                                 ; if digit number is 0, use pattern in r18
438
             breq digit_on
439
             <u>lsl</u> r18
                                 ; r18 shifted left if not 0
440
             dec r16
                                 ; decrement digit number offset
441
             rjmp digit_pos
442
443
         digit_on:
444
             eor r17, r18
                                 ; complement digit driver position
445
             out VPORTA_OUT, r17; turn selected digit ON
446
447
         pop r18
                         ; repopulate all registers with
448
         pop r17
                         ; original contents from stack
449
         pop r16
450
         ret
451
452 check_mode:
453
         push r16
454
         push r17
455
         push r18
456
                                 ; check mode
         lds r18, mode
457
         cpi r18, 0x01
458
         breq mode_1_cel
459
         cpi r18, 0x02
460
         breq mode_2_fah
461
462
         mode 0 hum:
463
             ldi r16, 0
464
             sts bcd_entries + 3, r16
465
             lds r16, measured_data + 0
466
             rcall bin2bcd8
467
             sts bcd_entries + 2, r17
```

```
sts bcd entries + 1, r16
468
469
             lds r16, measured_data + 1
470
             sts bcd entries + 0, r16
471
             rjmp return_from_check
472
473
         mode_1_cel:
474
             ldi r16, 0
475
             sts bcd_entries + 3, r16
             lds r16, measured_data + 2
476
477
             rcall bin2bcd8
478
             sts bcd_entries + 2, r17
479
             sts bcd entries + 1, r16
480
             lds r16, measured_data + 3
481
             sts bcd_entries + 0, r16
482
             rjmp return_from_check
483
484
         mode_2_fah:lm
485
             lds r16, measured_data + 2
             ldi r17, 10
486
487
             rcall mpy8u
488
             lds r16, measured_data + 3
489
             ldi r19, 0
             add r17, r16
490
491
             adc r18, r19
492
             mov r16, r17
493
             mov r17, r18
494
             ldi r18, 18
495
             rcall mpy16u
496
             ldi r16, 0x80
497
             ldi r17, 0x0C
498
             add r16, r18
499
             adc r17, r19
500
             rcall bin2bcd16
501
502
             mov r16, r13
503
             mov r17, r14
504
             mov r18, r14
505
             mov r19, r15
506
             andi r16, 0xF0
507
             andi r17, 0x0F
             andi r18, 0xF0
508
509
             andi r19, 0x0F
510
             ldi r20, 4
511
512
             right_shift:
513
                 lsr r16
514
                 lsr r18
515
                 dec r20
516
                 brne right_shift
517
518
             sts bcd entries + 3, r19
519
             sts bcd_entries + 2, r18
```

```
sts bcd_entries + 1, r17
521
           sts bcd_entries + 0, r16
522
523 return_from_check:
524
        pop r18
525
        pop r17
526
        pop r16
527
        ret
528
529 write_mode:
530
        push r17
531
        push r18
532
        push XL
533
        push XH
534
        push YL
535
        push YH
536
537
        ldi XH, HIGH(bcd entries)
                                  ; X points to bcd entries[0]
        ldi XL, LOW(bcd_entries)
538
539
        ldi YH, HIGH(led_display)
                                 ; Y points to led_display[0]
540
        ldi YL, LOW(led_display)
541
        ldi r17, 4
542
543
        convert_display:
544
           ld r18, X+
545
           rcall hex_to_7seg
546
            st Y+, r18
547
            dec r17
548
           brne convert_display
549
550
        pop YH
551
        pop YL
552
        pop XH
553
        pop XL
554
        pop r18
555
        pop r17
556
        ret
557
559 ;*
560 ;* "hex_to_7seg" - Hexadecimal to Seven Segment Conversion
561;*
562 ;* Description:
563 ;*
               Converts a right justified hexadecimal digit to the seven
564 ;*
               segment pattern required to display it. Pattern is right
565 ;*
               justified a through g. Pattern uses 0s to turn segments on ON.
566 ;*
567 ;* Author:
                       Ken Short
568 ;* Version:
                       0.1
569 ;* Last updated:
                       10/03/2022
570 ;* Target:
                       AVR128DB48
571 ;* Number of words:
```

```
572 ;* Number of cycles:
573 ;* Low registers modified: none
574 ;* High registers modified: r18
575 ;*
576 ;* Parameters: r18: hex digit to be converted
577 ;* Returns: r18: seven segment pattern. 0 turns segment ON
578 ;*
579 ;* Notes:
580 ;*
582 hex_to_7seg:
583
       push r16
584
       ldi ZH, HIGH(hextable * 2) ; set Z to point to start of table
585
       ldi ZL, LOW(hextable * 2)
586
587
       ldi r16, $00
                        ; add offset to Z pointer
588
                        ; mask for low nibble
       andi r18, 0x0F
       add ZL, r18
589
590
       adc ZH, r16
                        ; load byte from table pointed to by Z
591
       lpm r18, Z
592
       pop r16
593
       ret
594
595 ; Table of segment values to display digits 0 - F
596 ; dp, a - g
597 ;hextable: .db $01, $4F, $12, $06, $4C, $24, $20, $0F, $00, $04;, $08, $60, $31, →
      $42, $30, $38
598 ;hextable: .db $81, $CF, $92, $86, $CC, $A4, $A0, $8F, $80, $84
599 ; dp, g - a
600 ;hextable: .db $40, $79, $24, $30, $19, $12, $02, $78, $00, $10
601 hextable: .db $C0, $F9, $A4, $B0, $99, $92, $82, $F8, $80, $90
602
604 ;*
605 ;* "bin2BCD8" - 8-bit Binary to BCD conversion
606 ;*
607 ;* This subroutine converts an 8-bit number (fbin) to a 2-digit
608; * BCD number (tBCDH:tBCDL).
609 ;*
610 ;* Number of words :6 + return
611 ;* Number of cycles :5/50 (Min/Max) + return
612 ;* Low registers used
                        :None
613 ;* High registers used :2 (fbin/tBCDL,tBCDH)
614 ;*
615 ;* Included in the code are lines to add/replace for packed BCD output.
616 ;*
618
619 ;***** Subroutine Register Variables
620
           fbin
                            ;8-bit binary value
621 .def
                 =r16
622 .def
           tBCDL
                            ;BCD result MSD
                 =r16
```

```
623 .def tBCDH =r17 ;BCD result LSD
624
625 ;***** Code
626
627 bin2bcd8:
628 clr tBCDH ;clear result MSD
629 bBCD8_1:subi fbin,10 ;input = input - 10
630 brcs bBCD8_2 ;abort if carry set
631
      inc tBCDH    ;inc MSD
632 ;-----
633 ;
             ;Replace the above line with this one
            ;for packed BCD output
634 ;
635; subi tBCDH, -\$10; tBCDH = tBCDH + 10
636 ;-----
637
     rjmp bBCD8_1 ;loop again
638 bBCD8_2:subi fbin,-10 ;compensate extra subtraction
639 ;-----
      ;Add this line for packed BCD output
640 ;
641; add fbin,tBCDH
642 ;-----
643
644
646 ;*
647 ;* "mpy16u" - 16x16 Bit Unsigned Multiplication
648 ;*
649 ;* This subroutine multiplies the two 16-bit register variables
650 ;* mp16uH:mp16uL and mc16uH:mc16uL.
651 ;* The result is placed in m16u3:m16u2:m16u1:m16u0.
652 ;*
653 ;* Number of words :14 + return
654 ;* Number of cycles :153 + return
655 ;* Low registers used :None
656 ;* High registers used :7 (mp16uL,mp16uH,mc16uL/m16u0,mc16uH/m16u1,m16u2,
657 ;*
                       m16u3,mcnt16u)
658 ;*
661 ;**** Subroutine Register Variables
662
663 .def mc16uL =r16
                       ;multiplicand low byte
664 .def mc16uH =r17
                       ;multiplicand high byte
665 .def mp16uL =r18
                       ;multiplier low byte
666 .def mp16uH =r19
                       ;multiplier high byte
667 .def
       m16u0 =r18
                      ;result byte 0 (LSB)
                      ;result byte 1
668 .def
       m16u1 =r19
669 .def m16u2 =r20
                      ;result byte 2
670 .def
       m16u3 =r21
                      ;result byte 3 (MSB)
       m16u3 =r21 ;result byte :
mcnt16u =r22 ;loop counter
671 .def
672
673 ;***** Code
674
```

```
675 mpy16u: clr m16u3
                    ;clear 2 highest bytes of result
676
       clr m16u2
677
       ldi mcnt16u,16 ;init loop counter
678
       lsr mp16uH
679
      ror mp16uL
680
681 m16u 1: brcc
                           ;if bit 0 of multiplier set
                noad8
                      ;add multiplicand Low to byte 2 of res
682
       add m16u2,mc16uL
       adc m16u3,mc16uH ;add multiplicand high to byte 3 of res
683
684 noad8: ror m16u3
                      ;shift right result byte 3
685
     ror m16u2
                    ;rotate right result byte 2
                     ;rotate result byte 1 and multiplier High
686
      ror m16u1
     ror m16u0
                   ;rotate result byte 0 and multiplier Low
687
688
     dec mcnt16u
                    ;decrement loop counter
689
       brne
              m16u_1
                        ;if not done, loop more
690
691
693 ;*
694 ;* "div16u" - 16/16 Bit Unsigned Division
695 ;*
696 ;* This subroutine divides the two 16-bit numbers
697 ;* "dd8uH:dd8uL" (dividend) and "dv16uH:dv16uL" (divisor).
698 ;* The result is placed in "dres16uH:dres16uL" and the remainder in
699 ;* "drem16uH:drem16uL".
700 ;*
701 ;* Number of words :19
702 ;* Number of cycles :235/251 (Min/Max)
703 ;* Low registers used :2 (drem16uL,drem16uH)
704 ;* High registers used :5 (dres16uL/dd16uL,dres16uH/dd16uH,dv16uL,dv16uH,
705 ;*
                 dcnt16u)
706 ;*
708
709 ;***** Subroutine Register Variables
710
711 .def
         drem16uL=r14
712 .def
           drem16uH=r15
713 .def
         dres16uL=r16
714 .def
         dres16uH=r17
715 .def
         dd16uL =r16
716 .def
         dd16uH =r17
717 .def
         dv16uL =r18
718 .def
         dv16uH =r19
         dcnt16u =r20
719 .def
720
721 ;***** Code
722
                        ;clear remainder Low byte
723 div16u: clr drem16uL
724
       sub drem16uH,drem16uH;clear remainder High byte and carry
725
       ldi dcnt16u,17 ;init loop counter
726 d16u 1: rol dd16uL
                       ;shift left dividend
```

```
727
       rol dd16uH
                    ;decrement counter
728
       dec dcnt16u
729
       brne d16u 2 ;if done
730
       ret
              ; return
731 d16u 2: rol drem16uL
                        ;shift dividend into remainder
732
       rol drem16uH
733
       sub drem16uL,dv16uL ;remainder = remainder - divisor
734
       sbc drem16uH,dv16uH ;
                      ;if result negative
735
       brcc
              d16u 3
736
      add drem16uL,dv16uL ; restore remainder
737
       adc drem16uH,dv16uH
738
                      clear carry to be shifted into result
       clc
               ;
739
                     ;else
      rjmp
              d16u 1
740 d16u 3: sec
                        set carry to be shifted into result
741
       rjmp
              d16u_1
742
745 ;* "bin2BCD16" - 16-bit Binary to BCD conversion
746 ;*
747 ;* This subroutine converts a 16-bit number (fbinH:fbinL) to a 5-digit
748 ;* packed BCD number represented by 3 bytes (tBCD2:tBCD1:tBCD0).
749 ;* MSD of the 5-digit number is placed in the lowermost nibble of tBCD2.
750 ;*
751 ;* Number of words :25
752 ;* Number of cycles :751/768 (Min/Max)
753 ;* Low registers used :3 (tBCD0,tBCD1,tBCD2)
754 ;* High registers used :4(fbinL,fbinH,cnt16a,tmp16a)
755 ;* Pointers used
756 ;*
758
759 ;***** Subroutine Register Variables
760
761 .dseg
762 tBCD0: .byte 1 // BCD digits 1:0
763 tBCD1: .byte 1 // BCD digits 3:2
764 tBCD2: .byte 1 // BCD digits 4
765
766 .cseg
767 .def
                           ;BCD value digits 1 and 0
           tBCD0 reg = r13
768 .def
          tBCD1\_reg = r14
                          ;BCD value digits 3 and 2
769 .def
          tBCD2\_reg = r15
                            ;BCD value digit 4
770
771 .def
           fbinL = r16
                       ;binary value Low byte
772 .def
         fbinH = r17
                       ;binary value High byte
773
774 .def
         cnt16a =r18
                           ;loop counter
         tmp16a =r19
                           ;temporary value
775 .def
776
777 ;***** Code
778
```

```
779 bin2BCD16:
780
         push fbinL
781
         push fbinH
782
         push cnt16a
783
         push tmp16a
784
785
         ldi cnt16a, 16 ;Init loop counter
786
787
         ldi r20, 0x00
788
         sts tBCD0, r20 ;clear result (3 bytes)
789
         sts tBCD1, r20
790
         sts tBCD2, r20
791 bBCDx_1:
792
         // load values from memory
793
         lds tBCD0_reg, tBCD0
794
         lds tBCD1_reg, tBCD1
795
         lds tBCD2_reg, tBCD2
796
797
         1sl fbinL
                         ;shift input value
798
         rol fbinH
                         ;through all bytes
799
         rol tBCD0_reg
                             ;
800
         rol tBCD1_reg
801
         rol tBCD2 reg
802
         sts tBCD0, tBCD0 reg
803
804
         sts tBCD1, tBCD1_reg
805
         sts tBCD2, tBCD2_reg
806
807
         dec cnt16a
                         ;decrement loop counter
808
         brne bBCDx 2
                             ;if counter not zero
809
810
         pop tmp16a
811
         pop cnt16a
812
         pop fbinH
813
         pop fbinL
                 ; return
814 ret
815
         bBCDx 2:
         // Z Points tBCD2 + 1, MSB of BCD result + 1
816
         ldi ZL, LOW(tBCD2 + 1)
817
818
         ldi ZH, HIGH(tBCD2 + 1)
         bBCDx_3:
819
             ld tmp16a, -Z
820
                                 ;get (Z) with pre-decrement
821
             subi tmp16a, -$03
                                 ;add 0x03
822
823
             sbrc tmp16a, 3
                                 ;if bit 3 not clear
824
             st Z, tmp16a
                                 ;store back
825
826
             ld tmp16a, Z
                             ;get (Z)
827
             subi tmp16a, -$30 ;add 0x30
828
829
             sbrc tmp16a, 7 ;if bit 7 not clear
             st Z, tmp16a
830
                           ; store back
```

```
831
832
           cpi ZL, LOW(tBCD0) ;done all three?
833
       brne bBCDx 3
834
           cpi ZH, HIGH(tBCD0) ;done all three?
835
       brne bBCDx 3
836 rjmp bBCDx_1
837
839 ;*
840 ;* "mpy8u" - 8x8 Bit Unsigned Multiplication
841 ;*
342 ;* This subroutine multiplies the two register variables mp8u and mc8u.
843 ;* The result is placed in registers m8uH, m8uL
844 ;*
845 ;* Number of words :9 + return
846 ;* Number of cycles :58 + return
847 ;* Low registers used :None
848 ;* High registers used :4 (mp8u,mc8u/m8uL,m8uH,mcnt8u)
849 :*
850 ;* Note: Result Low byte and the multiplier share the same register.
851 ;* This causes the multiplier to be overwritten by the result.
852 ;*
    *************************
853
854
855 ;**** Subroutine Register Variables
856
857 .def
           mc8u
                 =r16
                            ;multiplicand
858 .def
         mp8u =r17
                            ;multiplier
859 .def
         m8uL
                =r17
                            ;result Low byte
860 .def
           m8uH
                            ;result High byte
                  =r18
         mcnt8u =r19
                            ;loop counter
861 .def
862
863 ;***** Code
864
865
866 mpy8u: clr m8uH
                        ;clear result High byte
867
       ldi mcnt8u,8
                   ;init loop counter
868
       lsr mp8u
                     ;rotate multiplier
869
870 m8u 1: brcc m8u 2
                            ;carry set
              m8uH,mc8u ; add multiplicand to result High byte
871
       add
872 m8u 2: ror m8uH
                        ;rotate right result High byte
873
       ror m8uL
                     ;rotate right result L byte and multiplier
874
       dec mcnt8u
                    ;decrement loop counter
875
                        ;if not done, loop more
       brne
              m8u_1
876
       ret
```