Amp-Hour Meter for Low Power 12v DC Systems by Andy Palm N1KSN

The original amp-hour meter project, developed by Mert Nellis W0UFO, appeared in an article in the Spring 2009 issue of "The QRP Quarterly" published by QRP ARCI. I used the original circuit except for substituting an Atmel ATtiny26 MCU and a parallel LCD in 4-bit mode for the original PICAXE-08M and serial LCD. Besides cost there was an additional advantage to this change in that using an 8 MHz crystal for the MCU clock made it unnecessary to empirically calibrate the main loop timing.

The amp-hour meter displays current, voltage, and the amp-hours used. It is designed for a maximum current of 5 amps in 12 volt DC systems with voltages between 8 and 20 volts, for example, low power communication systems.

There are two sensing circuits, one for voltage and one for current. The voltage sensing circuit is a simple voltage divider adjusted to give an analog-to-digital (ADC) count of 600 (out of 1023) when the input voltage is 12.0. The current sensing circuit uses a 0.1 ohm 3 watt resistor in the ground return circuit. The voltage across this resistor is amplified by an op amp to give an ADC count of 800 for a load drawing 4 amps. The voltage ADC count divided by 5 gives the voltage in deci-volts. The current ADC count divided by 2 gives the current in centi-amps. These are displayed in units of volts and amps on the first line of the display.

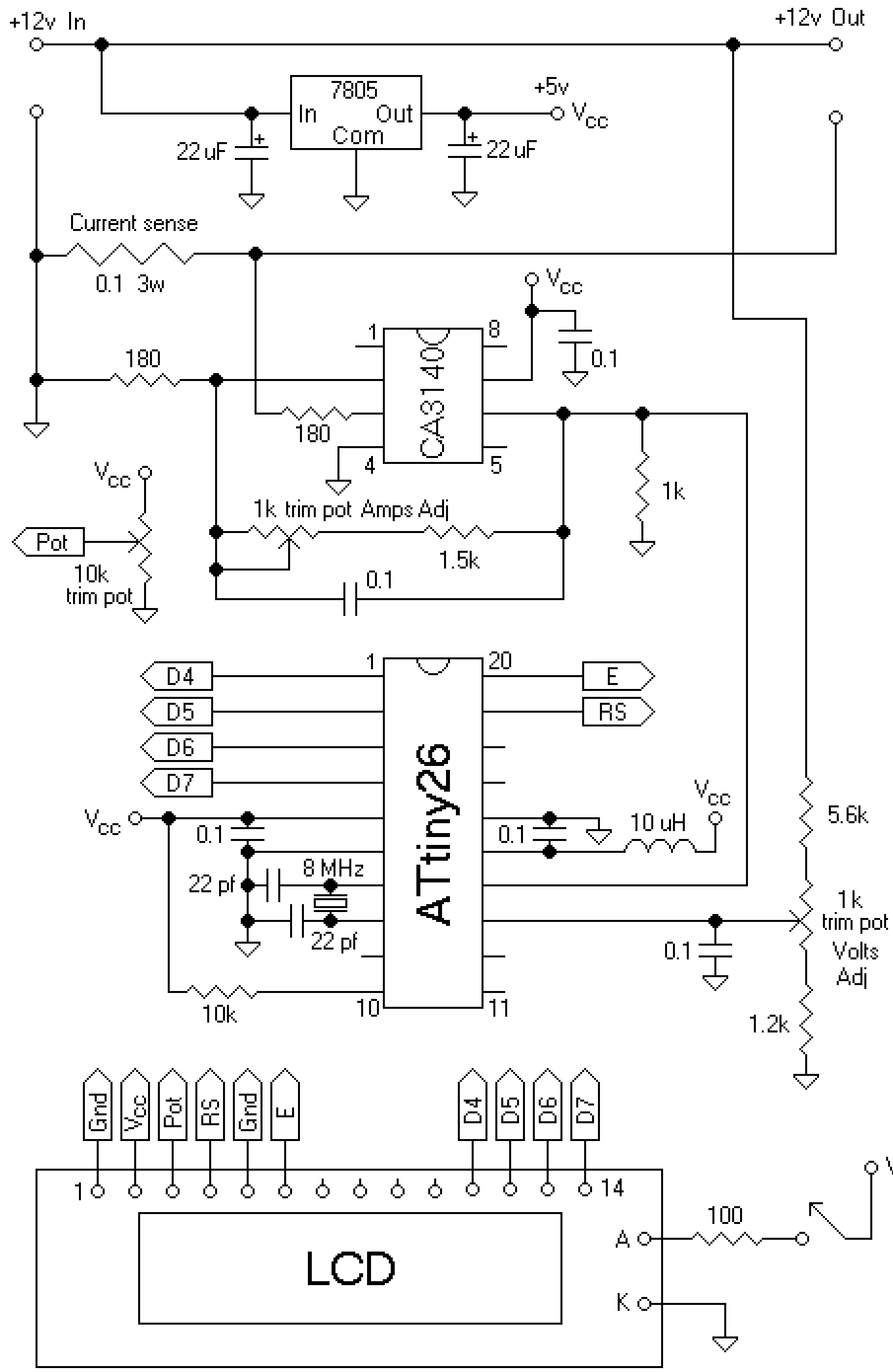
To calculate amp-hours, the main software loop is set to execute once every 360 milliseconds, which is 0.0001 hours. Thus the number of amp-hours consumed during one time interval is (ADC count)/2 centi-amps x 0.0001 hrs which is (ADC count)/2 micro-amp-hours. This value is added to a running total in a 16-bit register. When this register exceeds 1000 a second 16-bit register representing milli-amp-hours is incremented. The current value of this second register is shown on the second line of the display in units of amp-hours.

The Atmel AVR Attiny26 MCU was programmed in assembly language as previous projects had provided sections of software that could be used in this project. A great advantage to the Atmel AVR products are that software can be easily transferred from one model to another, as they all share the same instruction set and register structure.

A simple round-robin software structure is used. At the top of the main loop a small loop is executed until a flag is set by the TIMER0 interrupt service routine (ISR). This ISR reloads TIMER0 for a 4 ms overflow interval and counts 90 of these intervals to yield a 90 x 4 ms = 360 ms = 0.0001 hr tick. Both ADCs are run in one-shot, 10-bit mode once per tick. The display is updated once per tick.

The 12 character by 2 line LCD used for this project, the Crystalfontz CFAH1202A-YYH-JP, was obtained surplus from All Electronics. The connections are the same for any LCD that follows the common Hitachi 44780 standard. The more common size 2x16 LCD can be used without modification (but you can easily expand the display to 16 characters). The two display lines are set up in the data SRAM area. The display digits are calculated from the corresponding binary values by the same division macro used in a previous frequency counter project.

This amp-hour meter will be used to monitor battery consumption while operating low power amateur radio equipment in the field or while using emergency battery power at my home station.



Amp-Hour Meter for 12 v DC System

Original schematic by Mert Nellis W0UFO in Spring 2009 issue of "The QRP Quarterly" magazine, published by QRP ARCI. This version modified for ATtiny26 MCU and parallel LCD in 4-bit mode.

Andy Palm N1KSN
February 2010

Schematic of Amp-Hour Meter for Low Power 12 Volt Systems

```
;---------------------------------------------------------------------
; asmAHrMeter.asm - Amp-hour meter with LCD display
;
; Converts a current sense circuit voltage in ADC3 and increments a
; running total which represents amp-hours used.  Converts a voltage
; divider voltage in ADC4 to represent the voltage used.  A character
; LCD shows voltage, current, and amp-hours.
;
; This project is based on an article in the Spring 2009 issue of
; "The QRP Quarterly" by Mert Nellis W0UFO.  The original current and
; voltage sensing circuits are used here.  However, the original
; published project used a PICAXE-08M and a serial LCD.  Using an AVR
; ATtiny26 and a regular LCD allows quicker updates of the display.
;
; Hardware:
;   - ATtiny26 with external crystal at 8 MHz.  Fuse settings are
;     High: F1, Low: CC
;
;   - Standard 2x16 or 2x12 character LCD in 4-bit data mode
;
;   - A 0.1 ohm 3 watt current sensing resistor with CA3140 op amp
;     circuit is input for a current reading ADC (see schematic).
;
;   - A voltage divider is input for a voltage reading ADC (see
;     schematic).
;
; AVR pin assignments:
;     PB0:3 - LCD data lines
;     PB4:5 - 8 MHz crystal
;     PA0   - LCD E line
;     PA1   - LCD RS line
;     PA4   - ADC3 input from current sensing circuit
;     PA5   - ADC4 input from voltage divider circuit
;
; Notes:
;   - A simple round-robin structure is used.  At the top of the main
;     loop a small loop is executed until a flag is set by the
;     TIMER0 interrupt service routine.  This ISR reloads TIMER0 for
;     a 4 ms overflow interval and counts 90 of these intervals to
;     yield a 90 x 4 ms = 360 ms = 0.0001 hr tick.  Both ADCs are run
;     in one-shot, 10-bit mode once per tick.  The display is updated
;     once per tick.
;
;   - The current sensing circuit that is input to ADC3 is scaled so
;     that an ADC count of 800 corresponds to 4 amps.  With a tick
;     of 0.0001 hr, a current of 4 amps means 0.0004 amp-hours = 400
;     micro-amp-hours have been used during the tick interval.  Since
;     this corresponds to an ADC count of 800, dividing the ADC count
;     by 2 gives both the current in centi-amps and the number of
;     micro-amp-hours used during the tick interval (assuming a
;     constant current).  These values are accumulated in two 16-bit
;     registers, one for micro-amp-hours and one for milli-amp-hours.
;     The latter is shown on the LCD along with the current in amps.
;
;   - The voltage sensing circuit that is input to ADC4 is scaled so
;     that an ADC count of 600 corresponds to 12 volts.  Thus if the
;     ADC count is divided by 5 the result is in units of decivolts.
;     This value is displayed as volts on the LCD.
;
;   - The LCD subroutines are unsophisticated, but functional.  Since
;     PB7 is used as the RESET pin on the ATtiny26 during serial
;     programming, the E and R/S lines were moved to PORTA, the same
;     port as the ADC pins.  However, these digitial lines are not
;     changed during ADC conversions and so should not contribute to
;     ADC analog noise.
;
; Andy Palm
; 2010.02.28
;
;---------------------------------------------------------------------
;------------------ includes, defines, equates ------------------------
.nolist
.include "tn26def.inc"
.list
```

```
.equ  FREQ          = 8000000     ; Clock frequency in Hz

; System tick equates for 90 x 4 ms = 360 ms main loop tick
.equ  TICK_PRESCALE   = 256       ; Tick timer prescaler value
.equ  TICKS_PER_S     = 250       ; Ticks per sec for 4 ms inner tick
.equ  TIMER0_START    = 256 - (FREQ/(TICKS_PER_S*TICK_PRESCALE))
.equ  TICK_OUTER_CNT  = 90        ; Outer loop count

; LCD subroutine equates
.equ  CNT_200US     = FREQ/20000  ; Count value for 200 us delay loop
.equ  MLT_5MS       = 25          ; Multiplier for 5 ms delay
.equ  MLT_200MS     = 40          ; Multiplier for 200 ms delay
.equ  LCD_E         = 0           ; LCD E line bit number
.equ  LCD_RS        = 1           ; LCD RS line bit number
.equ  LCD_Port      = PORTB       ; Port for LCD data lines
.equ  LCD_DDR       = DDRB        ; DDR for LCD data lines
.equ  LCD_Port_Cntl = PORTA       ; Port for LCD E and RS lines
.equ  LCD_DDR_Cntl  = DDRA        ; DDR for LCD E and RS lines

; Display line lengths
.equ  LINE1_LGT = 12              ; Length of display line 1
.equ  LINE2_LGT = 11              ; Length of display line 2

; Register variables
.def  AH_micro_L    = r2          ; Micro-amp-hour total
.def  AH_micro_H    = r3
.def  AH_milli_L    = r4          ; Milli-amp-hour total
.def  AH_milli_H    = r5

.def  tempa         = r16         ; General purpose registers
.def  tempb         = r17
.def  tempc         = r18

.def  tick_counter  = r19         ; Tick outer counter
.def  tick_flag     = r20         ; Flags outer counter turned over

.def  remain_L      = r22         ; Low byte of division remainder
.def  remain_H      = r23         ; High byte of division remainder

.def  temp_L        = r24         ; General 16-bit register for ADC
.def  temp_H        = r25         ; calculations


;------------------ SRAM assignments ----------------------------------
.dseg
.org SRAM_START

; Two lines of text for display
Line1:      .byte   1     ; Leading blank
V_Tens:     .byte   1     ; Voltage tens digit
V_Ones:     .byte   1     ; Voltage ones digit
            .byte   1     ; Decimal point
V_Tenths:   .byte   1     ; Voltage tenths digit
            .byte   2     ; "V "
I_Ones:     .byte   1     ; Current ones digit
            .byte   1     ; Decimal point
I_Tenths:   .byte   1     ; Current tenths digit
I_Hndths:   .byte   1     ; Current hundredths digit
            .byte   1     ; "A"

Line2:      .byte   1     ; Leading blank
AH_Tens:    .byte   1     ; Amp-hours tens digit
AH_Ones:    .byte   1     ; Amp-hours ones digit
            .byte   1     ; Decimal point
AH_Tenths:  .byte   1     ; Amp-hours tenths digit
            .byte   6     ; " A-Hrs"

;------------------ macros -------------------------------------------
.macro  Calc_Digit
; Divide a 16-bit integer by a constant 16-bit integer with an 8-bit
; quotient.  Used with successive calls to get digits for display
; values.  The remainder is put into the dividend registers for the
; next call.
;
; Method of division taken from "Electrical Engineering 101" by
```

```
; Darren Ashby, 2006, Elsevier/Newnes, p. 122
;
; Call is
;         Calc_Digit dividend_H, dividend_L, H, L, quotient_reg
; where
;         dividend_H = register holding high byte of dividend
;         dividend_L = register holding low byte of dividend
;         H, L = High, low bytes of divisor as constants
;         quotient_reg = register to store quotient
;
  clr   @4                      ; Clear registers for division by
  clr   remain_L
  clr   remain_H
  ldi   tempb, 16               ; Load counter with number of bits
Calc_Digit_A:
  lsl   @4
  lsl   @1                      ; Rotate dividend left into remainder
  rol   @0
  rol   remain_L
  rol   remain_H
  push  remain_L                ; Save copy of remainder
  push  remain_H
  subi  remain_L, @3            ; Subtract divisor from remainder
  sbci  remain_H, @2
  brsh  Calc_Digit_B            ; Compare remainder and divisor
  pop   remain_H                ; Remainder < divisor so restore
  pop   remain_L                ; remainder and do nothing else
  rjmp  Calc_Digit_C
Calc_Digit_B:
  pop   tempa                   ; Remainder >= divisor so discard
  pop   tempa                   ; old remainder and keep new value
  inc   @4                      ; Add one to result digit
Calc_Digit_C:
  dec   tempb
  brne  Calc_Digit_A            ; Continue through all bits of dividend
  mov   @1, remain_L            ; Remainder is dividend in next step
  mov   @0, remain_H

.endmacro

;------------------ interrupt vectors ---------------------------------
.cseg
.org 0x0000
  rjmp  Reset       ; Reset service
  reti              ; EXT_INT0 external interrupt
  reti              ; PIN_CHANGE pin change
  reti              ; TIM1_CMP1A Timer1 compare match 1A
  reti              ; TIM1_CMP1B Timer1 compare match 1B
  reti              ; TIM1_OVF Timer1 overflow
  rjmp  Tick_Count  ; TIM0_OVF Timer0 overlow
  reti              ; USI_STRT USI start handler
  reti              ; USI_OVF USI overflow handler
  reti              ; EE_RDY eeprom ready
  reti              ; ANA_COMP analog comparator
  reti              ; ADC ADC conversion complete

;------------------ device initialization -----------------------------
Reset:
  ldi   tempa, RAMEND         ; Set up stack
  out   SP, tempa

; LCD setup
  rcall LCD_Init

; ADC setup
  ldi   tempa, (0<<REFS1)|(0<<REFS0)|(0<<ADLAR)|(1<<MUX1)|(1<<MUX0)
  out   ADMUX, tempa  ; AVCC ref, right adj, start with ADC3
  ldi   tempa, (1<<ADEN)|(1<<ADSC)|(0<<ADFR)|(0<<ADIE) \
              |(1<<ADPS2)|(1<<ADPS1)|(0<<ADPS0)
  out   ADCSR, tempa  ; Enable & initialize, no int, /64 prescaler

; Set up system tick outer counter and TIMER0 as system tick inner
; counter
  clr   tick_flag
  ldi   tick_counter, TICK_OUTER_CNT  ; Initialize tick outer counter
```

```
  ldi   tempa, TIMER0_START           ; Initialize tick inner counter
  out   TCNT0, tempa
  ldi   tempa, (1<<CS02)|(0<<CS01)|(0<<CS00)
  out   TCCR0, tempa                  ; Start with 256x prescaler
  ldi   tempa, (1<<TOIE0)             ; Enable TIMER1 overflow int
  out   TIMSK, tempa

  sei                                 ; Enable global interrupts

;------------------ main program -------------------------------------
Main:
  rcall Set_Display_Lines   ; Set up display messages in RAM

  clr   AH_micro_L          ; Zero micro-AH total
  clr   AH_micro_H
  clr   AH_milli_L          ; Zero milli-AH total
  clr   AH_milli_H

Main_Loop:
; Wait for outer tick counter to turn over
Wait_for_Tick:
  tst   tick_flag
  breq  Wait_for_Tick
  clr   tick_flag

; Get current reading from ADC3 and add to running total
  ldi   tempa, 0b00000011   ; Set input to ADC3
  out   ADMUX, tempa
  sbi   ADCSR, ADSC         ; Start a conversion
  sbis  ADCSR, ADIF         ; Wait until conversion complete
  rjmp  PC-1
  cbi   ADCSR, ADIF         ; Clear conv complete flag bit

  in    temp_L, ADCL        ; Get ADC value
  in    temp_H, ADCH
  lsr   temp_H              ; Divide ADC value by 2 to get micro-AH
  ror   temp_L              ; and centi-amps value
  add   AH_micro_L, temp_L  ; Add to micro-AH total
  adc   AH_micro_H, temp_H

; Calculate the display digits for current and write to RAM
  rcall Calc_I_Digits

  ldi   tempa, LOW(1000)    ; Micro-AH total >= 1000 ?
  ldi   tempb, HIGH(1000)
  cp    AH_micro_L, tempa
  cpc   AH_micro_H, tempb
  brlo  AH_Update_Done      ; No, done
  sub   AH_micro_L, tempa   ; Yes, subtract 1000 from micro-AH total
  sbc   AH_micro_H, tempb
  ldi   tempa, 1            ; and increment milli-AH total
  clr   tempb
  add   AH_milli_L, tempa
  adc   AH_milli_H, tempb
AH_Update_Done:

; Calculate display digits for amp-hours and write to RAM
  rcall Calc_AH_Digits

; Get voltage from ADC4
  ldi   tempa, 0b00000100   ; Set input to ADC4
  out   ADMUX, tempa
  sbi   ADCSR, ADSC         ; Start a conversion
  sbis  ADCSR, ADIF         ; Wait until conversion complete
  rjmp  PC-1
  cbi   ADCSR, ADIF         ; Clear conv complete flag bit

  in    temp_L, ADCL        ; Get ADC result
  in    temp_H, ADCH

  rcall Divide_By_5         ; Divide ADC result by 5 for decivolts
  mov   temp_L, tempc       ; Put decivolts in 16-bit register for
  clr   temp_H              ; display digit calculations

; Calculate display digits for voltage and write to RAM
```

```
  rcall Calc_V_Digits

; Write to display
  rcall Write_Display

  rjmp  Main_Loop

;------------------ interrupt service routines ------------------------
; Counter for 360 ms system tick
Tick_Count:
  push  tempa                    ; Save contents of tempa
  ldi   tempa, TIMER0_START      ; Load inner counter start value
  out   TCNT0, tempa
  in    tempa, SREG              ; Save status register since dec used
  push  tempa
  dec   tick_counter
  brne  Tick_Count_Done
  ldi   tick_counter, TICK_OUTER_CNT  ; Outer counter turned over
  ldi   tick_flag, 1                  ; Set flag
Tick_Count_Done:
  pop   tempa
  out   SREG, tempa              ; Restore status register
  pop   tempa                    ; Restore original contents of tempa
  reti

;------------------ subroutines --------------------------------------
;---------------------------------------------------------------------
; Set of subroutines for using LCD in 4-bit setup.
; Uses one port.  Bits 4 and 5 are not used, but written as zeros.
; Allows use of PORTB on ATtiny26 even though PB4:5 are used for
; external xtal oscillator, since writing to these two bits does not
; affect operation.
;
; The following must be set up in the calling routine:
;
; .equ  FREQ          = 8000000     ; Clock freq in Hz (e.g, 8 MHz)
; .equ  CNT_200US     = FREQ/20000  ; Count value for 200 us delay loop
; .equ  MLT_5MS       = 25          ; Multiplier for 5 ms delay
; .equ  MLT_200MS     = 40          ; Multiplier for 200 ms delay
; .equ  LCD_E         = 0           ; LCD E line bit number
; .equ  LCD_RS        = 1           ; LCD RS line bit number
; .equ  LCD_Port      = PORTB       ; Port for LCD data lines
; .equ  LCD_DDR       = DDRB        ; DDR for LCD data lines
; .equ  LCD_Port_Cntl = PORTA       ; Port for LCD E and RS lines
; .equ  LCD_DDR_Cntl  = DDRA        ; DDR for LCD E and RS lines
; .def  tempa         = r16         ; Reg for byte to LCD (e.g., r16)
;
; Delay subroutines are good up to a 13 MHz clock.
;
; Toggle delay for E line can be 2 cycles for 4 MHz, 4 cycles for 8
; MHz.  A dummy subroutine call is used here which gives 7 cycles and
; so is good up to 13 MHz clock.
;---------------------------------------------------------------------
; Delay 200 microseconds
LCD_Wait_200us:
  push  ZH
  push  ZL
  ldi   ZH, HIGH(CNT_200US)
  ldi   ZL, LOW(CNT_200US)
LCD_Wait_200US1:
  sbiw  ZL, 1
  brne  LCD_Wait_200US1
  pop   ZL
  pop   ZH
  ret

;---------------------------------------------------------------------
; Delay 5 milliseconds
LCD_Wait_5ms:
  push  tempa
  ldi   tempa, MLT_5MS
LCD_Wait_5ms1:
  rcall LCD_Wait_200US
  dec   tempa
  brne  LCD_Wait_5ms1
```

```
  pop   tempa
  ret

;--------------------------------------------------------------------
; Delay 200 milliseconds
LCD_Wait_200ms:
  ldi   tempa, MLT_200MS
LCD_Wait_200ms1:
  rcall LCD_Wait_5ms
  dec   tempa
  brne  LCD_Wait_200ms1
  ret

;--------------------------------------------------------------------
; Toggle LCD E line to clock in data
LCD_E_Toggle:
  sbi   LCD_Port_Cntl, LCD_E
  rcall LCD_E_Toggle_Wait
  cbi   LCD_Port_Cntl, LCD_E
  ret

;--------------------------------------------------------------------
; Seven cycle delay for E line toggle
LCD_E_Toggle_Wait:
  ret

;--------------------------------------------------------------------
; Write command byte in tempa to LCD (RS line low)
LCD_Cmmd:
  push  tempa               ; Save two copies of command byte
  push  tempa
  swap  tempa               ; Prepare to send 4 high bits
  andi  tempa, 0x0F
  out   LCD_Port, tempa     ; Load 4 high bits with E=0, RS=0
  clr   tempa
  out   LCD_Port_Cntl, tempa
  rcall LCD_E_Toggle        ; Clock out 4 high bits

  pop   tempa               ; Similarly send 4 low bits from copy
  andi  tempa, 0x0F
  out   LCD_Port, tempa
  clr   tempa
  out   LCD_Port_Cntl, tempa
  rcall LCD_E_Toggle

  pop   tempa               ; Use 2nd copy to determine proper delay
  andi  tempa, 0b11111100   ; Check for all zeros in high 6 bits
  breq  LCD_Cmmd1           ; If all zeros, do long delay
  rcall LCD_Wait_200us      ; 200 us delay for all other commands
  ret
LCD_Cmmd1:
  rcall LCD_Wait_5ms        ; Long delay for clear display and return
  ret                       ; home commands

;--------------------------------------------------------------------
; Initialize LCD
LCD_Init:
  in    tempa, LCD_Port
  ori   tempa, 0b00001111     ; Set LCD data port pins as outputs
  out   LCD_DDR, tempa
  in    tempa, LCD_DDR_Cntl
  ori   tempa, (1<<LCD_E)|(1<<LCD_RS)
  out   LCD_DDR_Cntl, tempa   ; Set LCD E and RS port pins as outputs

  rcall LCD_Wait_5ms          ; Wait 20 ms for LCD warm-up
  rcall LCD_Wait_5ms
  rcall LCD_Wait_5ms
  rcall LCD_Wait_5ms

  ldi   tempa, 0x03           ; Initialization bits for LCD
  out   LCD_Port, tempa
  cbi   LCD_Port_Cntl, LCD_RS ; Clear RS bit
  cbi   LCD_port_Cntl, LCD_E  ; Clear E bit
  rcall LCD_E_Toggle          ; Toggle 4 bits
  rcall LCD_Wait_5ms          ; Wait 5 ms
```

```
  ldi   tempa, 0x03           ; Repeat twice more with 200 us wait
  out   LCD_Port, tempa
  rcall LCD_E_Toggle
  rcall LCD_Wait_200us

  ldi   tempa, 0x03
  out   LCD_Port, tempa
  rcall LCD_E_Toggle
  rcall LCD_Wait_200us

  ldi   tempa, 0x02           ; LCD to 4-bit mode
  out   LCD_Port, tempa
  rcall LCD_E_Toggle
  rcall LCD_Wait_200us

  ldi   tempa, 0b00101000 ; 2 lines, 5x7 font
  rcall LCD_Cmmd

  ldi   tempa, 0b00000001 ; Clear LCD
  rcall LCD_Cmmd

  ldi   tempa, 0b00000110 ; Move cursor after each char write
  rcall LCD_Cmmd

;  ldi   tempa, 0b00001110 ; Turn on LCD and enable cursor
  ldi   tempa, 0b00001100 ; Turn on LCD and disable cursor
  rcall LCD_Cmmd

;  ldi   tempa, 0b10000000 ; Move cursor to start of line 1
;  ldi   tempa, 0b11000000 ; Move cursor to start of line 2

  ret

;-----------------------------------------------------------------------
; Write byte in tempa to LCD (RS line high)
LCD_Write:
  push  tempa                 ; Save copy of byte
  swap  tempa                 ; Prepare to send 4 high bits
  andi  tempa, 0x0F
  out   LCD_Port, tempa       ; Load 4 high bits
  sbi   LCD_Port_Cntl, LCD_RS  ; Set RS bit high
  cbi   LCD_Port_Cntl, LCD_E   ; Set E bit low
  rcall LCD_E_Toggle          ; Clock out 4 high bits

  pop   tempa                 ; Similarly send 4 low bits from copy
  andi  tempa, 0x0F
  out   LCD_Port, tempa
  sbi   LCD_Port_Cntl, LCD_RS
  rcall LCD_E_Toggle

  rcall LCD_Wait_200us        ; 200 us delay
  ret

;-----------------------------------------------------------------------
; Set up display lines in RAM
Set_Display_Lines:
  ldi   ZH, HIGH(Line1)           ; Load Line 2 address
  ldi   ZL, LOW(Line1)
  ldi   tempa, ' '                ; Leading blank
  st    Z+, tempa
  ldi   tempa, '0'                ; Volts tens digit
  st    Z+, tempa
  ldi   tempa, '0'                ; Volts ones digit
  st    Z+, tempa
  ldi   tempa, '.'                ; Decimal point
  st    Z+, tempa
  ldi   tempa, '0'                ; Volts tenths digit
  st    Z+, tempa
  ldi   tempa, 'V'                ; Voltage label
  st    Z+, tempa
  ldi   tempa, ' '                ; Blank
  st    Z+, tempa
  ldi   tempa, '0'                ; Current ones digit
  st    Z+, tempa
```

```
  ldi   tempa, '.'                  ; Decimal point
  st    Z+, tempa
  ldi   tempa, '0'                  ; Current tenths digit
  st    Z+, tempa
  ldi   tempa, '0'                  ; Current hundredths digit
  st    Z+, tempa
  ldi   tempa, 'A'                  ; Current label
  st    Z, tempa

  ldi   ZH, HIGH(Line2)             ; Load Line 1 address
  ldi   ZL, LOW(Line2)
  ldi   tempa, ' '                  ; Leading blank
  st    Z+, tempa
  ldi   tempa, '0'                  ; Amp-hour tens digit
  st    Z+, tempa
  ldi   tempa, '0'                  ; Amp-hour ones digit
  st    Z+, tempa
  ldi   tempa, '.'                  ; Decimal point
  st    Z+, tempa
  ldi   tempa, '0'                  ; Amp-hour tenths digit
  st    Z+, tempa
  ldi   tempa, ' '                  ; Blank
  st    Z+, tempa
  ldi   tempa, 'A'                  ; Amp-hour label
  st    Z+, tempa
  ldi   tempa, '-'
  st    Z+, tempa
  ldi   tempa, 'H'
  st    Z+, tempa
  ldi   tempa, 'r'
  st    Z+, tempa
  ldi   tempa, 's'
  st    Z, tempa

  ret

;-----------------------------------------------------------------------
; Calculate and store display digits for current
Calc_I_Digits:
; Divide value by 100 to get ones digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x00, 0x64, tempc
  ldi   ZH, HIGH(I_Ones)
  ldi   ZL, LOW(I_Ones)
  ldi   tempb, '0'
  add   tempc, tempb
  st    Z, tempc
; Divide value by 10 to get tenths digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x00, 0x0A, tempc
  ldi   ZH, HIGH(I_Tenths)
  ldi   ZL, LOW(I_Tenths)
  ldi   tempb, '0'
  add   tempc, tempb
  st    Z, tempc
; Remainder goes to hundredths digit, store in RAM
  ldi   ZH, HIGH(I_Hndths)
  ldi   ZL, LOW(I_Hndths)
  ldi   tempb, '0'
  add   remain_L, tempb
  st    Z, remain_L

  ret

;-----------------------------------------------------------------------
; Calculate and store display digits for amp-hour total
Calc_AH_Digits:
  mov   temp_L, AH_milli_L
  mov   temp_H, AH_milli_H
; Divide total by 10000 to get tens digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x27, 0x10, tempc
  ldi   ZH, HIGH(AH_Tens)
  ldi   ZL, LOW(AH_Tens)
  cpi   tempc, 0
  breq  PC+4
  ldi   tempb, '0'
  add   tempc, tempb
```

```
  rjmp  PC+2
  ldi   tempc, ' '            ; Display leading zero as blank
  st    Z, tempc
; Divide total by 1000 to get ones digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x03, 0xE8, tempc
  ldi   ZH, HIGH(AH_Ones)
  ldi   ZL, LOW(AH_Ones)
  ldi   tempb, '0'
  add   tempc, tempb
  st    Z, tempc
; Divide total by 100 to get tenths digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x00, 0x64, tempc
  ldi   ZH, HIGH(AH_Tenths)
  ldi   ZL, LOW(AH_Tenths)
  ldi   tempb, '0'
  add   tempc, tempb
  st    Z, tempc

  ret

;---------------------------------------------------------------------
; Divide ADC result by 5 (with rounding) to get decivolts in tempc
Divide_By_5:
  Calc_Digit  temp_H, temp_L, 0x00, 0x05, tempc
  cpi   remain_L, 3     ; Round up if remainder 3 or more
  brlo  PC+2
  inc   tempc
  ret

;---------------------------------------------------------------------
; Calculate and store display digits for voltage
Calc_V_Digits:
; Divide value by 100 to get tens digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x00, 0x64, tempc
  ldi   ZH, HIGH(V_Tens)
  ldi   ZL, LOW(V_Tens)
  cpi   tempc, 0
  breq  PC+4
  ldi   tempb, '0'
  add   tempc, tempb
  rjmp  PC+2
  ldi   tempc, ' '            ; Display leading zero as blank
  st    Z, tempc
; Divide value by 10 to get ones digit and store in RAM
  Calc_Digit  temp_H, temp_L, 0x00, 0x0A, tempc
  ldi   ZH, HIGH(V_Ones)
  ldi   ZL, LOW(V_Ones)
  ldi   tempb, '0'
  add   tempc, tempb
  st    Z, tempc
; Remainder goes to tenths digit, store in RAM
  ldi   ZH, HIGH(V_Tenths)
  ldi   ZL, LOW(V_Tenths)
  ldi   tempb, '0'
  add   remain_L, tempb
  st    Z, remain_L

  ret

;---------------------------------------------------------------------
; Write two lines to display
Write_Display:
  ldi   tempa, 0b10000000           ; Move cursor to start of line 1
  rcall LCD_Cmmd
  ldi   ZH, HIGH(Line1)             ; Load Line 1 address
  ldi   ZL, LOW(Line1)
  ldi   tempb, LINE1_LGT            ; Write Line 1 to LCD
Write_Display_Loop1:
  ld    tempa, Z+
  rcall LCD_Write
  dec   tempb
  brne  Write_Display_Loop1

  ldi   tempa, 0b11000000           ; Move cursor to start of line 2
  rcall LCD_Cmmd
```

```
  ldi   ZH, HIGH(Line2)             ; Load Line 2 address
  ldi   ZL, LOW(Line2)
  ldi   tempb, LINE2_LGT            ; Write Line 2 to LCD
Write_Display_Loop2:
  ld    tempa, Z+
  rcall LCD_Write
  dec   tempb
  brne  Write_Display_Loop2

  ret

;------------------ rom constants and tables -------------------------

;------------------ eeprom --------------------------------------------
.eseg
.org 100
```