```
;* ADC writes to STK500 LEDs on PORTB and saves to mega32 SRAM, THEN sends data via USART to PC. ADC is Triggered by timer1 OCR1B.
;* ADC writes to STK500 LEDs on PORTB based on 200Hz sampling interval, which is set by Timer1
;* The ADC is directly triggered by timer1 OCR1B i.e. no need to wait for timer, only wait for data sampled in ADC
;* Samples are stored in SRAM then sent via USART to PC
;* ATTENTION!!!! MPU CLOCK AT 8MHz for 200Hz sampling to be correct
;*************************************************************************
; STK500 default board  clock is 3.68MHz and requires jumper OSCSEL on the STK500 to be between 1 and 2
; STK500 default 3.68MHz board clock can change from STK500 in ISP mode (HW settings) window
;
; STK500 default board clock is NOT used since ATmega32 comes with fuse SUT_CKSEL for internal RC Osc 1MHz i.e.
; the ATmega32 internally generates the 1MHz clock. This can be changed from the STK500 in ISP mode (Fuses) window
; To change any of the above RS232 MUST be connected to STK500 RS232 CTRL (not to JTAG board)

.include "m32def.inc"

;***** Global register variables (remember R26-R31 is XYZ 16bit regs)
.def        temp        =R20
.def        cntr1       =R21
.def        cntr2       =R22
.def        temp1       =R16
.def        temp2       =R23
.def        temp3       =R24
.def        tempL       =R19
.def        tempH       =R18
.def        leds        =R17

.equ        DATASTART   =$0060; First internal SRAM address to start saving ADC samples (0x00 to 0x5f are register address space)
.equ        DATAEND  =RAMEND - DATASTART        ; max value is DATAEND, min is DATASTART+1. Leave 0x60 bytes for the STACK

.org 0x000

reset:                  ;Main program entry point on reset starts at the end of interrupt vector table

                        ldi temp, high(RAMEND)
                        out SPH, temp                   ;Set Stack Pointer to top of SRAM (for ATmega32 it is $85f)
                        ldi temp, low(RAMEND)
                        out SPL, temp

                        ldi         temp,0b11111111  ;turn OFF all STK500 LEDs (do this to avoid LEDs turning on for 2usec)
                        out         PORTB,temp

                        ldi         temp,0b11111111  ;set PB0-7 as outputs (STK500 LEDs)
                        out         DDRB,temp

                        ldi         temp,0b00000000  ;disable pull up resistor on each input pin of PORTA
                        out         PORTA,temp
```

```
                  ldi       temp,0b00000000  ;set PORTA as INPUT for ADC in PA0
                  out       DDRA,temp


                  ldi  ZL,low(DATASTART)         ; Initialize Z register, which holds SRAM address where next ADC sample will be stored
                  ldi  ZH,high(DATASTART)

; ************************************ ADC Initialization ************************************
                  ldi temp, 0b01100000                    ;REFS1bit7 and REFS0bit6 set to 01 for AVCC 5V. Set ADLARbit5=1 for 10 bit Left Adjusted Output.
                  out       ADMUX, temp                   ;Set MUX4bit5-MUX0bit0 to 00000 for ADC) single ended input selection

; ************************************ CTC Timer Initialization ************************************

          ldi temp2, 0b01000000          ; WGM for CTC mode (i.e. counter counting from 0x0000 to OC1RA) set to zero WGM10bit0 and WGM11bit1
          out TCCR1A, temp2              ; set COM1A1bit7 and COM1A0bit6 to 01 to enable toggling of OC1A (i.e. PD5) when OCR1A value is reached

          ldi temp2, 0b00001000          ; WGM for CTC mode (i.e. counter counting from 0x0000 to OC1RA) set WGM12bit3=1 and WGM13bit4=0
          out TCCR1B, temp2              ; Also, set CS12bit2, CS11bit1 and CS10bit0 to zero to stop counting before loading TCNT1 and OCR1A

          ldi tempH, high(41493-1)       ; high byte of upper limit to count upto is decimal 40000=5000us x 8 (since clock is 8MHz) for 200Hz fs (41493 to also compensate for error in CLK; see above)
          out OCR1AH, tempH              ; load OCR1A high byte
          ldi tempL, low(41493-1)        ; low byte of upper limit to count upto is decimal 40000=5000us x 8 (since clock is 8MHz) for 200Hz fs (41493 to also compensate for error in CLK; see above)
          out OCR1AL, tempL              ; load OCR1A low byte

          ldi tempH, 0x00                ; Timer 1 will count upwards from 0x0000
          ldi tempL, 0x00
          out TCNT1H, tempH              ; load timer high byte FIRST since it is stored internally in a temporary location until the low byte is written
          out TCNT1L, tempL              ; now that high byte is loaded, load timer low byte

          ldi temp2, 0b00010000          ; clear timer 1 overflow flag OCFR1A1bit4 by writing a logic 1 to it
          out TIFR, temp2

          ldi temp2, 0b00001001          ; WGM for CTC mode (i.e. counter counting from 0x0000 to OC1RA) set WGM12bit3=1 and WGM13bit4=0
          out TCCR1B, temp2              ; Also, set CS12bit2, CS11bit1 and CS10bit0 to 001 starts counting

   waittimer:
          in temp, TIFR
          sbrs temp, OCF1A                         ; skip next instruction if OCF1A flag is set i.e. after the timer reaches OCR1A
          rjmp waittimer                                    ; loop while OCF1A flag is not set

;=================================== ADC Conversion ===================================================
                                startConversion:
                                       ldi temp, 0b11000000                ;ADENbit7=1 to enable ADC, ADSCbit6=1 to start conversion, ADPS2bit2-ADPS0=000 for prescaler set to 1
                                       out       ADCSRA, temp              ;Set MUX4bit5-MUX0bit0 to 00000 for ADC) single ended input selection
                                waitadc:
                                       sbic ADCSRA, ADSC                   ;ADSC bit = 0 after the ADC conversion is complete
                                       rjmp waitadc                        ;loop until the ADCS bit = 0
                                       ldi temp2, 0b00010000           ; clear timer 1 overflow flag OCFR1A1bit4 by writing a logic 1 to it
                                       out TIFR, temp2

                                       in      tempL, ADCL             ;Must read FIRST ADCL and THEN ADCH otherwise new conversion does not start
                                       in      tempH, ADCH

                                       st      Z+, tempH               ;store ADC 8bit value to SRAM (via register Z) and icrease Z

                                       ldi temp, 0xff                  ;turn 0s into 1s and 1s into 0s since STK500 LEDs turn on when PBx is zero
                                       eor temp, tempH                 ;turn 0s into 1s and 1s into 0s since STK500 LEDs turn on when PBx is zero
                                       out     PORTB, temp

                                       ldi temp, low(DATAEND+1)        ;check the end of SRAM. If not, continue with the conversion.
                                       cpse temp, ZL                   ;cpse >> compare, skip if equal
                                       rjmp    waittimer
                                       ldi temp, high(DATAEND+1)
                                       cpse temp, ZH
                                       rjmp    waittimer

       ;=====================================================================================================

       USART_Init:                     ; on STK500 connect jumpers between RXDandPD0  TXDandPD1 for SPARE RS232 to work
                                       ; Set baud rate UBRR=0008 i.e. 115200bps for 8MHz system clock and doubleUARTspeed=1
                  ldi temp, 0x00
                  out UBRRH, temp
                  ldi temp, 0x08
                  out UBRRL, temp
                                                  ; Enable double speed on UART (for less clock error -3.5%)
                  ldi temp, 0b00000010            ; U2X=1
                  out UCSRA, temp

                                                  ; Enable receiver and transmitter and part of 8data
                  ldi temp, 0b00011000            ; RXEN=1 TXEN=1 UCSZ2=0
                  out UCSRB, temp

                                                  ; Set frame format: 8data, 1stop bit
                  ldi temp, 0b10000110            ; URSEL=1 UMSEL=0 UPM10=00 USBS=0 UCSZ10=11 UCPOL=0
                  out UCSRC,temp

       ;******************************CR / LF***********************************************
                  ldi temp, 13                    ; ascii CR (Carriage Return > Return to the beginning of the current line)
                  rcall USART_Transmit

                  ldi temp, 10                    ; ascii LF (Line Feed > Downward to the next line)
                  rcall USART_Transmit

       ;*******************************Start *********************************************
                  ldi temp, 'S'
                  rcall USART_Transmit

                  ldi temp, 't'
                  rcall USART_Transmit
                                                  ; send Start message to USART to mark data send begining
                  ldi temp, 'a'
                  rcall USART_Transmit

                  ldi temp, 'r'
                  rcall USART_Transmit

                  ldi temp, 't'
                  rcall USART_Transmit

       ;*****************************CR / LF********************************************
                  ldi temp, 13                    ; ascii CR
                  rcall USART_Transmit

                  ldi temp, 10                    ; ascii LF
                  rcall USART_Transmit

       ;*********************************SRAM Mapping with ASCII table*****************************

                  ldi     XL,low(DATASTART)       ;set X register to SRAM DATASTART
                  ldi     XH,high(DATASTART)

       loop_chars:
                  ldi     tempH, 0x00             ; 8 bit conversion so high byte is always zero
                  ld      tempL, X+               ; 8 bit ADC value from SRAM goes to low byte


                  ldi     ZH,high(2*(adc2ascii))  ; Find offset for msg to be sent to USART
                  ldi     ZL,low(2*(adc2ascii))   ; by comparing tempH and tempL to two bytes in prog memory
       findADCvalue:
                  lpm                             ; Load 1 byte from program memory (in address Z)  into r0
                  eor r0, tempH
                  breq checkLow                   ; high byte matches so check low byte
                  adiw ZL,8                       ; Else, increase Z register by 8 since bin/ascii data are stored in 8 consecutive bytes
                  rjmp findADCvalue
       checkLow:
                  adiw    ZL,1                    ; Increase Z register
                  lpm                             ; Load byte from program memory into r0
                  eor r0, tempL
                  breq msgFound                   ; if found print this msg to USART
                  adiw ZL,7                       ; Else, increase Z register and continue searching the table in cseg
                  rjmp findADCvalue
```

```
;**********************************Send message to USART************************************
msgFound:
                adiw    ZL,1                    ; Increase Z registers

                lpm     temp, Z+                ; load ascii value to send to serial port
                rcall USART_Transmit

                lpm     temp, Z+                ; load ascii value to send to serial port
                rcall USART_Transmit

                lpm     temp, Z+                ; load ascii value to send to serial port
                rcall USART_Transmit

                lpm     temp, Z+                ; load ascii value to send to serial port
                rcall USART_Transmit

                ldi temp, 13                    ; ascii CR
                rcall USART_Transmit

                ldi temp, 10                    ; ascii LF
                rcall USART_Transmit

;************************************check the end of SRAM*******************************
                ldi temp, low(DATAEND)
                cpse temp, XL
                rjmp    loop_chars
                ldi temp, high(DATAEND)
                cpse temp, XH
                rjmp    loop_chars


                ldi temp, 'E'                   ; send End message to USART to mark data send end
                rcall USART_Transmit

                ldi temp, 'n'
                rcall USART_Transmit

                ldi temp, 'd'
                rcall USART_Transmit

                ldi temp, 13                    ; ascii CR
                rcall USART_Transmit

                ldi temp, 10                    ; ascii LF
                rcall USART_Transmit

;****************************************************************************************
```

```
done:
                    ldi temp, 0xAA
                    out         PORTB, temp
                    rjmp done                         ; when SRAM is full stop sampling


USART_Transmit:                                       ; transmits to USART the byte in register temp
                                                      ; Wait for empty transmit buffer
                    sbis UCSRA,UDRE                   ;check flag UDRE. If it is set, the UDR is empty, so we can write data
                    rjmp USART_Transmit

                    out UDR,temp
                    ret


.cseg               ;code segment
.org 0x500          ;place code and constants at specific locations in Program Memory
; format is adcH, adcL, volts ascii, dot ascii, mVolts 1st decimal ascii, mVolts 2nd decimal ascii, zero, zero
.include "adc2ascii5volt8bit.txt"
```