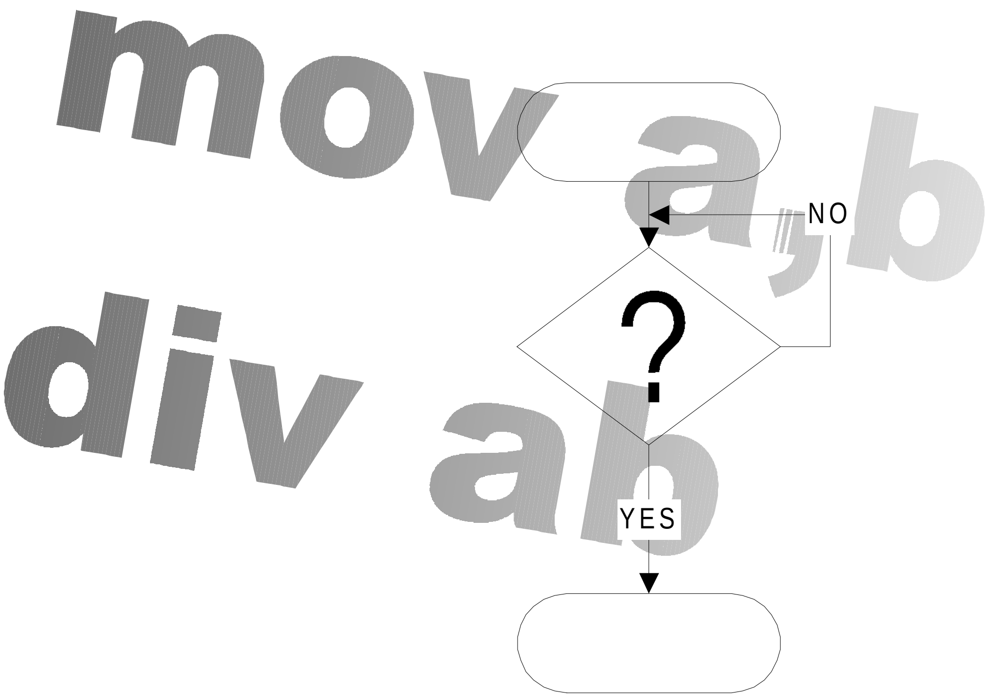
Lab-Report
Microprocessors

Digital Voltage Meter (DVM)



<u>Name:</u> Dirk Becker
<u>Course:</u> BEng 2
<u>Group:</u> A
<u>Student No.:</u> 9801351
<u>Date:</u> 05/May/1999

UNIVERSITY of EAST LONDON

1. Contents

2. Introduction

Microelectronics is increasingly pervading all aspects of industry, education and the home. A leading example of microelectronic techniques is the microprocessor, and as its use increases the need for knowledge and understanding will also grow.
The microprocessor lab was designed to give an overview over the programming of such a microprocessor system. Therefor a Digital Voltage Meter was to implement on the UELMON 51.

3. The Project

With the UELMON system a digital voltmeter with the following specifications was to implement:

- Input Voltage Range: 0..5 Volts
- Display: 2½ digits
- Refresh Rate: 500ms +/- 1ms

The project was divided into 5 different sections (Milestones). These sections were as follows:

Section 1
Read data from AD converter and write it to DA for determination of dynamic range and I/P - O/P relationship of the DA and AD.

Section 2
Implement a 500ms timing loop, for reducing the sample rate to $\frac{2}{s}$ (2 samples per second).

Section 3
Convert the hexadecimal data from the AD converter to 3 ASCII digits.

Section 4
Write the converted ASCII data to the LCD.

Section 5
Refinements.

4. The Main program

First an overall flowchart of the voltmeter program was developed. It puts the different task into a chronological order. First the applied voltage has to be converted to a digital value and written to the DAC. Then the hexadecimal ADC value has to be converted to a 3 digit ASCII, which can be written to the LC-Display. Then the program has to wait until the 500ms are finished. Therefore the timer has to be started before reading from the ADC.
Figure 1 shows the resulting flowchart.

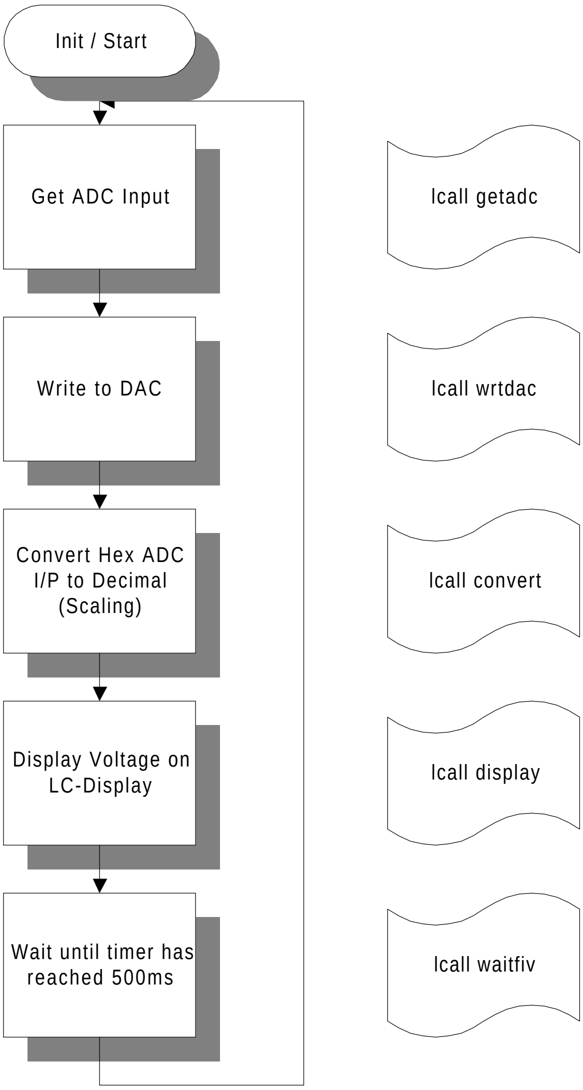


figure 1 - The main program (Overall flowchart)

5. The Lab

a) Milestone 1 – Read from ADC and write to DAC

First section of the Lab was to implement a short program, which was able to read the content of the Analogue to Digital Converter and write it to the Digital to Analogue Converter.
The I/P value was always printed on the screen via the pint8u function of the UELMON.
Pint8u prints automatically the actual content of the Accumulator to the serial interface as a decimal number (0..255).
Later the program was divided into the subroutines GETADC and WRTDAC, which are called from the main program.

i. Read from A/D converter

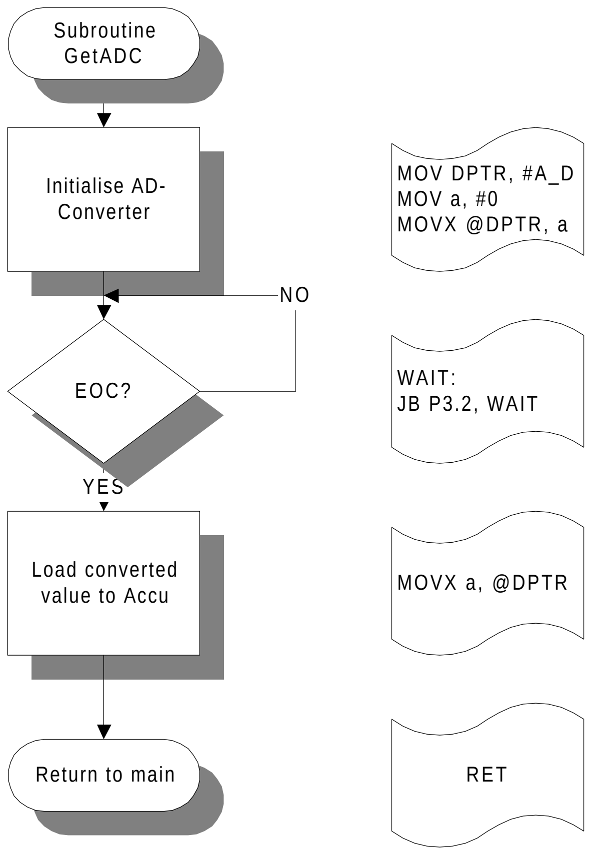


figure 2 - Subroutine GetADC

Figure 2 shows the flowchart for the subroutine, which reads from the ADC and writes it to the Accumulator.
The resulting code is shown on the right hand side of the flowchart.
The A/D has to be started by writing a dummy value to it, and then the program wait until the conversion is done and writes the resulting value to the Accu.

ii. Write to D/A converter

As a next part of the first section the read value of the A/D had to be written to the D/A converter and the characteristics of both were to be obtained.

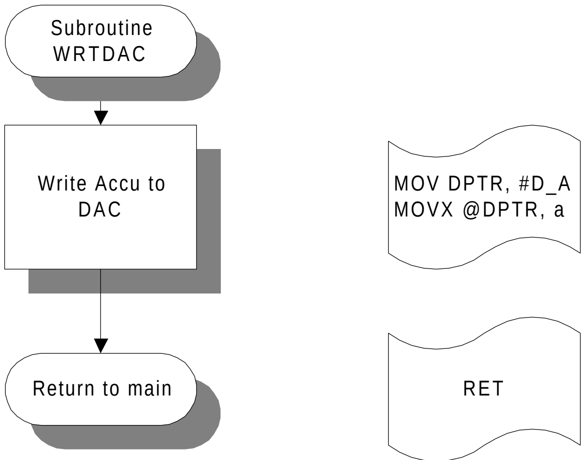


figure 3 – Subroutine Writ to DAC

iii. AD and DA converter characteristics

Input to ADC/V	Screen Out (Dec - 0..255)	Output of DA/V
0	**0**	**0**
0.1	**1**	**0.01**
0.3	**16**	**0.16**
1	**52**	**0.52**
2	**104**	**1.04**
3	**156**	**1.56**
4	**207/208**	**2.07**
4.90	**254**	**2.54**
4.91	**254/255**	**2.55**
4.92	**255**	**2.55**

From this table the characteristics of the ADC and DAC can be obtained:

ADC:

$$\text{Resolution} = \frac{\text{max.Voltage}}{\text{stepsize}} = 20\text{mV} \qquad \text{Range: } 0..4.92\text{V}$$

$$\text{Quantisation error} = \frac{\text{Resolution}}{2} = 10\text{mV}$$

$(4.90\text{V} \Rightarrow 254\text{dec.}$

$\underline{-\,4.92\text{V}} \Rightarrow 255\text{dec.}$

$\underline{\underline{= 20\text{mV}}})$

DAC:

$$0..255 \qquad (\text{Range}: 0..2.55\text{V})$$

$$0..2.55\text{V} \Rightarrow \text{Resolution}\ \frac{2.55\text{V}}{255} = 10\text{mV}$$

$$Q_{\text{Err}} = \frac{\text{Res}}{2} = 5\text{mV}$$

<u>b) Milestone 2 – 500ms timing loop</u>

For sampling the incoming voltage with a sampling rate of ½ second the program must provide a timer, which is started before reading from the ADC and halts the program, until 500ms are done and the next input value can be read.
The 500ms timing loop consists of a) The ISR (Figure 4) and b) an external counter routine (Figure 5).
The timer is started before the ADC starts to work and is preloaded with a defined value. If the timer overflows the 8051 generates an interrupt, which forces the program to continue at the address of the interrupt vector (UELMON=080b). With every overflow a second variable (n) is decremented and the timer again set to the defined value. When the helping counter n has reached 0, the program starts continuing at the beginning.

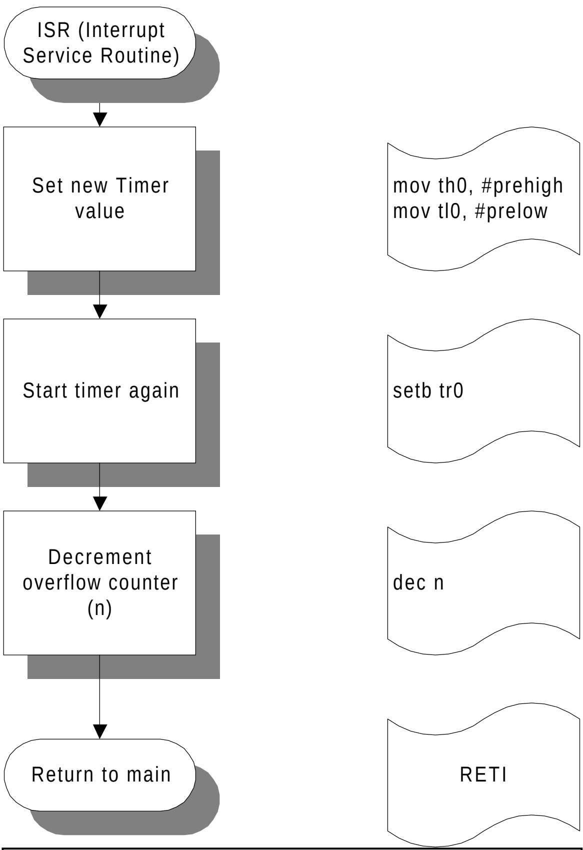


figure 4 - ISR

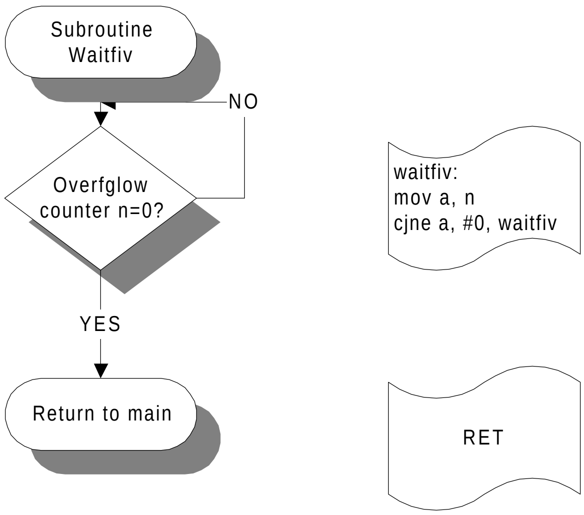


figure 5 - Overflow counter routine

The overflow counter routine is called at the end of the main program in order to secure a proper timing.

Calculation of the Timer preset-value:

The Timer counts without preset from 0 up to 65535 and generates then an interrupt. For these 65535 counts the timer needs about 65ms, which means for a delay of 500ms the timer has to be restarted $\frac{500\text{ms}}{65\text{ms}} = 7.69$ times to provide 500ms delay.
Also the timer can be restarted 8 times, but then it must be presetted by a value, to do the same timing (500ms). Therefore the timer has not to start with 0, because the timing loop would increase 500ms (8*65ms= 520ms).
Hence the counter only should count $\frac{7.6923 \times 65536}{8} = 63015$ turns.

Proof:

$$8 \times 63.015\text{ms} = 504\text{ms}$$
$$\Downarrow$$
$$8 \times 62.5\text{ms} = 500\text{ms}$$
$$\Downarrow$$
$$\text{TMR0} : 0..62500 \text{ counting}$$

but TMR0 counts up – Hence 65536-62500=$(3036)_{10}$

$(3036)_{10}$=\$0BDC $\Rightarrow$ \$0B=Highbyte and \$DC=Lowbyte of TMR0

c) Milestone 3 – Scaling to 3 digit ASCII

The LC-Display needs a 3 digit value in ASCII format to work correct. So the HEX value coming from the ADC must be converted (scaled) in an appropriate way.

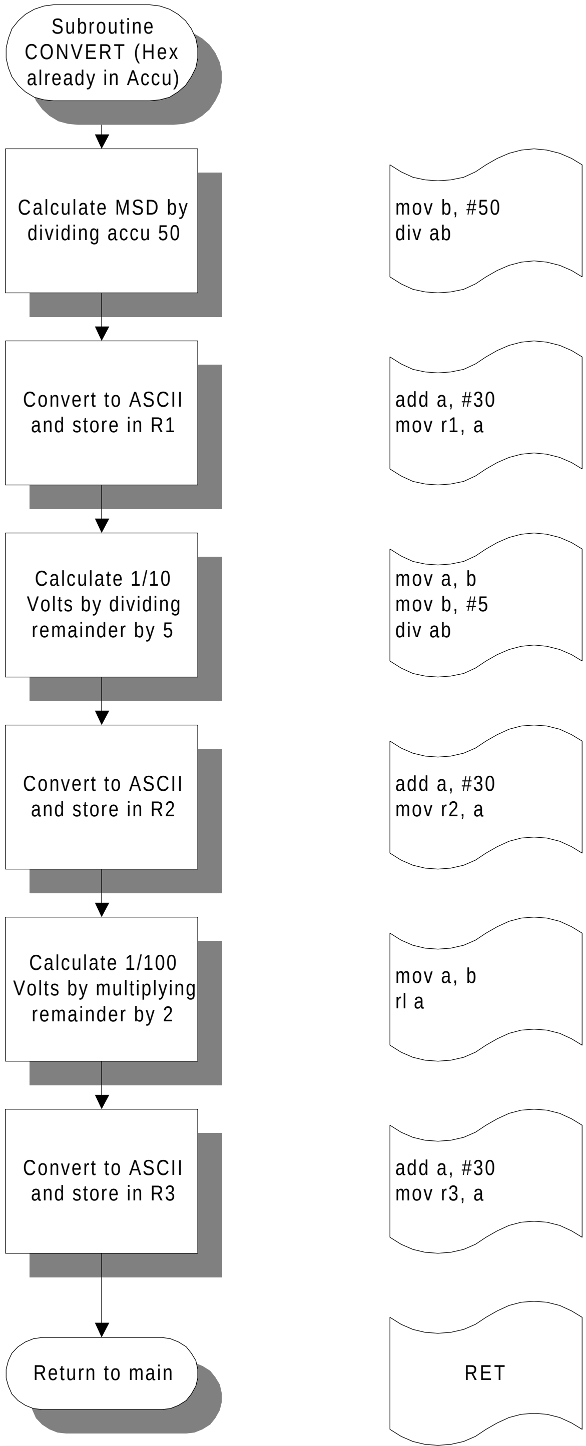


figure 6

Figure 6 shows the flowchart of the scaling procedure. The different digits are stored in the registers R1..R3.

d) Milestone 4 – Output to LC Display

The last part of the lab (Milestone 4) was to display the contents of the internal register R1..R3 of the 8051 in a “Voltage meter” appropriate form. The initialisation routine for the display was copied from the Lab-examples, because of the lack of documentation on the LCD.
The LC-Display is controlled serial by port 3 of the 8255 and the LCD data are applied via port 2 of the 8355. To obtain a proper work of the display it has to be initialised in a special way (subroutine INITL).

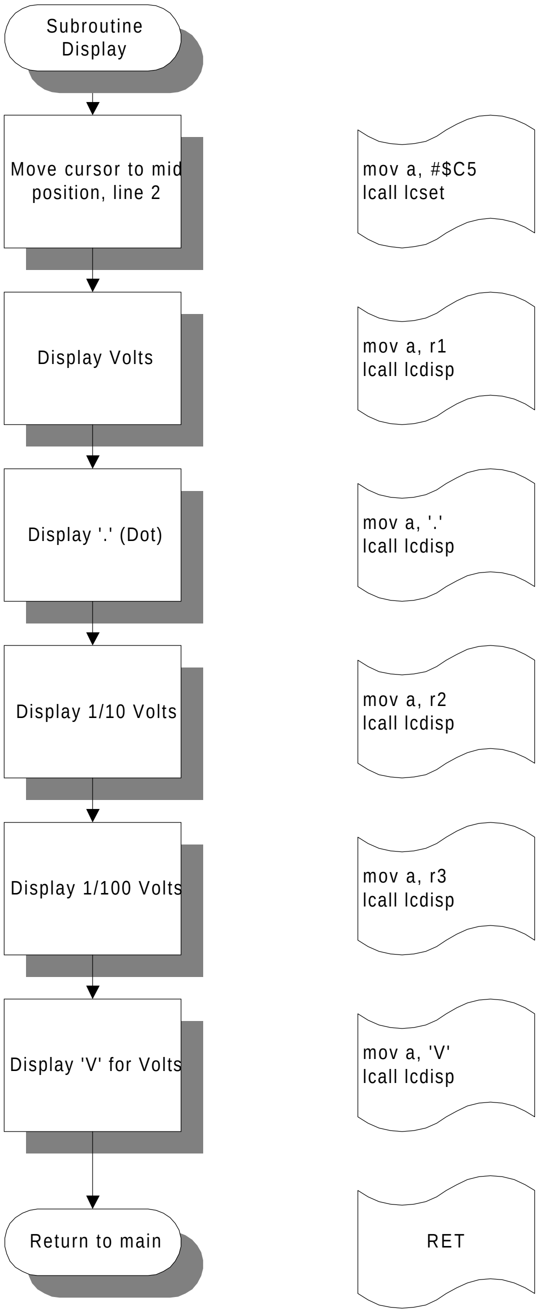


figure 7 - Subroutine Display

Figure 7 shows the subroutine to display the stored data on the LC-Display. First the cursor is moved to a mid position in the second row via the LCSET function, and then the registers are output in the format #.##V, like on a usual DVM.

6. Conclusion

The UELMON is a mighty tool to develop and test programs for the Intel 8051 processor. After assembling and simulating the code it can be direct downloaded and tested on the UELMON. With its implemented routines for accessing the serial port debugging and error searching is made very easy.
It is very important to use a simulator for developing assembler code, because otherwise some errors can't be found.
In a time, where microprocessors become more and more important and are in use in every day's life every engineer should be able to use them, because of their high flexibility.

7. Appendix

e) The complete code

```
; Read ADC Input
; Write it to serial and DAC, with a sampling rate of 1/2 second
; and convert it to decimals
;
; Dirk Becker, 9801351, >= 1-MAR-1999
;
;

         .EQU  A_D,$E000          ; Set address of AD Converter
         .EQU  D_A,$C000          ; Set address of DA Converter
         .EQU  pint8u,$004D       ; Set address of print ACC to serial
         .EQU  newline,$0048      ; Set address off serial CR/LF
         .equ  prelow, $dc        ; Preload TMR0 LowByte
         .equ  prehigh, $0b       ; Preload TME0 HighByte
         .equ  n, $30             ; Overflow counter
         .equ  p8255, $4000       ; Address of the Port-Interface 8255

         .org $8000
         ljmp init

         .org $800b               ; ISR Start vector
isr:     mov   th0, #prehigh      ; Presets TMR0 High
         mov   tl0, #prelow       ; and low byte
         setb  tr0                ; Starts TMR0 again
         dec n                    ; Decrement helping counter
         reti                     ; back and wait for next interrupt

init:

         setb  ea                 ; Enable interupts
         setb  et0                ; with TMR0 overflow interrupt
         mov   a, tmod            ; ( TMR1 must
         anl   a,#$f0             ; not be
         orl   a,#$01             ; changed )
         mov   tmod, a            ; and set to 16-Bit Counter mode

         lcall initl              ; Call LCD - Init



START:

         mov   th0, #prehigh      ; Presets TMR0 High
         mov   tl0, #prelow       ; and low byte
         setb  tr0                ; Starts TMR0
         mov   n, #8              ; Set helping counter n to 8
         lcall getadc             ; Read ADC
         lcall wrtdac             ; write it to DAC
         lcall convert            ; convert it to ascii
         lcall display            ; and print it on the LCD
         lcall waitfiv            ; wait until 500ms are done

         LJMP  start              ; Goto Start – forever
```

```
; *****************************************************************
; * Subroutine GETADC                                             *
; * Reads content from ADC and writes it to the accu              *
; *****************************************************************
getadc:  MOV   DPTR,  #A_D       ; Set Datapointer to Adress of ADC
         MOV   A,     #0         ; Load #0 to Acc
         MOVX  @DPTR, A          ; load Acc to Address of ADC
WAIT:    JB    P3.2,  WAIT       ; Wait until End of Conversion (Port3,
Pin2)
         MOVX  A,     @DPTR      ; Load Result to Acc
         ; LCALL pint8u            ; Print content of ACC to serial
         ; LCALL newline           ; Send CR/LF to serial
         ret

; *******************************************************************
; * Subroutine WRTDAC                                               *
; * Reads content from accu and writes it to the DAC                *
; *******************************************************************
wrtdac:  MOV   DPTR,  #D_A       ; Set Datapointer to Address of DAC
         MOVX  @DPTR, A          ; Move Content of Acc to DAC for Output
         ret

; *******************************************************************
; * Subroutine WAIT                                                 *
; * Waits until 500ms are done                                      *
; *******************************************************************
waitfiv: mov a, n
         cjne a, #0, waitfiv     ; are 500ms
         ret                     ; done? - go back

; *******************************************************************
; * Subroutine Convert                                              *
; * Converts the accu into decimals and stores the                  *
; * results int r1, r2 and r3 (MSB ... LSB)                         *
; *******************************************************************
convert: mov b, #50              ; divide Accu
         div ab                  ; by 50 - Remainder to register B
         add a, #$30             ; convert accu to ASCII
         mov r1, a               ; and store it to R1 --> Volts
         ;LCALL pint8u            ; Print content of ACC to serial

         mov a, b                ; divide remainder
         mov b, #5               ; by 5
         div ab
         add a, #$30             ; convert accu to ASCII
         mov r2, a               ; store it in R2 --> 1/10 Volts
         ;LCALL pint8u            ; Print content of ACC to serial

         mov a, b                ; Multiply remainder
         rl a                    ; by 2 --> 1/100 Volts
         add a, #$30             ; convert accu to ASCII
         mov r3, a
         ;LCALL pint8u            ; Print content of ACC to serial
         ;lcall newline
         ret                     ; done? - go back
```

```
; ******************************************************************
; * Subroutine Display                                             *
; * Prints the content or R1 .. R3 to the LCD                      *
; ******************************************************************
display:
          mov a, #$C5
          lcall lcset

          mov a, r1
          lcall lcdisp

          mov a,#'.'
          lcall lcdisp

          mov a, r2
          lcall lcdisp

          mov a, r3
          lcall lcdisp

          mov a,#'V'
          lcall lcdisp

         ret



;**********************************************
; initialisation of LCD                       *
;**********************************************

initl:
   mov dptr,#P8255+3    ;8255 setup register
   mov a,#80h
   movx @dptr,a         ;port C is o/p
   mov dptr,#setlcd
init1:
   mov a,#0
   movc a,@a+dptr
   CJNE A,#0,init2
   ret
init2:
   inc dptr
   lcall LCset
   ljmp init1
```

```
;*********************************************************************
; LCD write routines. LCdisp sends the (ASCII) char contained in A. *
;                     LCset sends the command contained in A        *
;              Note - The DPTR is preserved                         *
;*********************************************************************

lcdisp:
        setb p1.5                       ;setup for data
        ajmp sendit
LCset:
        clr p1.5                        ;setup for command
sendit:
        push dpl
        push dph
        mov dptr,#P8255+2                       ;address of 8255 port c
        movx @dptr,a                    ;send data to 8255
        clr p1.6                        ;write enabled
        nop
        nop
        nop
        setb p1.7                       ;clock the data
        nop
        nop
        nop
        acall delay
        clr p1.7
        setb p1.6
        acall delay
        pop dph                         ;restore dptr, Note: last in
        pop dpl                         ;is first out when using the stack
        ret

delay:  mov r0,#0FFh
        djnz r0,*
        ret
;****************************************************************
setlcd:
        .db $3C,$06,$0E,$01,$81,$81,$00

        .end
```