Mikrocontrollertechnik Protokolle

Protokolle zu den Versuchen 1 bis 8 des Moduls Mikrokontrollertechnik and der JLU Gießen

Julian Bergmann, Jan Mühlhans, Katja Kleeberg

30. Oktober 2013

Inhaltsverzeichnis

Inhaltsverzeichnis **I**

Versuch 1 **1**
Versuchsziel . . . 1
Quellcode . . . 1
Flussdiagramm . . . 2

Versuch 2 **3**
Versuch 2.1 . . . 3
2.1.1 Versuchsziel . . . 3
2.1.2 Quellcode . . . 3
2.1.3 Flussdiagramm . . . 4
Versuch 2.2 . . . 5
2.2.1 Versuchsziel . . . 5
2.2.2 Quellcode . . . 5
2.2.3 Flussdiagramm . . . 6

Versuch 3 **7**
Versuch 3.1 . . . 7
3.1.1 Versuchsziel . . . 7
3.1.2 Quellcode . . . 7
Versuch 3.2 . . . 10
3.1.1 Versuchsziel . . . 10
3.1.2 Quellcode . . . 10

Versuch 4 **12**
Versuchsziel . . . 12
Quellcode . . . 12
Flussdiagramm . . . 15

Versuch 5 **16**
Versuchsziel . . . 16
Quellcode . . . 17
Flussdiagramm . . . 19

Versuch 6 **20**
Versuchsziel . . . 20
Quellcode . . . 21
Flussdiagramm . . . 24

Versuch 7 **25**
Versuchsziel 25
Quellcode 25
Flussdiagramm 28

Versuch 8 **29**
Versuchsziel 29
Quellcode 31
Flussdiagramm 40

Versuch 1

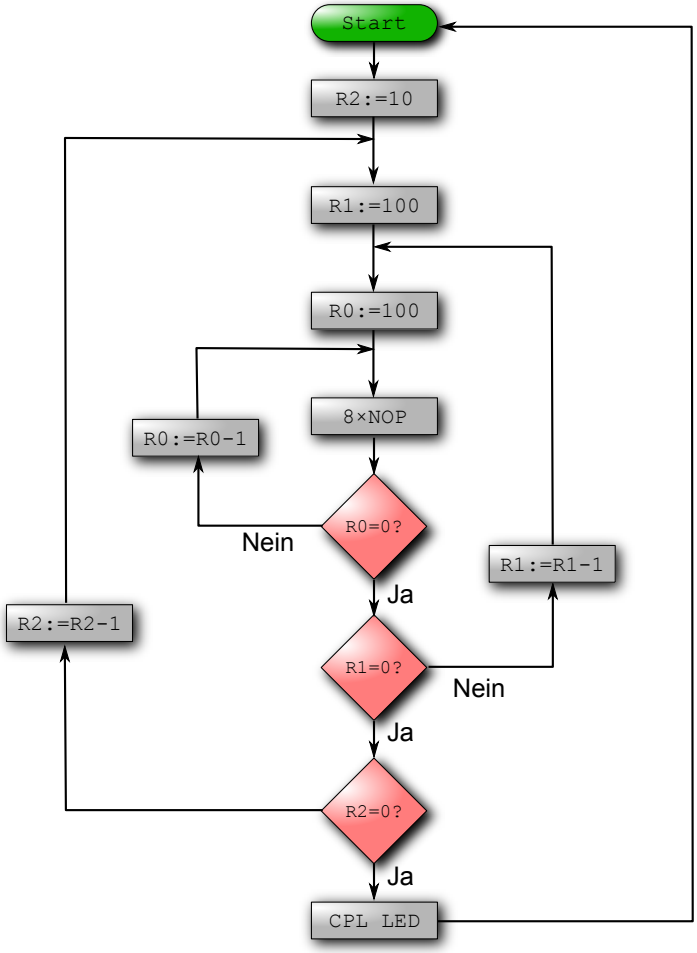
Versuchsziel

Durch die Programmierung von Verzögerungsschleifen, sollte eine LED im Sekundentakt zum Blinken gebracht werden. Die LED war an den Pin-Port P3.5 angeschlossen und konnte über diesen geregelt werden. Mit Hilfe der NOP-Befehle wird jeweils 1 μs verschwendet. Dadurch wurde eine Zeitbasis von 10 μs erzeugt, welche mit Hilfe von drei Schleifen 100.000 mal durchlaufen wurde, sodass genau eine Sekunde verging. Hierbei wurde berücksichtigt, dass für den DJNZ-Befehl 2 μs benötigt werden.

Quellcode

```
$nomod51
$include(reg515.inc)

        LED Bit P3.5
        jmp main
                        /*- - - - - - - - - - - -
                        ;lässt LED in 1s Intervall blinken
                        - - - - - - - - - - - - - */
        org 100h

main:                   ; Beginn des Hauptprogramms

                        ;- - - - - - - - - -Zeitschleife- - - - - - - - - -
        Mov R2,#10      ;Variablen
Loop2:  Mov R1,#100     ; Sprungmarke 3. Schleife wiederholt innere beiden 10 mal
Loop1:  Mov R0,#100     ; Sprungmarke 2. Schleife wiederholt innere 100 mal
Loop0:  Nop             ;Sprungmarke innerste Schleife läuft 10µs*100
        Nop             ; NOP für 1µs;
        Nop
        Nop
        Nop
        Nop
        Nop
        Nop
        DJNZ R0, Loop0  ;Ende der innersten Schleife
        DJNZ R1, Loop1  ;Ende der 2. Schleife
        DJNZ R2, Loop2  ;Ende der 3. Schleife
                        ;- - - - - - - - - - - - - - - - - - - - - - - - - - -

        CPL LED         ;Invertieren des Port-Bit's (Ein/Aus-Schalten)
        jmp main        ;Endlosschleife
End
```

Flussdiagramm



Versuch 2

Versuch 2.1

2.1.1 Versuchsziel

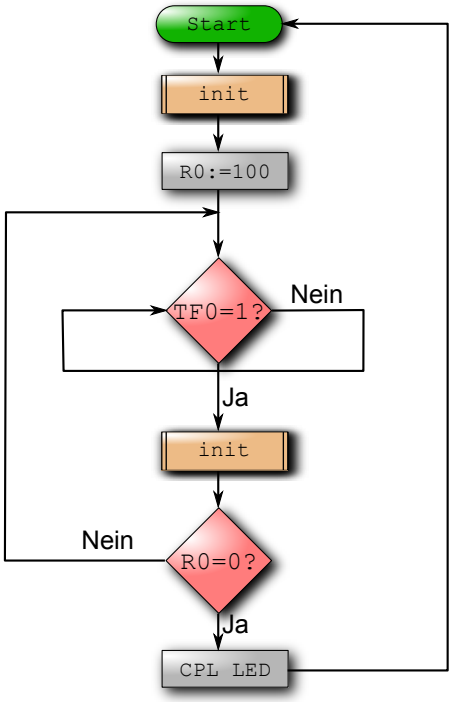
Ziel des Versuchs war es wieder die LED in einem gewünschten Takt leuchten zu lassen. Im Unterschied zu Versuch 1, sollte zum Programmieren der Zeitbasis nun die interne Zähler/Zeitgeber-Einheit verwendet werden. Im ersten Teil des Versuchs wurde der Timer im 16-Bit Modus verwendet, hierbei sollte eine Zeitbasis von 10 ms entstehen. Diese wurde in der Timerinitialisierung durch Setzen des Startwertes realisiert, der Startwert für 10 ms ergibt sich durch die Subtraktion vom Maximalwert unter Berücksichtigung der einzelnen Befehlsdauern, die hier mit 11 μs eingingen. Zum Erreichen einer vollen Sekunde wurde die Zeitbasis, mit Hilfe einer Schleife, 100 mal durchlaufen.
In der Initialisierung des Timers wurde der errechnete Startwert in das entsprechende high bzw. low Byte geschrieben, in unserem Fall war dies $TH0 = 0D8$ h und $TH0 = 0FA$ h. Außerdem musste in der Initialisierung das Überlaufflag TF0, mit Hilfe des clear-Befehls, zurückgesetzt werden. Über das TMOD Register wurde eingestellt, dass der Timer 0 im Modus 1 verwendet werden sollte und Timer 1 nicht benutzt wurde, dazu musste hier dass letzte bit gesetzt werden, wobei das TMOD Register nicht bitadressierbar ist und demnach das ganze Byte gesetzt wurde. Durch das Setzen des bits TR0 wurde zuletzt noch der Timer gestartet.

2.1.2 Quellcode

```
$nomod51
$include(reg515.inc)

        LED Bit P3.5
        jmp main

                        /*- - - - - - - - - - - -
                        ;lässt LED in 1s Intervall blinken mit Timer0 im 16-bit Modus
                        - - - - - - - - - - - - */
        org 100h

main:   Call init          ; Beginn des Hauptprogramms


                        ;- - - - - - - - - - -Zählschleife- - - - - - - - - - -
        Mov R0,#100        ; Schleife wiederholt Timerdurchlauf 100 mal
Repeat: JNB TF0,Repeat     ; 10ms bis zum Timerüberlauf
        Call init          ; Zurücksetzen und neustarten des Timers

        DJNZ R0, Repeat    ;Ende der Schleife
                        ;- - - - - - - - - - - - - - - - - - - - - - - - - - - -
        CPL LED            ;Invertieren des Port-Bit's (Ein/Aus-Schalten)
```

```
        jmp main            ;Endlosschleife

init:   mov TH0,#0D8h       ;max-Wert -(10000-11) (11µs: Befehlsdauer)
        mov TL0,#0FAh       ;high-byte/low-byte
        Clr TF0             ;Überlauf-Bit leeren
        Mov TMOD,#00000001b ;Timer1: nicht benutzt, Timer0: Modus1
        Setb TR0            ;Timer0 Start!
        RET
End
```

2.1.3 Flussdiagramm



Versuch 2.2

2.2.1 Versuchsziel

Im zweiten Teil des Versuchs sollte der Timer nun im 8-Bit Autoreload Modus betrieben werden. Hierbei wird das niederwerige Byte TL0 beim Auslösen des Interrupts durch Überlauf mit dem höherwertigen Byte TH0 überschrieben, dieses muss demnach den benötigten Reload-Wert enthalten. Im Gegensatz zum vorigen Versuchsteil sollte die Zeitbasis 250 μs betragen. Der Reload-Wert ergibt sich durch Abzug der gewünschten Zeit von 256 zuzüglich der Dauer für die Ausführung enthaltener Befehle, die diesmal mit 2 μs berücksichtigt wurden, d.h. sowohl in TH0 als auch in TL0 sollte der Wert

$$256 - 250 + 2 = 8 \quad \rightarrow \quad 8 \text{ h}$$

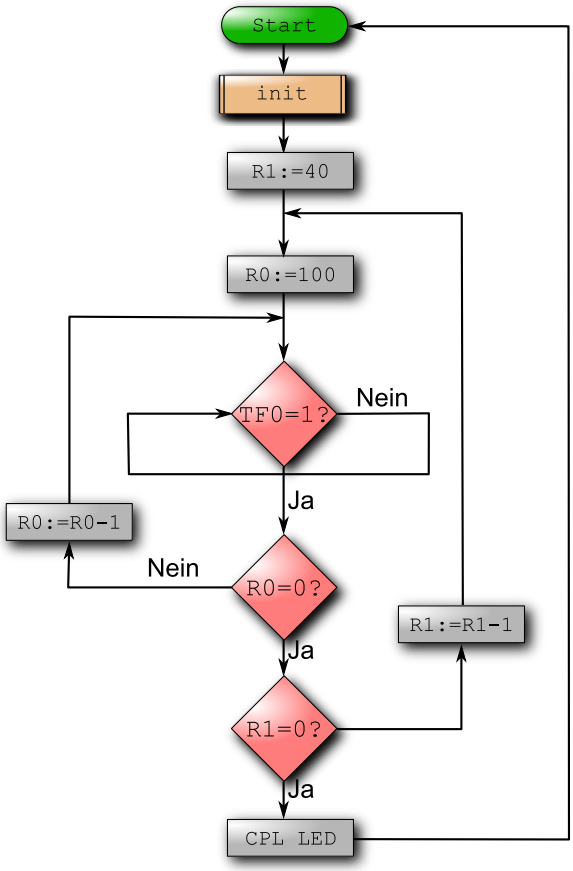
eingetragen werden. Dies geschah wieder in der Initialisierung, in der ausserdem wie oben das Überlaufflag gelöscht und das TMOD Register gesetzt wurde. Um den Timer 0 nun im Modus 2 zu verwenden wurde im TMOD Register das vorletzte bit gesetzt. Über das Setzen des TR0 bits wurde wieder der Timer gestartet.

2.2.2 Quellcode

```
$nomod51
$include(reg515.inc)

        LED Bit P3.5
        jmp main
                        /*- - - - - - - - - - - -
                        ;lässt LED in 1s Intervall blinken mit Timer0 im 8-bit Autoreload-Modus
                        - - - - - - - - - - - - */
        org 100h

main:   Call init          ; Beginn des Hauptprogramms, Aufruf des Unterprogramms zur Initialisierung


                           ;- - - - - - - - - - -Zählschleife- - - - - - - - - - -
        Mov R1,#40         ; Schleife wiederholt Timerdurchlauf 40 mal
Loop1:  Mov R0,#100        ; Schleife wiederholt Timerdurchlauf 100 mal
Repeat: JNB TF0,Repeat     ; 0.25ms bis zum Timerüberlauf
        Clr TF0
        DJNZ R0, Repeat    ;Ende der Schleife
        DJNZ R1, Loop1     ;Ende der Äußerenschleife
                           ;- - - - - - - - - - - - - - - - - - - - - - - - - - - -

        CPL LED            ;Invertieren des Port-Bit's (Ein/Aus-Schalten)

        jmp main           ;Endlosschleife

init:   mov TH0,#08h       ;max-Wert -(250-3) (3µs: Befehlsdauer)
        mov TL0,#08h       ;high-byte/low-byte
        Clr TF0            ;Überlauf-Bit leeren
```

```
        Mov TMOD,#00000010b ;Timer1: nicht benutzt, Timer0: Modus2
        Setb TR0            ;Timer0 Start!
        RET
End
```

2.2.3 Flussdiagramm



Versuch 3

Versuch 3.1

3.1.1 Versuchsziel

In diesem Versuch sollte das Port der seriellen Schnittstelle auf dem μ-Controller programmiert werden. Dazu sollte der interne Baudratengenerator im ersten Versuchsteil mit einer Oszillatorfrequenz von 12 MHz eine Baudrate von 9600 Baud erzeugen. Zur Kommunikation mit der seriellen Schnittstelle wurden die Unterprogramme "Rec" und "Trans" geschrieben, welche dem Empfang und dem Zurücksenden von Zeichen dienen. Außerdem wurde ein Unterprogramm "Carret" geschrieben, welches den Carriage-Return Befehl erkennt und durch ein CR-Zeichen und ein anschließendes Line-Feed Zeichen ersetzt. Zusätzlich wurde ein Unterprogramm "Buchstaben" zum Umwandeln von Klein- in Großbuchstaben erarbeitet.

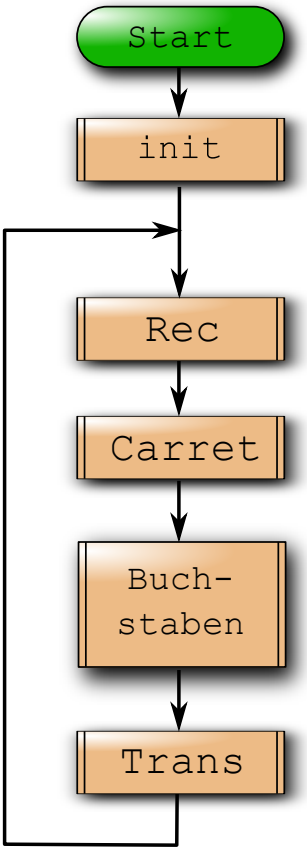
In der Initialisierung des Programms wurden zunächst folgende bits gesetzt SM0 = 0, SM1 = 1 und REN = 1, die ersten beiden dienen der Einstellung der seriellen Schnittstelle, welche somit im Modus 1 (8-bit-UART mit einstellbarer Baudrate) betrieben werden kann. Über das bit REN ist zunächst der serielle Empfang durch die Software im Allgemeinen freigegeben. Durch Setzen des bits BD wird der interne Baudratengenerator aktiviert. Weiterhin wurde noch das SMOD bit gesetzt, welches die Baudrate in dem verwendeten Modus verdoppelt. Dieses bit ist dabei nicht direkt bitaddressierbar und wurde deswegen über den Akku, mit Hilfe des gesamten Registers (PCON), gesetzt.

3.1.2 Quellcode

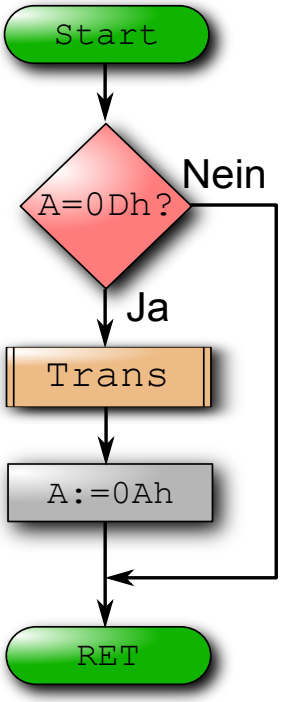
```
$nomod51
$include(reg515.inc)

            jmp main
            org 100h

main:       Call init           ; Initialisierung Baudratengenerator/Ser.Schnittstelle
Loop:       Call Rec            ; Warten auf empfang eines Zeichens
            Call Carret         ; Ersetzen von CR durch CR,LF
            Call Buchstaben     ; Ersetzen von Kleinbuchstaben durch Großbuchstaben
            Call Trans          ; Zurücksenden des Accumulators
            jmp Loop            ; Endlosschleife

Buchstaben: cjne A,#61h,buch1   ;Falls Accu < 61h setze CarryFlag, springe immer in nächste Zeile
buch1:      jc buchret          ;Falls CF gesetzt ist -> RET
            cjne A,#7Bh,buch2   ;Falls 60h < Accu < 7Bh setze CF, spinge immer in nächste Zeile
buch2:      jnc buchret         ;Falls CF nicht gesetzt -> RET
            subb A,#1Fh         ;Mache Großbuchstaben aus Kleinbuchstaben
buchret:    RET

Carret:     cjne A,#0Dh,Ende    ;Überprüfe ob CR empfangen wurde (wenn nicht -> Ende)
            Call Trans          ;Sende CR zurück
```

```
            Mov A,#0Ah          ;Schreibe LF in den Accu
Ende:       RET

Rec:        JNB RI,Rec          ;Abfragen ob Zeichen empfangen wurde
            mov A,SBUF          ;Empfangenes Zeichen in Accu laden
            clr RI
            RET

Trans:      mov SBUF,A          ;Senden des Zeichens im Accu
loop2:      JNB TI,loop2        ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET

init:       clr SM0             ;Seriellеschnittstelle in Modus 1
            setb SM1
            setb REN            ;Seriellen Empfang starten
            setb BD             ;internen Bautratengenerator aktivieren
            Mov A,PCON          ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A

            RET

End
```

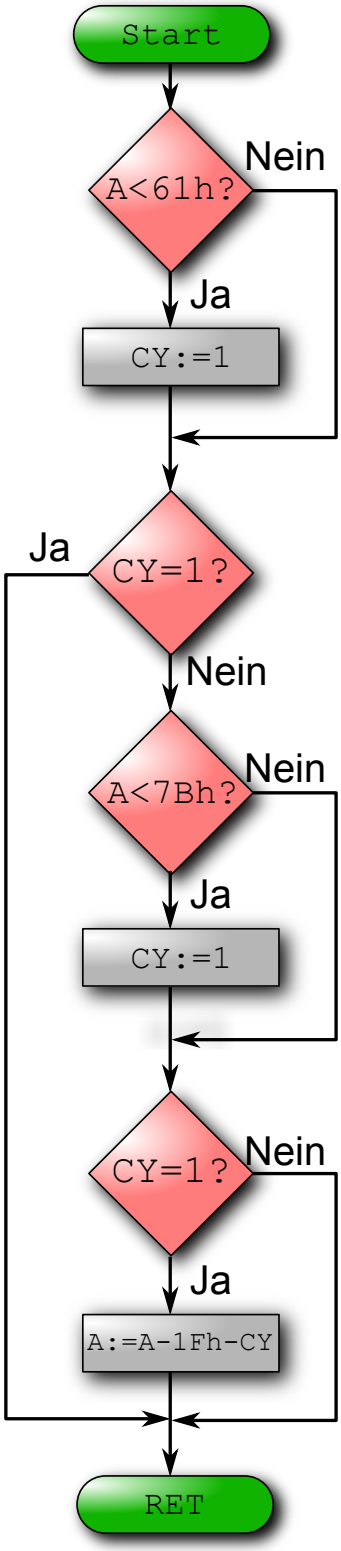
(a) Hauptprogramm



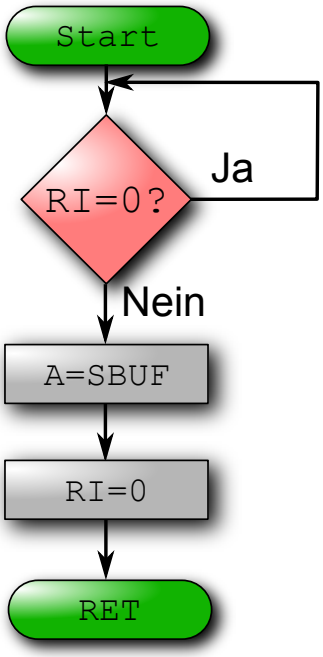
(b) Carret



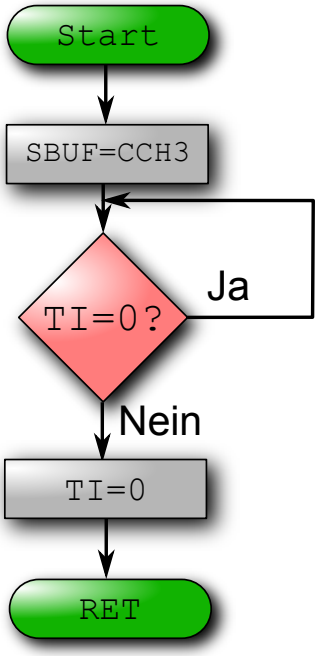
(b) Buchstaben



(c) rec



(d) trans



Versuch 3.2

3.1.1 Versuchsziel

Anders als in Teil 1 sollte die Baudrate mit Hilfe des Timers erzeugt werden. Hierbei wurden verschiedene Baudraten ausprobiert, die Baudraten mit den zugehörigen Werten des Nachladeregisters können dem Quellcode entnommen werden. Bereits bei einer Baudrate von 9600 Baud wurde die Übertragung fehlerhaft, bei einer Baudrate von 19200 Baud fand gar keine korrekte Übertragung mehr statt.
Die Initialisierung ist ähnlich zu der in Teil 1, nur das hier anstelle der Aktivierung des internen Baudratengenerators eine zusätzliche Timerinitialisierung vorgenommen wurde. Die Timerinitialisierung ist analog zu der in Versuch 2.2, mit dem Unterschied, dass diesmal Timer 1 verwendet wurde und je nach gewünschter Baudrate der Reload-Wert angepasst wurde. Der Wert für TH1 bzw. TH0 ergibt sich aus der in der Anleitung angegebenen Formel für die Baudrate:

$$\text{Baudrate} = \frac{2^{\text{SMOD}}}{32} \cdot \frac{\text{Oszillatorfrequenz}}{12 \cdot (256 - \text{TH1})}.$$

3.1.2 Quellcode

```
$nomod51
$include(reg515.inc)

            jmp main
            org 100h
                              /*- - - - - - - - - - - -
                              ;Serielle Schnittstelle mit Baudraten aus Timer1 im 8-bit Autoreload-Modus
                              - - - - - - - - - - - - */

main:       Call init         ; Initialisierung Baudratengenerator/Ser.Schnittstelle
Loop:       Call Rec          ; Warten auf empfang eines Zeichens
            Call Carret       ; Ersetzen von CR durch CR,LF
            Call Buchstaben   ; Ersetzen von Kleinbuchstaben durch Großbuchstaben
            Call Trans        ; Zurücksenden des Accumulators
            jmp Loop          ; Endlosschleife

Buchstaben: cjne A,#61h,buch1 ;Falls Accu < 61h setze CarryFlag, springe immer in nächste Zeile
buch1:      jc buchret        ;Falls CF gesetzt ist -> RET
            cjne A,#7Bh,buch2 ;Falls 60h < Accu < 7Bh setze CF, spinge immer in nächste Zeile
buch2:      jnc buchret       ;Falls CF nicht gesetzt -> RET
            subb A,#1Fh       ;Mache Großbuchstaben aus Kleinbuchstaben
buchret:    RET

Carret:     cjne A,#0Dh,Ende  ;Überprüfe ob CR empfangen wurde (wenn nicht -> Ende)
            Call Trans        ;Sende CR zurück
            Mov A,#0Ah        ;Schreibe LF in den Accu
Ende:       RET

Rec:        JNB RI,Rec        ;Abfragen ob Zeichen empfangen wurde
```

```
            mov A,SBUF          ;Empfangenes Zeichen in Accu laden
            clr RI
            RET

Trans:      mov SBUF,A          ;Senden des Zeichens im Accu
loop2:      JNB TI,loop2        ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET

init:       clr SM0             ;Seriellеschnittstelle in Modus 1
            setb SM1
            setb REN            ;Seriellen Empfang starten

                                ;- - - - - - - - - - - Timerinit- - - - - - - - - - - - - - - -
                                ;mov TH1,#230       ;baudrate 2400 => T1=230 (funktioniert)
                                ;mov TL1,#230       ;
                                ;mov TH1,#243       ;baudrate 4800 => T1=243 (funktioniert)
                                ;mov TL1,#243       ;
                                ;mov TH1,#249       ;baudrate 9600 => T1=249,49 (fehlerhafte Übertragung)
                                ;mov TL1,#249       ;
            mov TH1,#253        ;baudrate 19200 => T1=252,74 (Fehlerquote ~ 100%)
            mov TL1,#253        ;


            Clr TF1             ;Überlauf-Bit leeren
            Mov TMOD,#00100000b ;Timer1: Modus2, Timer0: nicht benutzt
            Setb TR1
                                ;- - - - - - - - - - - Timerinit- - - - - - - - - - - - - - - -

            Mov A,PCON          ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A

            RET
End
```

Versuch 4

Versuchsziel

In diesem Versuch sollte die Kommunikation über die serielle Schnitstelle und der Timer auf Interrupt-Betrieb umgestellt werden, sodass Versuch 2 und 3 parallel ablaufen konnten. Die Zeitbasis des Timers zum Blinken der LED sollte 100 μs betragen, weswegen sich der Reload-Wert des Timers zu dem in Versuch 2 unterscheidet. Aus diesem Grund musste die Zeitbasis nun 10000 mal durchlaufen werden. Dies wurde mit Hilfe zweier Schleifen mit jeweils 100 Durchläufen realisiert. Die Unterprogramme sind mit denen aus Versuch 3 identisch und deswegen nicht nochmal aufgeführt. Im Hauptprogramm wird neben der Initialisierung eine Endlosschleife ausgeführt, die nur durch den entsprechenden Interrupt unterbrochen wird. Dabei ist in den verschiedenen Interrupt-Einsprungadressen, der Verweis auf die zugehörige Serviceroutine vermerkt, wodurch diese im Falle eines Interrupts ausgeführt wird.

Die Initialisierung besteht zum einen aus der Timerinitialisierung, die bis auf den erwähnten Reload-Wert mit der in Versuch 2.2 übereinstimmt, und zum Anderen aus der Initialisierung der seriellen Schnittstelle, die der in Versuch 3.1 entspricht. Weiterhin werden hier die Register für die Schleifen gesetzt (R0 = R1 = 100) und die benötigten Interrupt Einstellungen vorgenommen, wofür die drei bits ETO, ES und EAL gesetzt werden. Diese bits entsprechen der Aktivierung des Timer Interrupts, der Aktivierung des Interrupts der seriellen Schnittstelle und der allgemeinen Aktivierung von Interrupts.

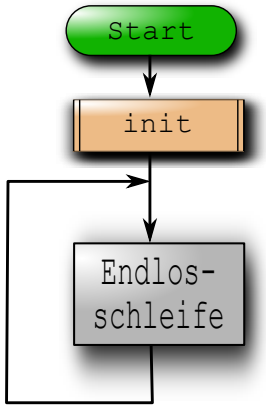
Quellcode

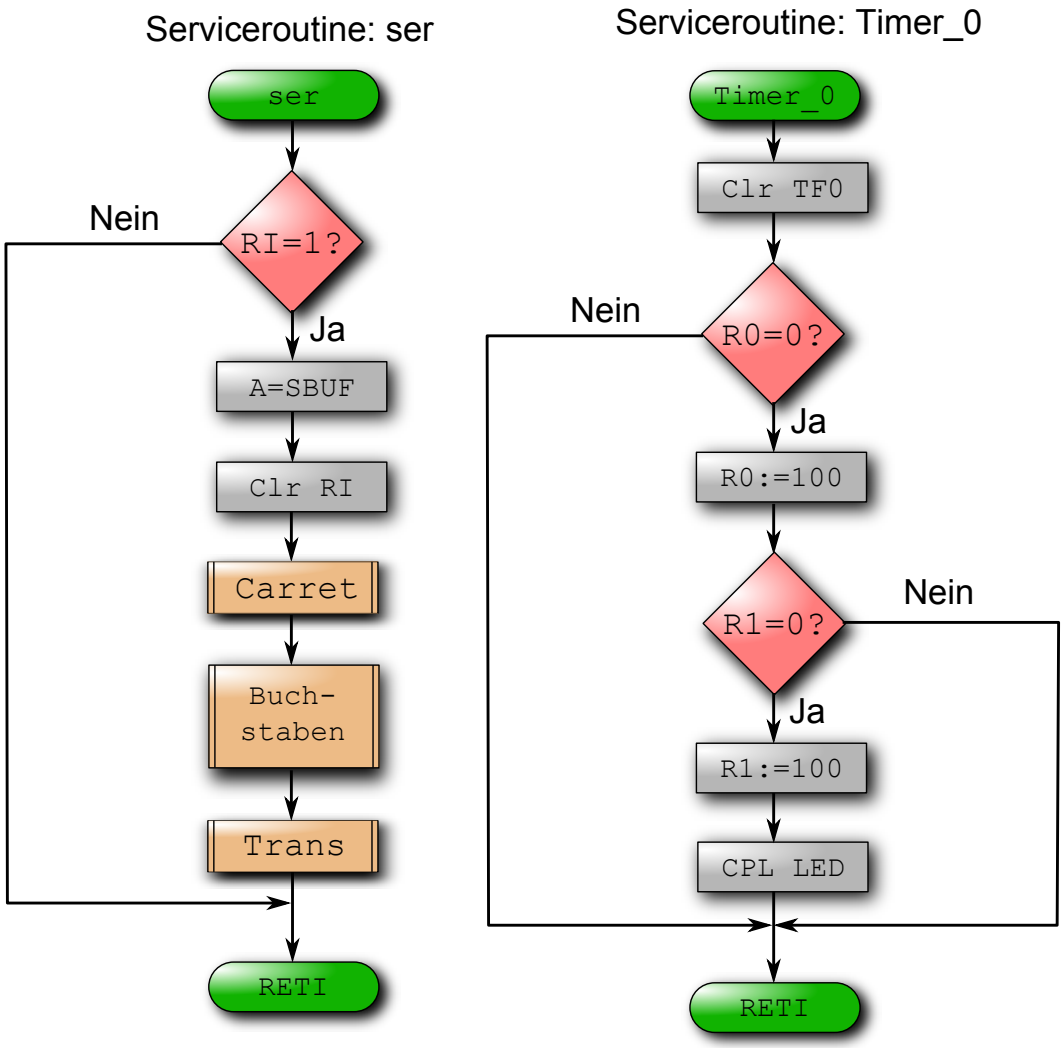
```
$nomod51
$include(reg515.inc)

            LED Bit P3.5
            jmp main
                                /*- - - - - - - - - - - -
                                ;lässt LED in 1s Intervall blinken mit Timer0 im 8-bit Autoreload-Modus
                                ;und Interupt
                                - - - - - - - - - - - - */
            org 0Bh
            jmp Timer_0
            org 23h
            jmp ser

            org 100h

main:       Call init           ;Beginn des Hauptprogramms, Aufruf des Unterprogramms zur Initialisierung

jump:       jmp jump            ;Endlosschleife

init:                           ;- - - - - - - - - - - Timer - - - - - - - - - - - - - - -
            mov TH0,#9Bh        ;max-Wert-(100)
```

```
            mov TL0,#9Bh            ;high-byte/low-byte
            Clr TF0                 ;Überlauf-Bit leeren
            Mov TMOD,#00000010b     ;Timer1: nicht benutzt, Timer0: Modus2
            Setb TR0                ;Timer0 Start!

            Mov R0,#100             ;Initialisieren der Zählschleifen
            Mov R1,#100
                                    ;- - - - - - - - - - - -  Serielle Schnitstelle - - - - - - - -
            clr SM0                 ;Serielleschnittstelle in Modus 1
            setb SM1
            setb REN                ;Seriellen Empfang starten
            setb BD                 ;internen Bautratengenerator aktivieren
            Mov A,PCON              ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A
                                    ;- - - - - - - - - - - -  Interrupts - - - - - - - - - - - - - -
            SETB ET0                ;Interrupt des Timer_0
            SETB ES                 ;Interrupt der Ser.Schnittstelle
            SETB EAL                ;Alle Interrupts zulassen

            RET

Trans:      mov SBUF,A              ;Senden des Zeichens im Accu
loop2:      JNB TI,loop2            ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET

Carret:     cjne A,#0Dh,Ende_Car    ;Überprüfe ob CR empfangen wurde (wenn nicht -> Ende)
            Call Trans              ;Sende CR zurück
            Mov A,#0Ah              ;Schreibe LF in den Accu
Ende_Car:   RET

Buchstaben: cjne A,#61h,buch1       ;Falls Accu < 61h setze CarryFlag, springe immer in nächste Zeile
buch1:      jc buchret              ;Falls CF gesetzt ist -> RET
            cjne A,#7Bh,buch2       ;Falls 60h < Accu < 7Bh setze CF, spinge immer in nächste Zeile
buch2:      jnc buchret             ;Falls CF nicht gesetzt -> RET
            subb A,#1Fh             ;Mache Großbuchstaben aus Kleinbuchstaben
buchret:    RET

Timer_0:    Clr TF0                 ;Serviceroutine Timer0 (alle 0.25ms)

            DJNZ R0,return          ;Zählt 100 mal bis 100 -> 1s
            MOV R0,#100

            DJNZ R1,return
            MOV R1,#100

            CPL LED                 ;Schalten der LED
return:     RET

ser:        JNB RI,Ende_ser         ;Abfragen ob Zeichen empfangen wurde
            mov A,SBUF              ;Empfangenes Zeichen in Accu laden
            clr RI
            Call Carret             ;Ersetzen des Returns
            Call Buchstaben         ;Ersetzen von Kleinbuchstaben
            Call Trans              ;Zurücksenden des Zeichens
```

```
Ende_ser:   RETI

End
```

Flussdiagramm





Versuch 5

Versuchsziel

Mit Hilfe eines pulsweitenmodulierten Signals soll die Position eines Stellhebels an einem Servomotor entsprechend der Benutzereingaben am Terminal verändert werden. Der Mikrocontroller muss hier ein Signal mit einer Frequenz von ca. 60Hz (Periodendauer = 16,3 ms) und einer Pulszeit zwischen 0,6 und 2,1 ms erzeugen. Die Elektronik des Servomotors übersetzt dann das Verhältnis aus Pulszeit und Pausenzeit in einen festen Stellwinkel.
Für die Pulsweitenmodulation wird Timer2 des Mikrocontrollers verwendet. Dieser verfügt über 4 16-Bit-Compare-Register die jeweils mit einem Ausgangspin verbunden sind. Eines der Register (CRCH/CRCL) wird für den Reload des Timers verwendet. Der Reloadwert C000h entspricht einer Frequenz von 61,03 Hz. Wärend der Timer von C000h nach oben zählt wird in jedem Schritt mit den drei restlichen Compare-Registern verglichen. Stimmt der Timerwert mit dem Wert eines der Register überein wird der entsprechende Ausgangspin auf 1 gesetzt. Bei Timerüberlauf werden wieder alle Pins auf 0 gesetzt. In diesem Versuch ist der Servomotor am Ausgang P1.3 angeschlossen. Entsprechend bestimmt der Wert des dritten Registers (CCL3/CCH3) die Pulsweite und somit die Position des Stellhebels.
Nach Reset des 80C535 wird der Stellhebel zunächst auf eine mittige Position gefahren. Der hierfür nötige Compare-Wert wurde durch Ausprobieren bestimmt.. Die Kommunikation mit dem Terminal verläuft analog zu Versuch 3 über die serielle Schnittstelle. Die Zeichen 1,2 und 3 werden als Steuerzeichen für den Servomotor verwendet. Gibt der Benutzer eine 1 oder 3 ein wird 0020h zum Wert des dritten Compare-Registers addiert (oder subtrahiert), entsprechend ändert sich die Position des Hebels um ein kleines Stück. Der Comparewert muss hierbei aber in einem bestimmten Intervall bleiben um sicherzustellen dass der Servomotor beim Erreichen des Anschlags nicht beschädigt wird. Das Programm muss also zunächst den Comparewert mit einem Maximal/Minimal Wert vergleichen.
Bei Eingabe des Zeichen 2 wird der Hebel wieder auf seine Ausgangsposition gefahren, d.h. der Initialisierungswert wird ins Compareregister geladen.

Quellcode

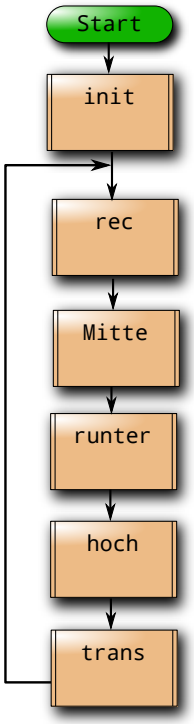
```
X$nomod51
$include(reg515.inc)

            jmp main
            org 100h

;-------------------------------------main-----------------------------------------
main:       Call init             ;Initialisierung Baudratengen./Ser.Schnittstelle
Loop:       Call Rec              ;Warten auf empfang eines Zeichens
            Call Mitte            ;falls Zeichen "2" empfangen
                                  ;-> Hebel auf ausgangsposition fahren
            Call runter           ;falls Zeichen "3" empfangen
                                  ;-> Hebel einen Schritt nach unten (bis min.)
            Call hoch             ;falls Zeichen "1" empfangen
                                  ;-> Hebel einen Schritt nach oben (bis max)
            Call Trans            ;Rückgabe aktuellen Position (Wert von CRCH)
            jmp Loop              ;Endlosschleife

;-------------------------------------Mitte----------------------------------------
Mitte:      cjne A,#32h,mittEnde ;Prüfe ob 2 empfangen, wenn nicht -> RET
            mov CCH3,#0FAh       ;wenn ja -> Fahre zur Mitte
            mov CCL3,#00h
mittEnde:   RET

;-------------------------------------runter---------------------------------------
runter:     cjne A,#31h,runEnde  ;Prüfe ob 3 empfangen, wenn nicht -> RET
            mov R1,A             ;wenn ja -> Sicher Akku in Zwischenvariable R1

            mov A,CCH3           ;Überprüfe ob Highbyte des Compareregisters
                                 ;Minimal ist (CCH3 = F6h).
                                 ;F6E0h entspricht Hebelposition kurz vor Anschlag
                                 ;da der Comparewert nur in 20h schritten verändert
                                 ;wird ist sichergestellt dass das Minimum nicht
                                 ;"übergangen" wird.

            cjne A,#0F6h,runweit ;Wenn A=F6h -> lädt R1 in Akku (empfangenes
                                 ;-Zeichen) und springe zum Ende des Unterprogramms.
            jmp runres           ;Wenn A != F6h -> CCH3 ist größer als minimum ->
                                 ;es können 20h abgezogen werden.
runweit:
            CLR CY               ;Carry Flag löschen
            mov A,CCL3
            SUBB A,#20h          ;ziehe 20h vom Lowbyte ab.
                                 ;Carry-Flag wird mitabgezogen (hier 0)
                                 ;und bei überlauf neu gesetzt
            mov CCL3,A
            mov A,CCH3
            SUBB A,#0h           ;ziehe 0h und Carry-Flag vom highbyte ab.
            mov CCH3,A
runres:     mov A,R1             ;Wiederherstellen des empfangengen Zeichen
                                 ;aus Register R1

runEnde:    RET
```

```
;-----------------------------------hoch-----------------------------------------
hoch:       cjne A,#33h,hocEnde  ;Prüfe ob 1 empfangen, wenn nicht -> RET
            mov R1,A             ;wenn ja -> Sicher Akku in Zwischenvariable R1

            mov A,CCH3
            cjne A,#0FDh,hocweit ;FDh entspricht Hebelposition kurz vor Anschlag
            jmp hocres

hocweit:    CLR CY               ;lösche Carry
            mov A,CCL3
            ADDC A,#20h          ;Addiere 20h zum lowbyte (Carry Flag = 0)
            mov CCL3,A

            mov A,CCH3
            ADDC A,#0h           ;Addiere hier 0h und CF (übertrag vom lowbyte)
            mov CCH3,A
hocres:     mov A,R1             ;Wiederherstellen des empfangengen Zeichen
                                 ;aus Register R1
hocEnde:    RET

;-----------------------------------Rec------------------------------------------
Rec:        JNB RI,Rec           ;Abfragen ob Zeichen empfangen wurde
            mov A,SBUF           ;Empfangenes Zeichen in Akku laden
            clr RI
            RET

;----------------------------------Trans-----------------------------------------
Trans:      mov SBUF,CCH3        ;zu Testzwecken:
                                 ;Sende Highbytes des Compareregisters
loop2:      JNB TI,loop2         ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET

;-----------------------------------init-----------------------------------------
init:       clr SM0              ;Serielleschnittstelle in Modus 1
            setb SM1
            setb REN             ;Seriellen Empfang starten
            setb BD              ;internen Bautratengenerator aktivieren
            Mov A,PCON           ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A

            mov CCEN,#10000000b  ;Compare-Modus für Register CC3
            clr T2I1             ;Funktion als Timer: oszillator
            setb T2I0
            clr T2PS             ;Compare Modus 0
            setb T2R1            ;Reload Modus 0
            clr T2R0

            clr T2CM             ;Compare-Modus 0

            mov CRCL,#00h        ;Reloadwert für Frequenz: ~61hz
            mov CRCH,#0C0h
            mov CCL3,#00h        ;Comparewert für Mittige Ausgangsposition (FA00h)
            mov CCH3,#0FAh
```

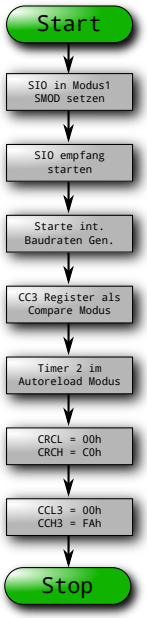
```
                RET
End
```

Flussdiagramm

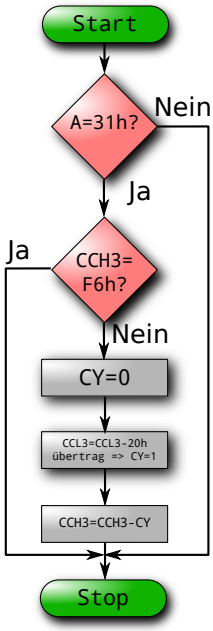
(a) main



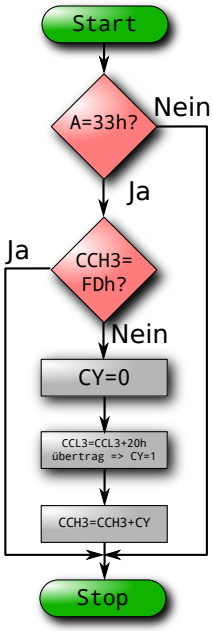
(b) init



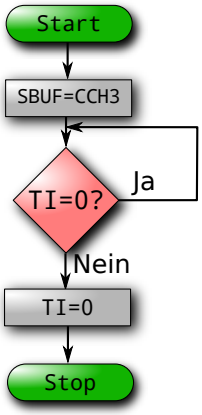
(c) runter



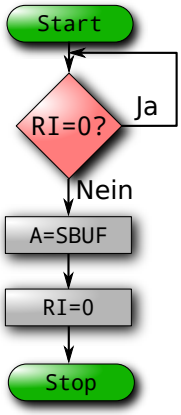
(d) hoch



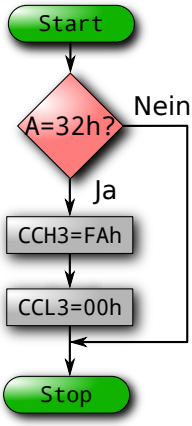
(e) trans



(f) rec



(g) Mitte



Versuch 6

Versuchsziel

In diesem Versuch soll ein LC-Display mit Hilfe des Mikrocontrollers angesteuert werden. Die Kommunikation mit dem Display läuft über 8-Datenleitungen (Port 5.0 - 5.7) und über drei Steuerleitungen RS, RW und E (Port 4..0 - 4.2). Ist das Bit RS nicht gesetzt wird die Bitfolge an der Datenleitung als Befehl für den Display-Controller interpretiert. Bei $RS = 1$ dagegen als ASCII-Zeichen für die Darstellung. Das RW Bit gibt an, ob Daten vom Mikrocontroller geschrieben oder gelesen werden. Um Befehle oder Zeichen zu übertragen, muss $RW = 0$ sein. E (Enable) muss zur Übertragung zunächst gesetzt sein. Sind alle Daten und Steuerleitungen richtig gesetzt wird E gelöscht. Die fallende Flanke von E gibt den Zeitpunkt an, zu welchem der Dispaly-Controller die Daten liest.

Für die Initilaisierung des Displays müssen nach dem Einschalten eine Reihe von Befehlen in bestimmten zeitlichen Abständen gesendet werden. In dem Unterprogramm *init* werden unter anderem die für die Befehle entsprechenden Bitfolgen in den Akku geladen und das Unterprogramm $Befehl$ aufgerufen. Dieses setzt die Daten- und Steuerleitungen, erzeugt die fallende Flanke an Port E und ruft anschließend das Unterprogramm $delay5$ auf. Durch den Aufruf von $delay5$ (5 ms Pause) ist sichergestellt, dass mindestens die benötigten Pausenzeiten verstreichen die der Dispaly-Controller benötigt.

Neben dem LCD wird auch die serielle Schnittstelle (s. Versuch 3) für die Kommunikation mit dem Terminal initialisiert. Zeichen die auf dem Terminal eingegeben werden gibt der Mikrocontroller zur Darstellung an das LCD weiter. Dabei prüft das Unterprogramm $deletpruf$ ob eines der drei Steuerzeichen eingegeben wurde. Ist dies der Fall wird ein entsprechender Befehl an den LCD gesendet. Für Backspace (08h) wird der gesamte Display gelöscht und der Cursor auf die Ausgangsposition gesetzt. Wird Esc (1Bh) eingegeben springt der Cursor in die zweite Zeile. Bei Return (0Dh) springt das Programm in eine Endlosschleife. Dabei wird der Inhalt des Displays in Abständen von 250 ms nach links verschoben.

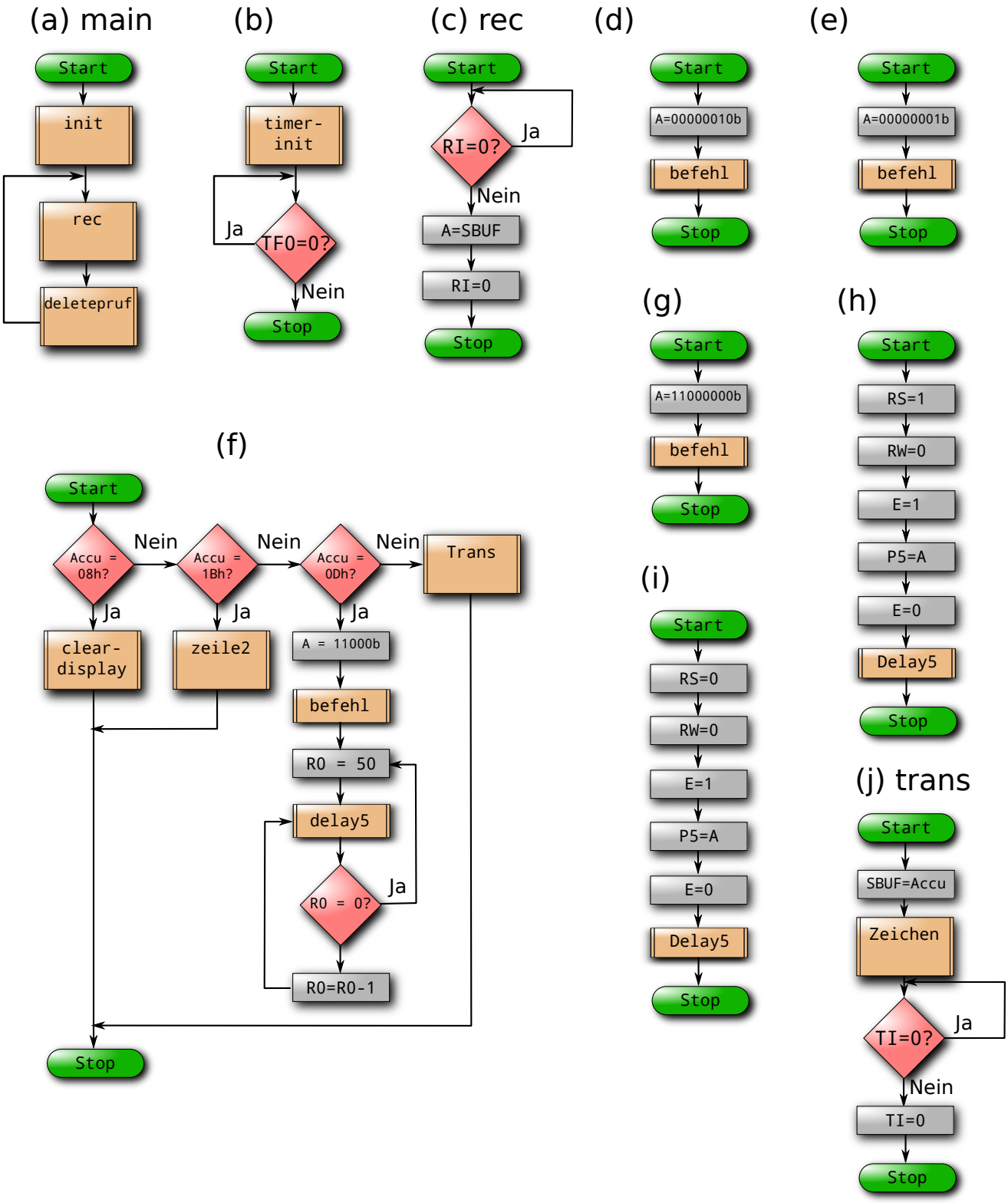
Quellcode

```
$nomod51
$include(reg515.inc)

RS    Bit P4.0
RW    Bit P4.1
E     Bit P4.2

            jmp main
            org 100h

;-------------------------------------main-----------------------------------------
main:       Call init            ;Initialisierung Baudratengen./Ser.Schnittstelle
Loop:       Call Rec             ;Warten auf empfang eines Zeichens
            Call deletepruf      ;Prüfe welches Zeichen empfangen wurde
            jmp Loop             ;Endlosschleife

;-------------------------------------Rec------------------------------------------
Rec:        JNB RI,Rec           ;Abfragen ob Zeichen empfangen wurde
            mov A,SBUF           ;Empfangenes Zeichen in Accu laden
            clr RI               ;
            RET

;-------------------------------------Trans----------------------------------------
Trans:      mov SBUF,A           ;Senden des Zeichens im Accu an LCD und Terminal
            Call zeichen
loop2:      JNB TI,loop2         ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET

;-------------------------------------deletpruf------------------------------------
;--Überprüft ob steuerungszeichen(Backspace,Return,Esc) empfangen wurde         --
deletepruf: cjne A,#08h,runesc   ;Backspace empfangen? wenn nicht -> Prüfe Esc
            Call cleardisplay    ;wenn ja -> display inhalt löschen
            jmp runEnde          ;RET

runesc:     cjne A,#1Bh,runret   ;Esc empfangen? wenn nicht -> Prüfe Return
            Call zeile2          ;wenn ja -> Cursor des LCD in Zeile zwei
            jmp runEnde          ;RET

runret:     cjne A,#0Dh,runtrans ;Return empfangen? wenn nicht -> Sende Zeichen
            jmp shift            ;wenn ja -> Endlosschleife für Lauftext starten

runtrans:   Call Trans           ;Zeichen Senden (LCD und Terminal)
runEnde: RET

;-------------------------------------befehl---------------------------------------
;--Teilt dispaly controler mit dass information an Portpins als Befehl          --
;--zu interpretieren ist und sendet inhalt des Akkus an den LCD                 --
befehl:     clr RS               ;
            clr RW               ;
            setb E               ;
            Mov P5,A             ;zu Schreibenden Befehl an Portpins anlegen
            clr E                ;erzeugt fallende Flanke
```

```
            Call delay5          ;Warte bis befehl gesendet wurde
            Ret

;-----------------------------------zeichen---------------------------------------
;--Teilt dispaly controler mit dass information an Portpins als Zeichen         --
;--zu interpretieren ist und sendet inhalt des Akkus an den LCD                  --
zeichen:    setb RS              ;
            clr RW               ;
            setb E
            Mov P5,A             ;zu Schreibendes Zeichen an Portpins anlegen
            clr E                ;erzeugt fallende Flanke
            Call delay5          ;Warte bis befehl gesendet wurde
            Ret


;-----------------------------------delay5----------------------------------------
delay5:     Call timerinit       ;Zurücksetzen und neustarten des Timers
Repeat:     JNB TF0,Repeat       ;5ms bis zum Timerüberlauf danach RET
            RET

;-----------------------------------shift----------------------------------------
;--Erzeugt Laufschrift durch ständiges verschieben des Displayinhalte mit kurzen--
;--pausen zwischen den befehlen                                                  --
shift:      MOV A,#00011000b     ;befehl: verschiebe Displayinhalt (nach links)
            Call befehl          ;befehl senden
            Mov R0,#50           ;warte 50*5ms = 250 ms
Repeat2:    Call delay5
            DJNZ R0, Repeat2
            jmp shift            ;endlos schleife

;-----------------------------------timerinit-------------------------------------
timerinit:  mov TH0,#0ECh        ;TH0 = 0ECh, TL0 = 082h entspricht
            mov TL0,#082h        ;max-Wert - 5000 (-> 5 ms bis überlauf)

            Clr TF0              ;Überlauf-Bit leeren
            Mov TMOD,#00000001b  ;Timer1: nicht benutzt, Timer0: Modus1
            Setb TR0             ;Timer0 Start!
            RET

;-----------------------------------cleardisplay----------------------------------
cleardisplay:
            Mov A,#00000001b     ;befehl zum löschen des displayinhalts
            Call befehl
            RET

;-----------------------------------Cursorhome------------------------------------
Cursorhome: Mov A,#00000010b     ;befehl für Cursor auf Homeposition
            Call befehl
            RET

;-----------------------------------zeile2----------------------------------------
zeile2:     Mov A,#11000000b     ;befehl für Sprung in 2. Zeile
            Call befehl
            RET

;-----------------------------------init------------------------------------------
```

```
init:       clr SM0              ;Serielleschnittstelle in Modus 1
            setb SM1
            setb REN ;Seriellen Empfang starten
            setb BD     ;internen Bautratengenerator aktivieren
            Mov A,PCON  ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A

            Call timerinit

            ;----------------LCD initialisierung---------------
            Mov A,#00110000b  ;Software-Reset: function set
            Call befehl
            Mov A,#00110000b
            Call befehl
            Mov A,#00110000b
            Call befehl
            Mov A,#00111100b  ;function set: 2 Lines, 5*10 dots
            Call befehl
            Mov A,#00001111b  ;Display on, cursor on, blink on
            Call befehl
            Mov A,#00000001b  ;display clear
            Call befehl
            Mov A,#00000110b  ;entry mode, inkrement, shift aus
            Call befehl
            ;-------------------------------------------------

            RET
End
```

Flussdiagramm



Versuch 7

Versuchsziel

Der Mikrocontroller soll in diesem Versuch als Analog/Digital-Wandler programmiert werden. Dabei sollen die gemessenen Spannungen über die serielle Schnittstelle auf dem Terminal ausgegeben werden.

Der 80535 verfügt über einen 8-Bit A/D-Wandler. Mit diesem können Spannungen zwischen $0V$ und $+5V$ gemessen werden.

Zusätzlich kann über das DAPR-Register der Messbereich variiert und damit die Auflösung vergrößert werden. Diese Möglichkeit soll dazu genutzt werden einen Softwareseitigen 10-Bit A/D-Wandler zu programmieren. Zunächst wird eine 8-Bit Messung durchgeführt und mit dem Ergebnis der Messbereich für die genauere Messung festgelegt. Aus der ersten Messung ergeben sich dann die Bits 8 und 9, aus der genaueren Messung die Bits 0 - 7 des 10-bit Ergebnis. 10-Bit entsprechen hier einer Auflösung von $4,88mV$ im Gegensatz zur 8-Bit Auflösung von $19,53mV$. Vorraussetzung ist allerdings, dass die Eingangsspannung zwischen den beiden Messungen konstant bleibt.
Nach Reset des Mikrocontrollers wartet dieser zunächst auf die Eingabe des Benutzers am Terminal. Sobald die Eingabe erfolgt, ruft das Unterprogramm *runreturn* das Unterprogramm *Messung* auf. Dieses startet die 8-Bit Messung in dem 0 in das DARP-Register geschrieben wird. Nach erfolgter Messung steht das Ergebnis im Register ADDAT und wird an den Terminal gesendet (*Trans*). Das Unterprogramm *calc-DARP* berechnet die neuen Grenzen und schreibt die entsprechenden Bitfolgen in DARP. Die Ergebnisse der beiden Messungen werden zu einem 10-Bit Ergebnis zusammengesetzt. Hierfür werden unter anderem die Befehle *RR*, *RL* und *ANL* benötigt. *RR* verschiebt die Bitfolge im Akku nach rechts (*RL* nach links). *ANL* verundet den Akku mit einer Bitfolge, wodurch es möglich ist einzelne Bits zu löschen, obwohl der Akku nicht direkt bitadressierbar ist. Das 10-Bit Ergebnis wird schließlich auf dem Terminal ausgegeben.

Quellcode

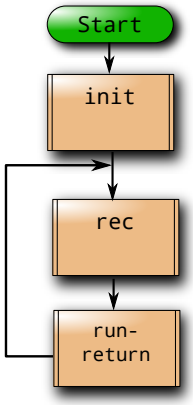
```
$nomod51
$include(reg515.inc)

RS    Bit P4.0
RW    Bit P4.1
E     Bit P4.2

            jmp main
            org 100h

;-------------------------------------main-------------------------------------------
main:       Call init           ;Initialisierung Baudratengen./Ser.Schnittstelle
Loop:       Call Rec            ;Warten auf empfang eines Zeichens
```

```
            Call runreturn      ;Starte Messung falls Return empfangen wurde
            jmp Loop            ;Endlosschleife

;-----------------------------------Rec-----------------------------------------
Rec:        JNB RI,Rec          ;Abfragen ob Zeichen empfangen wurde
            mov A,SBUF          ;Empfangenes Zeichen in Accu laden
            clr RI              ;
            RET

;-----------------------------------Trans---------------------------------------
Trans:      mov SBUF,A          ;Senden des Zeichens im Accu
loop2:      JNB TI,loop2        ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET

;-----------------------------------runreturn-----------------------------------
runreturn:  cjne A,#0Dh,runEnde ;Prüfe ob Return empfangen, wenn nicht -> RET
            Call Messung        ;Starten der 8-Bit Messung
            Call Trans          ;Sende das Ergebnis der 8-Bit Messung
            Call calcDAPR       ;Berechne neue Grenze, Starte '10 Bit' Messung
runEnde:    RET                 ;

;-----------------------------------calcDAPR------------------------------------
calcDAPR:                       ;Ergebnis der 8Bit Messung im Akku.
            SWAP A              ;Vertausche Low/High Nibble
            ANL A,#00001111b    ;Löschen des High Nibble
            clr C               ;
            SUBB A,#00000010b   ;Low-Nibble - 10b = Untergrenze

            JNB ACC.7,jmp2      ;Prüfe ob Untergrenze < 0, wenn nein -> jmp2
            MOV A,#00000000b    ;wenn ja: Untergrenze = 0, -> jmp4
            jmp jmp4

jmp2:       CJNE A,#00001100b,jmp3     ;CF = 1 wenn A < 00001100b, CF = 0 sonst
jmp3:       JC jmp4             ;Prüfe ob Untergrenze > 1100b, wenn nein -> jmp4
            MOV A,#00001100b    ;wenn ja -> untergrenze = 1100b

jmp4:       MOV R1,A            ;Speicher Untergrenze in R1
            SWAP A              ;Vertausche Low/High Nibble
            ADD A,#01000000b    ;Berechne Obergrenze aus Untergrenze (+0100b)
                                ;entspricht Messfenster von 1,25 V
            ADD A,R1            ;Ober und Untergrenze zusammenfügen
            mov DAPR,A          ;und in DAPR schreiben um 2. Messung zu Starten
LoopM2:     JB BSY,LoopM2       ;Warte auf Ende der Messung
            mov A,ADDAT         ;Ergebnis in R2 Speichern.
            Mov R2,A
            ANL A,#11000000b    ;Bit 7 und Bit 6 von Ergebnis in Akku Laden
            RL A                ;und auf Position 1 und 0 verschieben
            RL A
            ADD A,R1            ;A=A+R1 (untergrenze in Low-Nibble von R1)
            RR A                ;Ergebnis der Addition wird aufgeteilt:
                                ;Bit 0,1 werden Bit 7,6 des Ergebnis
            RR A                ;Bit 3,2 werden Bit 9,8 des Ergebnis
            Mov R3,A            ;Speichern A in R3,
                                ;Bit 9,8 des Ergebnis auf Position 0,1
            ANL A,#11000000b    ;Bit 7,6 des Ergebnis auf Pos. 7,6 sonst 0
```

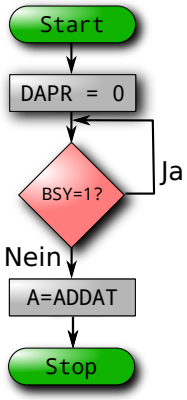
```
            MOV R4,A            ;Bit 7,6 in R4 zwischen Speichern

            MOV A,R2            ;
            ANL A,#00111111b    ;Bit 7,6 des Ergebnis der 2. Messung Löschen
            MOV R2,A            ;

            MOV A,R3            ;
            ANL A,#00000011b    ;Bit 7,6 in R3 Löschen,
            MOV R3,A            ; R3 soll nur bit 9,8 (auf Pos. 0,1) enthalten

            MOV A,R4            ;
            ADD A,R2            ;Neu berechnetes Bit 7,6 zum Ergebnis
            Mov R2,A            ;der 2. Messung hinzufügen

            Mov A,R3            ;Sende Bit 9,8 an Terminal
            Call Trans
            Mov A,R2            ;Sende Bit 7 bis 0 an Terminal
            Call Trans
            RET

;-----------------------------------Messung---------------------------------------
Messung:    mov DAPR,#0         ;messung starten (intevall 0..5V)
LoopM:      JB BSY,LoopM        ;Warte auf Ende der Messung
            mov A,ADDAT         ;Ergebnis der Messung in Accu
            RET

;-----------------------------------init------------------------------------------
init:       clr SM0             ;Serielleschnittstelle in Modus 1
            setb SM1
            setb REN            ;Seriellen Empfang starten
            setb BD             ;internen Bautratengenerator aktivieren
            Mov A,PCON          ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A

            ;Call timerinit

            clr ADM
            clr BSY

            RET
End
```

Flussdiagramm

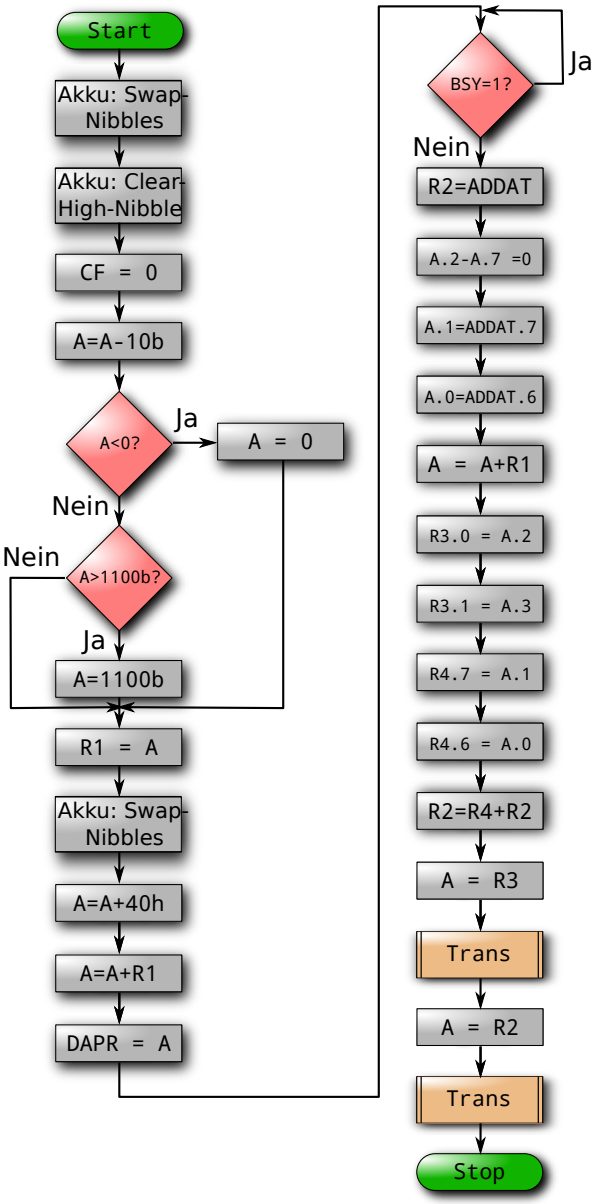
(a) main



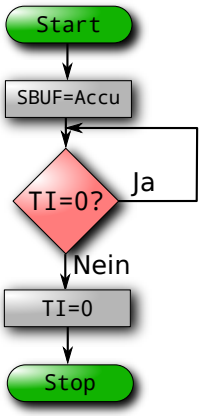
(b) Messung



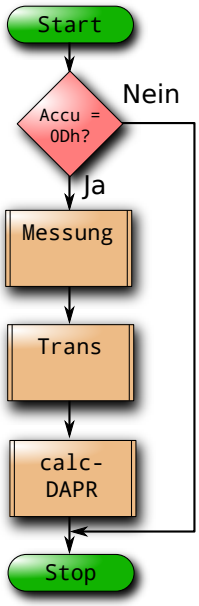
(c) clacDAPR



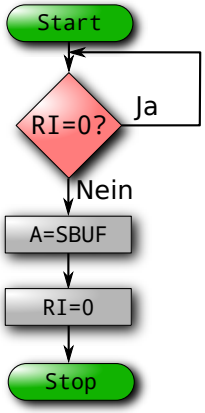
(d) trans



(e) runreturn



(f) rec



Versuch 8

Versuchsziel

Ziel dieses Versuches war es, das Zeitsignal einer DCF77-Funkuhr zu empfangen, zu decodieren und gut lesbar auf einem Display anzuzeigen

Hierbei wurde zusätzlich eine LED angeschlossen, welche bei Pulshöhen leuchtet. Jede Sekunde leuchtet diese 100 bzw. 200 ms auf. Lediglich in der letzten Sekunde jeder Minute erfolgt keine Pulshöhe als Indikator für das Ende der Minute.

Das Programm wurde dabei von uns in 4 größere Unterprogramme unterteilt. "*Init*" Initialisiert dabei den Timer und die LCD-Kommunikation. Der Timer0 wird hierbei im Modus 0 benutzt, um nach dem Senden der LCD–Befehle 5 ms zu warten. Auch wird damit in Impulstiefen ein Zähler im 10 ms Takt betrieben, um die Länge der Pause zu bestimmen. Das LCD–Dispay wird während der Initialisierung mehrfach über einen Software–Befehl zurück gesetzt, um dessen Speicher und Anzeige tatsächlich zu resetten. Während der Initialisierung wird zudem `init` auf dem Display angezeigt.

Danach betritt das Hauptprogramm eine Endlos–Schleife, in der es die Unterprogramme "*Puls*", "*Analyse*" und "*Decoder*" aufruft.

"*Puls*" schaltet zunächst über das Programm "*LED*" die LED bei Pulshöhen ein, bzw. bei Pulstiefen aus. Danach wird auf ein Pulstief gewartet, die LED ausgeschaltet und bestimmt, wie lange es zum nächsten Pulshoch dauert. Hierbei wird ein Zähler im 10 ms Takt erhöht. Der damit erreichte Wert entspricht ungefähr der durch 10 ms geteilten Zeitdauer zwischen zwei Impulsen. Dieser Wert sollte idealerweise 80 für Bitwert 1 und 90 für Bitwert 0 betragen. Um Ungenauigkeiten zu vermeiden, wird nun geprüft, ob es in den Intervallen (0,75] (Fehler), (75,85] (Bitwert 1), (85,95] (Bitwert 0), (95,105] (Fehler) oder (105,∞] (Minuten–Ende) liegt. Das Minuten–Ende sollte etwa 1800 ms-1900 ms betragen, den Bereich 950-1050 als Fehlerpuffer zu benutzen hat jedoch ausgereicht, daher vergleichen wir hier mit 1050 ms für das Minuten–Signal. Der Typ des aktuellen Signals wird nun in den ACC kodiert und gespeichert.

"*Analyse*" inkrementiert dabei einen Zeiger auf die Speicherstelle der aktuellen Sekunde. Der Speicherbereich beginnt dabei bei 30h. Danach zeigt es den Typ des aktuellen Bits in der Konsole an, wobei das Minuten–Signal `CR+LF` ausgibt. Außerdem wird der Typ nun an die Speicherstelle geschrieben, zu der der Sekundenzeiger zeigt. Wird ein Minuten–Signal empfangen, wird hier zudem noch R3 überprüft, welche die Ausgabe und Dekodierung abschalten kann. Dieser Wert wird bei Fehlern oder der Initialisierung auf 1 bzw. 2 gesetzt und lässt Dekodierung und Ausgabe nur zu, wenn die entsprechende Menge an

Minuten-Signalen vorher bereits gezählt wurde. R3=1 lässt das Programm also bis zur nächsten Minute warten, um 59 Fehlerfreie Sekunden–Bits zu erhalten.

"*Decoder*" überprüft auch zunächst R3 und stoppt bei $R3 \neq 0$. Zusammengefasst überprüft "*Decoder*" ob die aktuelle Sekunde einen Informationsblock (Stunde, Minute, Datum etc.) abschließt, überprüft das Prüfbit und gibt das entsprechende Zeichen auf dem LCD-Display aus.
Im Detail prüft es zunächst auf Minutenanfang, ob nun eine vollständige Minute vorliegt. Danach wird nun die 35. Sekunde als Stunden-Prüfbit mittels "*Pruf*" überprüft. Das Programm erwartet in R1 die Adresse des Prüfbits (34d+30h⇒54h) und in R5 die Anzahl der betroffenen Bits ({29-34}+{35}=7) und gibt bei Fehlern im ACC=1 aus. Tritt ein Fehler auf, zeigt "*Fehler*" `FEHLER` im LCD an, setzt R3 auf 1, um bis zum nächsten Minutensignal zu warten, und verlässt das Unterprogramm.
Hier sollte erwähnt werden, dass das Signal der 1. Sekunde bei 30h abgespeichert wird, die Adresse der 35. Sekunde also mit $34d + 30h = 54h$ berechnet wird.

Ohne Fehler wird nun die aktuelle Stunde ausgewertet. Dazu wird das Bit der 34. Sekunde ausgelesen (für 20 Stunden), um eine Stelle nach links verschoben und mit dem Bit der 33. Sekunde (für 10 Stunden) addiert. Um die Zahl als Text darzustellen wird nun 30h addiert (e.g. 4d+30h=34h= "4"). Man erhält die somit erste Stelle der Stunde. Dies wiederholt man mit dem 32-29. Sekundenbit (8, 4, 2, 1 Stunden) für die 2. Stelle der Anzeige. Nun wird ein ":" angezeigt und mit der Prüfung des Minuten–Prüfbits wie oben gezeigt, fortgefahren. Stimmt die Prüfung, werden hier die Sekunden–Bits 27-25 (40, 20, 10 Minuten) bzw. 24-21 (8, 4, 2, 1 Minuten) ausgewertet und angezeigt.

Wird das Programm bei Sekunde 43 ausgeführt, wird nun an den vorangegangen Teil mit dem gleichen Verfahren der Tag und "." angehangen. In der 50. Sekunde dann der Monat und in der 56. das Jahr. Da mit der 56. Sekunde auch das Prüfbit für das Datum verfügbar wird, wird es auch hier überprüft. Im Anschluss erfolgt die Ausgabe des Wochentags in der 2. LCD–Anzeigen–Zeile. Hierbei wird im Unterschied zu vorher jedoch nicht 30h addiert, sondern mit 0 bis 6 verglichen, wobei 0 Sonntag, 1 Montag,... und 6 Samstag bedeutet. Der Wochentag wird dabei abgekürzt in der 2. Zeile des LCD angezeigt angezeigt (e.g. `So` Für Sonntag).

Quellcode

```
$nomod51
$include(reg515.inc)
            RS Bit P4.0
            RW Bit P4.1
            E Bit P4.2

            jmp main
            org 100h

main:       Call init           ; Initialisierung Baudratengenerator/Ser.Schnittstelle/AD-Wandler
Loop:
            Call Puls           ; Leuchten/Blinken der LED, Umwandeln Impuls zu Bitwert (Akku)
            Call Analyse        ; Übersetzen Bits in Minutenspeicher und Bitweise Anzeige in Konsole
            Call Decoder        ; Dekodierung Minutenspeicher in Zeitformat und Anzeige in Display
            jmp Loop            ; Endlosschleife

                                ;- - - Unterprogramme- - -


                                ;- - - LED- - -
                                ;passt  LED (P3.5) an Zeitgeber (P1.1) an
                                ;- - - - - -
LED:
            jb P1.1, LED_AN
                                ;Ist    P1.1 gesetzt (Signal Zeitgeber), schalte LED (P3.5=0) an
            setb P3.5
                                ;Ansonsten  schalte LED (P3.5=1) aus
            RET
LED_AN:     clr P3.5
                                ;schalte    LED (P3.5=0) an
            RET
                                ;- - - Puls- - -
                                ;LED    leuchtet wenn Signalpuls anliegt, sonst 0.
                                ;Der    Puls jede Sekunde ist 100ms oder 200ms lang.
                                ;Jede   Minute fehlt ein Puls
                                ;Wandelt    außerdem Zeit des Pulses in Bit-Wert (Akku) um
                                ;- - - - - -
Puls:
Pulsanf:    Call LED            ;passt LED (P3.5) an Zeitgeber (P1.1) an
            jb P1.1,Pulsanf     ;Warte bis P1.1=0: Zeitpuls hört auf
            setb P3.5           ;schalte LED aus
            Mov A,#0            ;Akku als Zähler wie oft die Schleife durchlaufen wurde
pauseanf:
            Call delay5         ;5ms delay
            Call delay5         ;+5ms delay =10ms
            INC A               ;Schleifenzähler += 1
            jnb P1.1, pauseanf  ;Durchlaufe Schleife solange bis P1.1=1, also neuer Zeipuls anfängt
            clr C               ;Carry-Flag leeren, da im Folgenden benutzt
                                ;Jumps  werden wegen Carry-Flag(Konstante > Akku) benutzt
            CJNE A,#75, k750    ;Springt wenn A != 75 (75*10ms=750ms => Puls=1s-750ms=250ms),
                                ;wichtiger: Carry Flag; wenn A<75: Puls < 250ms. Sonst: Fehler!
o750:       CJNE A,#85, k850    ;85*10ms=850ms => Puls 150ms. Carry Flag hier: 250ms > Pulsdauer >150ms => 1
o850:       CJNE A,#95, k950    ;95*10ms=950ms => Puls 50ms. Carry Flag hier: 150ms > Pulsdauer >50ms => 0
```

```
o950:       CJNE A,#105, k1050  ;105*10ms=1050ms => Puls -50ms. Carry Flag hier: 50ms > Pulsdauer >-50ms => 2:
                                ;Zeitdauer zu klein für normales Signal, Pause zu kurz für Minutenende => Fehler!
o1050:      jmp k1600           ;Pause > 1050ms => Pause lang genug für Minutensignal
k750:
            JNC o750            ;Carry: Puls < 250ms => Sprung auf nächste Prüfung
            Mov A,#2            ;Ohne Carry: Puls >250ms => Fehler!
            RET
k850:
            JNC o850            ;Carry: Puls < 150ms => Sprung auf nächste Prüfung
            Mov A,#1            ;Ohne Carry: 250ms> Puls >150ms => Bit gesetzt: 1
            RET
k950:
            JNC o950            ;Carry: Puls < 50ms => Sprung auf nächste Prüfung
            Mov A,#0            ;Ohne Carry: 150ms> Puls >50ms => Bit nicht gesetzt: 0
            RET
k1050:
            JNC o1050           ;Carry: Puls < -50ms => Sprung auf nächste Prüfung
            Mov A,#2            ;Ohne Carry: 50ms> Puls >-50ms => Pause und Signal zu kurz: Fehler!
            RET
k1600:
            Mov A,#4            ; Sonst: Pause >1050ms => Minutenwechsel
            RET

                                ;- - - Analyse- - -
                                ;Eintrag    des gelesenen Bits in Speicher, der gesamte Minute darstellt.
                                ;Sendet gelesenes Bit außerdem jede Sekunde an Konsole zur Darstellung.
                                ;R0 ist hier speicher für Bits für die Minute (startet bei 30h)
                                ;- - - - - - - - - - -
Analyse:
            inc R0              ;erhöhe R0 (starte bei 30h) jede s. Pointer für Speicherbereich des
                                ;Minutenbit-Speichers
            CJNE A,#0,ans1      ;wen Akku(aktuelles Bit)=0 schreibe bei 30h="0" bei A=0, sonst nächste Prüfung
            jmp ana0            ;Schreibe 0
ans1:       CJNE A,#1,ans2      ;schreibe 31h="1" bei A=1 wenn Akku =1, sonst ächste Prüfung
            jmp ana1            ;Schreibe 1
ans2:       CJNE A,#2,ans4      ;schreibe 32h="2" bei A=2
            jmp ana2            ;schreibe 2
ans4:       Mov A,#13           ;schreibe Carriage Return
            Call Trans          ;senden an Konsole
            Mov A,#10           ;schreibe Line Feed
            Call Trans          ;senden an Konsole
            Mov A,R3            ;R3 Anfangszähler, der Ausgabe aufhält: benötigt bei Initialisierung und Fehler,
                                ;sodass auf Anfang nächster Minute gewartet wird, bis 4 Ausgegeben
            JZ jmpR3            ;schreibe 4 in display und Setze den Pointer in R0 zurück auf
                                ; "Anfang der Minute" 30h, wenn Zähler 0 erreicht
            DEC A               ;decrementiere Zähler, wenn ungleich 0 (da jump, nicht call)
            Mov R3,A            ;Speichere decrementierten Zähler wieder ab
            RET                 ;Zähler weiterhin herunterzuzählen, daher keine Ausgabe von 4
jmpR3:
            Mov A,#34h          ;schreibe 4
            Mov R0,#30h         ;Speicherbereich 30h-7Fh

            Call Trans          ;Konsole schreiben
            RET
ana0:
            Mov @R0,#0          ;schreibe 0 an Pointer-Position für Minutenbit-Speicher
```

```
            Mov A,#30h          ; schreibe 0 in Konsole
            Call Trans          ;Konsole schhreiben
            RET
ana1:
            Mov @R0,#1          ; " 1 "
            Mov A,#31h          ; schreibe 1 in Konsole
            Call Trans
            RET
ana2:
            Mov @R0,#2          ;" 2 "
            Mov A,#32h          ; schreibe 2 in Konsole
            Call Trans
            RET
                                ;- - - Decoder- - -
                                ;Analysier  Minutenspeicher und stellt die Daten in Textform im Display dar.
                                ;Verschiedene   Display-Teile werden über die Minute verteilt aktualisiert,
                                ;je nach Verfügbarkeit.
                                ;- - - - - - - - - -
Decoder:
            CJNE R3,#0,retR3    ;Abbruch wenn Zähler noch nicht 0 ist: warte bis zur nächsten R3-ten vollen Minute
                                ;(Fehler i=1, Init i=2) bis zur Ausführung.
            jmp retR3
retR3:      RET
retR32:                         ;Zähler=0: führe Dekodierung aus
            Mov A,R0
            CJNE A,#30h,dectag  ;minute fertig gelesen

            Call Cursorhome     ;Cursor auf Anfang des Displays (Teilweise aktualisierung des Display)
                                ;:::::::::stunden:::::::
            Mov R1,#54h
            Mov R5,#7h
            Call pruef          ;Zähle ab R1 R5 (Anzahl) Adressen (herunter) mittels XOr zusammen
            JZ stdfehler        ; ohne Fehler, (AK=0), Dekodiere Daten
            Call fehler         ; sonst gebe Fehler aus und warte bid zur nächsten
            Mov A,#31h          ;Fehlercode 1
            Call Zeichen        ; Fehlercode ausgeben
            RET                 ;Dekodierung beenden
stdfehler:
            Mov R1,#53h         ;34. bit: 34d +30h , Ausgeben der Stunde, 1. Ziffer
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;33. bit
            Add A,#30h
            Call zeichen

            Mov R1,#51h         ;32. bit: 32d +30h ,  Ausgeben der Stunde, 2. Ziffer
            Mov A,@R1           ; 32. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;31. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;30. bit
            RL A
            DEC R1
```

```
            ADD A,@R1           ;29. bit
            Add A,#30h
            Call zeichen

            Mov A,#03ah         ;Doppelpunkt
            Call zeichen
                                ;::::::::minuten::::::::
            Mov R1,#4dh         ;Prüfbit Minute
            Mov R5,#8h
            Call pruef
            JZ fehlermin
            Call fehler
            Mov A,#32h
            Call Zeichen
            RET
fehlermin:
            Mov R1,#4Ch         ;27. bit: 27d +30h, Ausgeben der Minute, 1. Ziffer
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;26. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;25. bit
            Add A,#30h
            Call zeichen

            Mov R1,#49h         ;24. bit: 24d +30h , Ausgeben der Minute, 2. Ziffer
            Mov A,@R1           ; 24. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;23. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;22. bit
            RL A
            DEC R1
            ADD A,@R1           ;21. bit
            Add A,#30h
            Call zeichen

            Mov A,#20           ;leerzeichen
            Call zeichen
                                ;::::::::Tag::::::::
dectag:
            CJNE A,#5bh,decmonat

            Mov R1,#58h         ;39. bit: 39d +30h
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;38. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;37. bit
            RL A                ;shift links
```

```
            DEC R1
            ADD A,@R1           ;36. bit
            Add A,#30h
            Call zeichen

            Mov R1,#5ah         ;41. bit: 41d +30h
            Mov A,@R1           ; 41. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;40. bit
            Add A,#30h
            Call zeichen

            Mov A,#46           ;punkt
            Call zeichen
                                ;::::::::Monat::::::::
decmonat:
            CJNE A,#62h,decjahr
            Mov R1,#62h         ;49. bit: 49d +30h
            Mov A,@R1           ; 49. bit
            Add A,#30h
            Call zeichen

            Mov R1,#61h         ;48. bit: 48d +30h
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;47. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;46. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;45. bit
            Add A,#30h
            Call zeichen

            Mov A,#46           ;punkt
            Call zeichen
                                ;::::::::Jahr::::::::
decjahr:
            CJNE A,#68h,decRET
            jmp jahr
decRET:     RET
jahr:
            Mov R1,#6bh         ;Fehlerbit prüfen: R1 Startadresse, R5: Anzahl zu verwertenden Bits (herabzählend,
                                ;hier ab 59 (6bh). Bit 23 herab bis Bit 36 (P2)
            Mov R5,#23
            Call pruef          ;prüft ab R1 R5 Bits mit Prüfbüt (XOR-Verknüpfung soll 0 sein).
                                ;Gibt Ergebnis in Akku aus.
            JZ fehlerdatum      ;Prüfung=0: kein Fehler
            Call fehler         ;sonst Fehler-routine! Display-Azeige und Warte bis zur nächsten vollen Minute
            Mov A,#33h          ;Fehlercode 3 (Fehler nummeriert zur Identifikation)
            Call Zeichen        ;Darstellen auf Display
            RET
fehlerdatum:                    ;Analyse des Datums
```

```
            Mov R1,#6ah         ;57. bit: 57d +30h
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;56. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;55. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;54. bit
            Add A,#30h
            Call zeichen

            Mov R1,#66h         ;53. bit: 53d +30h
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;52. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;51. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;50. bit
            Add A,#30h
            Call zeichen

            Call zeile2
                                ;::::::WTag::::::::::
            Mov R1,#5dh         ;44. bit: 39d +30h
            Mov A,@R1
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;43. bit
            RL A                ;shift links
            DEC R1
            ADD A,@R1           ;42. bit

            CJNE A,#0,montag    ;sunday So, nacheinander auf Gleichheit von 0 (Sonntag) bis 6 (Samstag) prüfen
            Mov A,#53h
            Call zeichen
            Mov A,#6fh
            Call zeichen
            RET
montag:     CJNE A,#1,dienst    ;montag Mo
            Mov A,#4dh
            Call zeichen
            Mov A,#6fh
            Call zeichen
            RET
dienst:     CJNE A,#2,mittw     ;dienstag Di
            Mov A,#44h
            Call zeichen
            Mov A,#69h
            Call zeichen
```

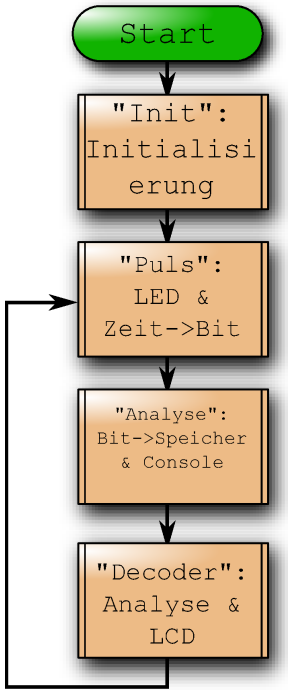
```
            RET
mittw:      CJNE A,#3,donner    ;mittwoch Mi
            Mov A,#4dh
            Call zeichen
            Mov A,#69h
            Call    zeichen
            RET
donner:     CJNE A,#4,freit     ;donnerstag Do
            Mov A,#44h
            Call zeichen
            Mov A,#6fh
            Call zeichen
            RET
freit:      CJNE A,#5,samst     ;freitag Fr
            Mov A,#46h
            Call zeichen
            Mov A,#72h
            Call zeichen
            RET
samst:                          ;Samstag Sa
            Mov A,#53h
            Call zeichen
            Mov A,#61h
            Call zeichen
decende:
            RET
                                ;- - - Trans- - -
Trans:
            mov SBUF,A          ;Senden des Zeichens im Accu
loop2:      JNB TI,loop2        ;Warten bis Zeichen vollständig gesendet wurde
            clr TI
            RET
                                ;- - - Zeile2- - -
zeile2:
            Mov A,#11000000b    ;befehl für Sprung in 2. Zeile
            Call befehl
            RET
                                ;- - - Cursorhome- - -
Cursorhome:
            Mov A,#00000010b    ;befehl für Cursor auf Homeposition
            Call befehl
            RET
                                ;- - - cleardisplay- - -
cleardisplay:
            Mov A,#00000001b    ;befehl zum löschen des display
            Call befehl
            RET
                                ;- - - befehl- - -
befehl:
            clr RS              ;befehl (keine Daten)
            clr RW              ;schreiben
            setb E
            Mov P5,A            ;zu Schreibenden Befehl an Portpins anlegen
            clr E               ;erzeugt fallende Flanke
            Call delay5         ;Warte bis befehl gesendet wurde
            Ret
```

```
                        ;- - - zeichen- - -
zeichen:
            setb RS             ;zeichen (kein befehl)
            clr RW              ;schreiben
            setb E
            Mov P5,A            ;zu Schreibendes Zeichen an Portpins anlegen
            clr E               ;erzeugt fallende Flanke
            Call delay5         ;Warte bis befehl gesendet wurde
            Ret
                        ;- - - delay5- - -
delay5:
            Call timerinit      ; Zurücksetzen und neustarten des Timers
Repeat:     JNB TF0,Repeat      ; 5ms bis zum Timerüberlauf
            RET
                        ;- - - - - - - - - - timerinit- - - - - - - - - -
timerinit:
            mov TH0,#0ECh       ;max-Wert - 5000 (5 ms bis überlauf)
            mov TL0,#082h       ;high-byte/low-byte
            Clr TF0             ;Überlauf-Bit leeren
            Mov TMOD,#00000001b ;Timer1: nicht benutzt, Timer0: Modus1
            Setb TR0            ;Timer0 Start!
            RET
                        ;- - - init- - -
init:       clr SM0             ;Seriellеschnittstelle in Modus 1
            setb SM1
            setb REN            ;Seriellen Empfang starten
            setb BD             ;internen Bautratengenerator aktivieren
            Mov A,PCON          ;setzen des Nichtbitadressierbaren SMOD
            Setb Acc.7
            Mov PCON,A

            Call timerinit
                        ;- - - LCD  initialisierung- - -
            Mov A,#00110000b    ;Software-Reset: function set
            Call befehl
            Mov A,#00110000b
            Call befehl
            Mov A,#00110000b
            Call befehl
            Mov A,#00111100b    ;function set: 2 Lines, 5*10 dots
            Call befehl
            Mov A,#00001110b    ;Display on, cursor on, blink on
            Call befehl
            Mov A,#00000001b    ;display clear
            Call befehl
            Mov A,#00000110b    ;entry mode, inkrement, shift aus
            Call befehl
                        ;- - - - - -
            Mov R3,#2           ;Bei Initialisierung: Warte 1 Min-Signal bis Minutenstart + 1 Min-Signal für
                                ;vollständige Displayanzeige
            Call uhrinit        ;Zeie Init" an
            RET
uhrinit:
            Mov A,#69h          ;i
            Call Zeichen
            Mov A,#6Eh          ;n
```

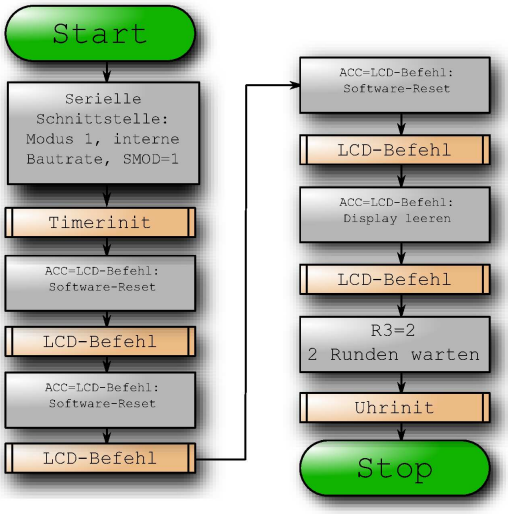
```
            Call Zeichen
            Mov A,#69h          ;i
            Call Zeichen
            Mov A,#74h          ;t
            Call Zeichen
            RET
fehler:
            Call cleardisplay
            Mov A,#46h          ;F
            Call Zeichen
            Mov A,#65h          ;e
            Call Zeichen
            Mov A,#68h          ;h
            Call Zeichen
            Mov A,#6Ch          ;l
            Call Zeichen
            Mov A,#65h          ;e
            Call Zeichen
            Mov A,#72h          ;r
            Call Zeichen
            mov R3,#1           ;Warte bis zum  nächsten Min-Signal (doppelte Sekundenpause). Zähler wird bei
                                ;nächstem Analyse-Versuch abgerufen, bei 0 analysiert,
                                ;sonst dekrementiert und bis zum nächsten gewartet
            RET

                                ;- - - Prüfbits analysieren- - -
pruef:                          ;Prüfbits analysieren
            Mov A,#0            ;Akku=0
prflp:      XRL A,@R1           ;Bit an Speicheradresse R1 wird XOR verknüpft (Bitweise) mit Akku
            DJNZ R5,prflp2      ;Anzahl zu lesende Bits. 0: fertig, sonst nächster Speicherstelle => Akku
            RET                 ;Akku beinhaltet das Ergebnis der bitweisen XOR-Verknüpfung aller Bits.
prflp2:     DEC R1              ;nächste Speicherstelle (herabzählend) betrachten
            jmp prflp           ;Weiter verknüpfen und erneut prüfen
End
```

Flussdiagramm

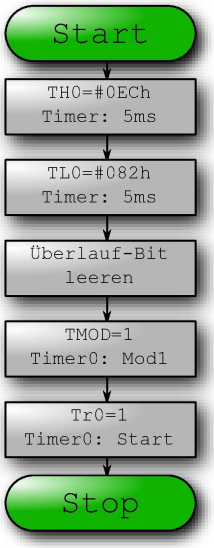
(a) Gesamtablauf



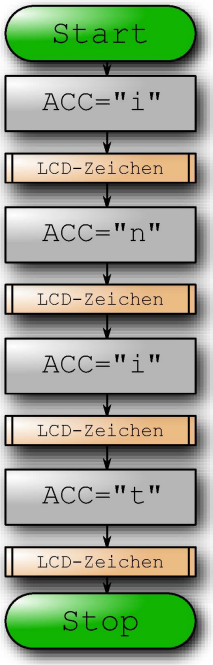
(b) Init



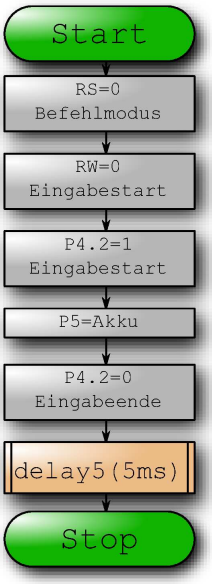
(c) Timerinit



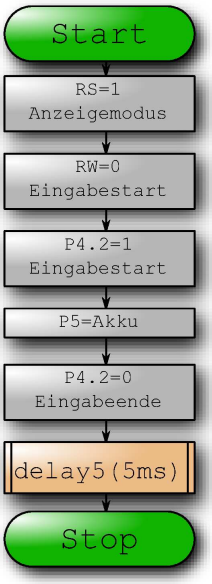
(d) Uhrinit



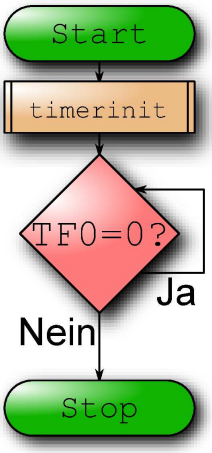
(e) LCD-Befehl



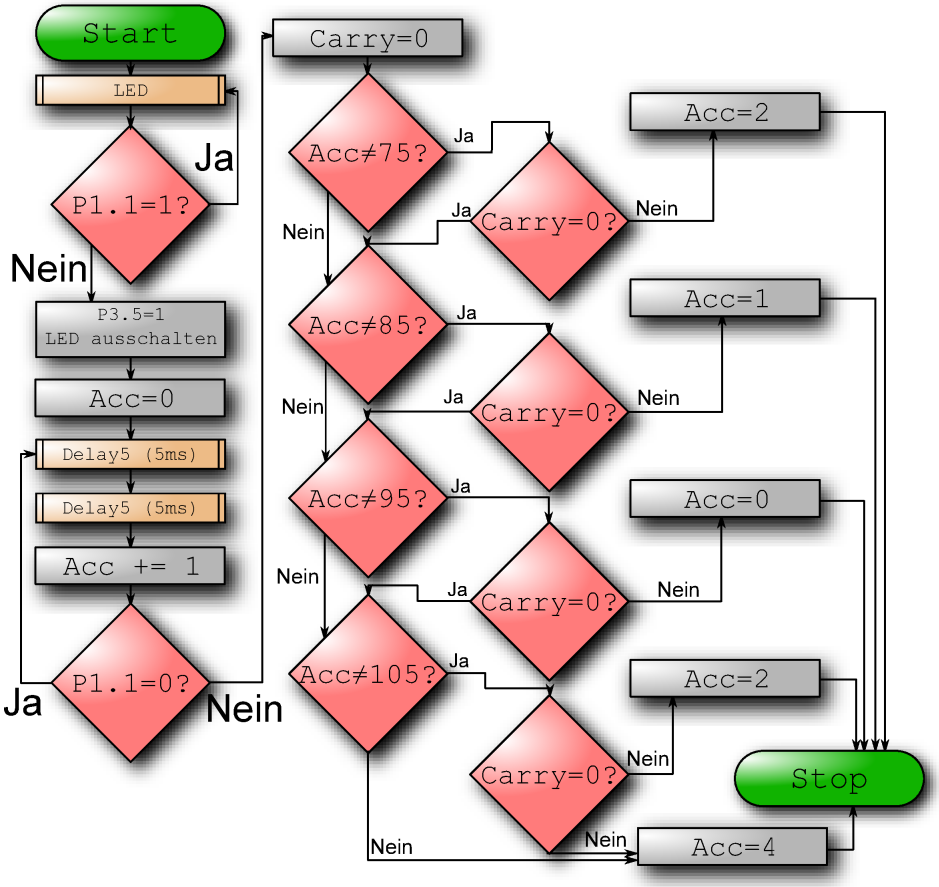
(f) LCD-Zeichen



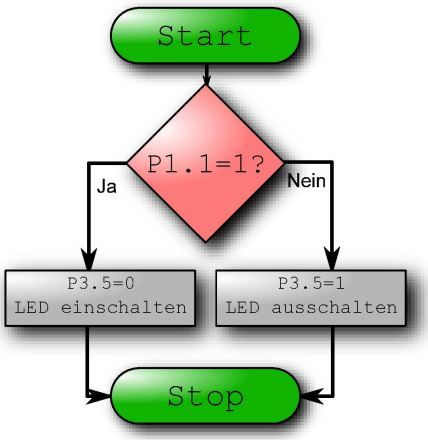
(g) delay (5ms)



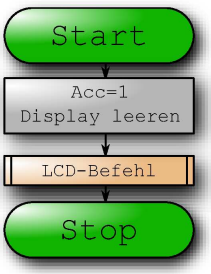
(h) Puls



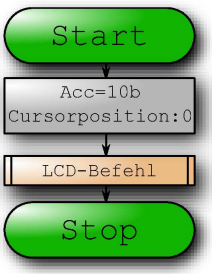
(i) LED



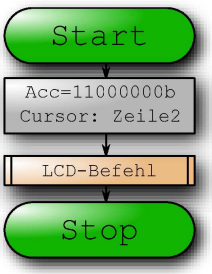
(j) Cleardisplay



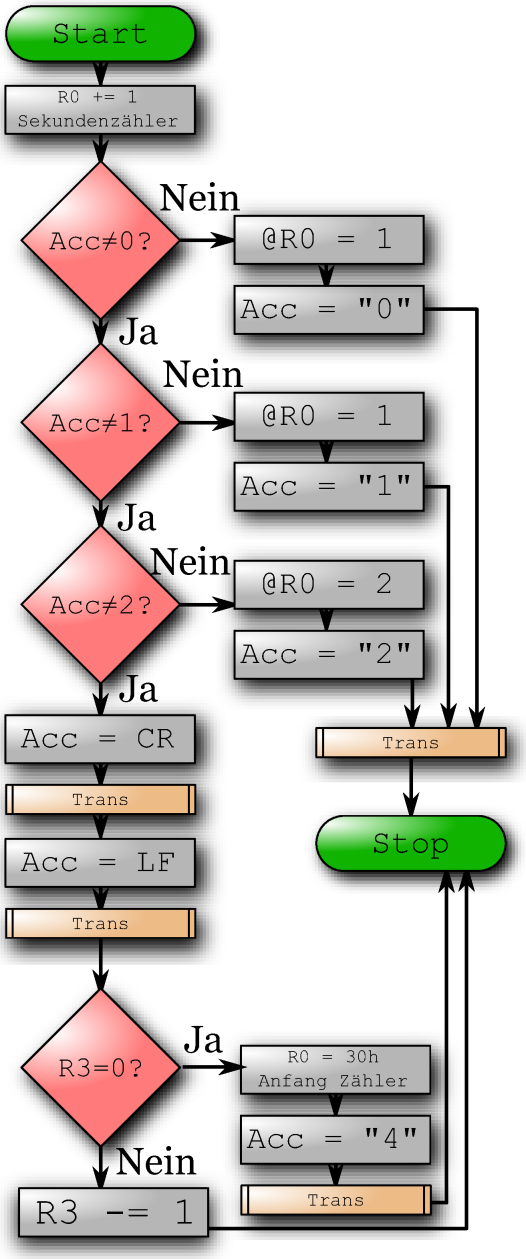
(k) Cursorhome



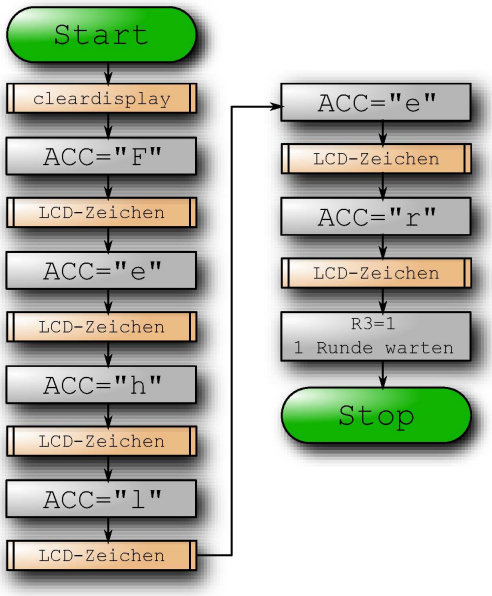
(l) Zeile 2



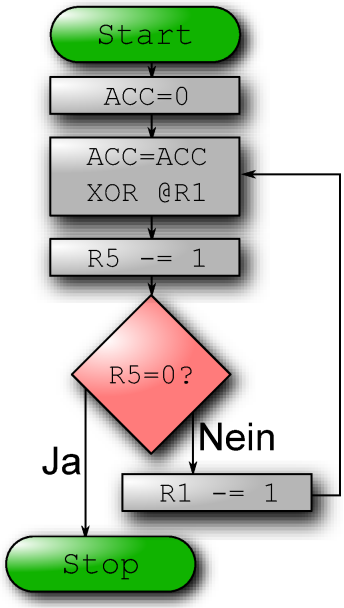
(m) Analyse



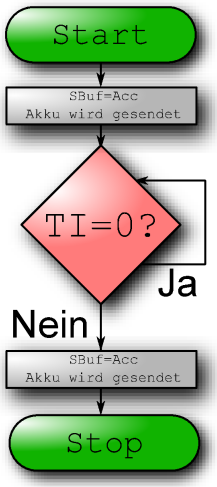
(n) Fehler



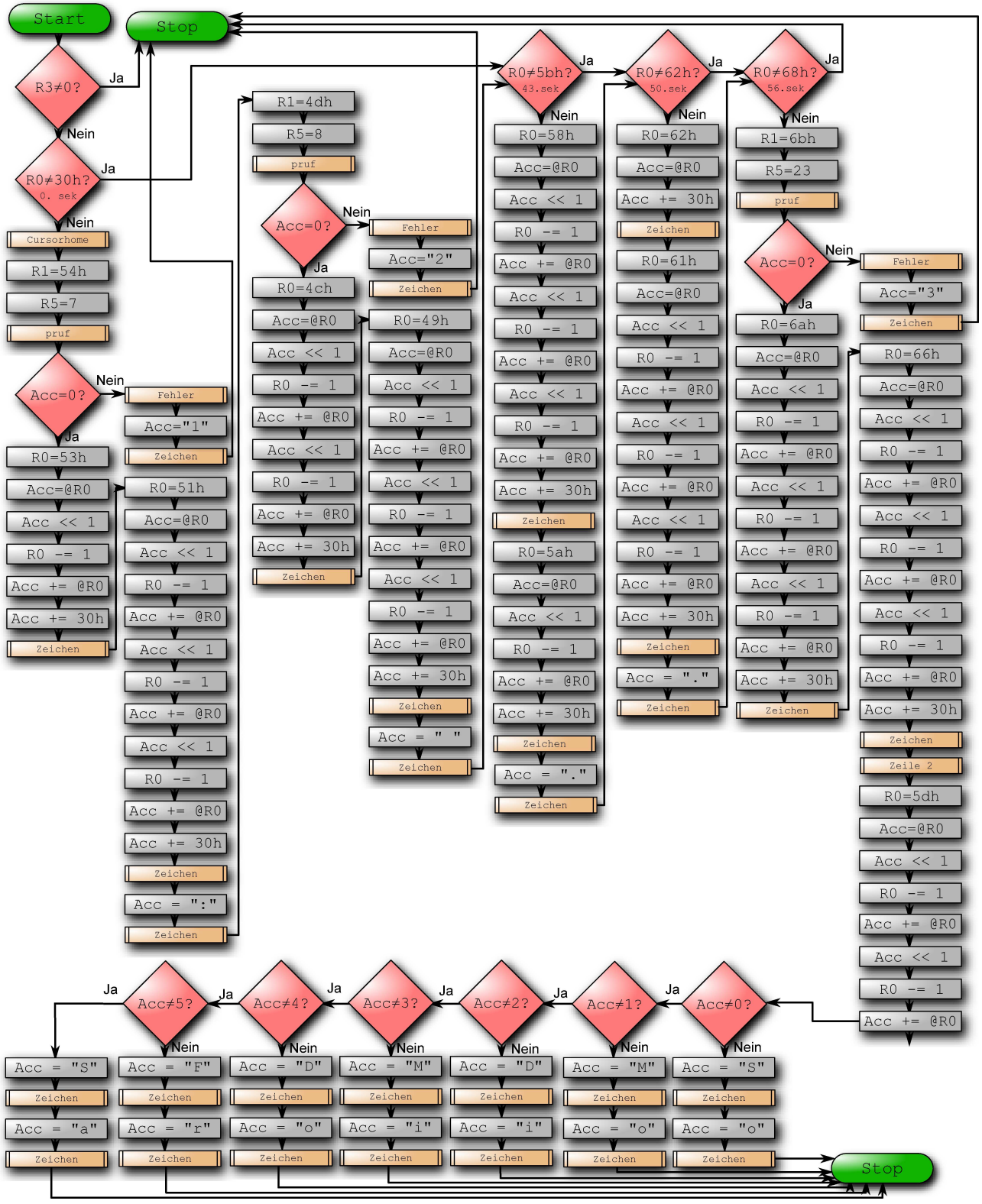
(o) Pruf



(p) Trans



(q) Decoder


Start
Stop
R3≠0?
Ja
Nein
R0≠30h?
0. sek
Cursorhome
R1=54h
R5=7
pruf
Acc=0?
Fehler
Acc="1"
Zeichen
R0=53h
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
R0=51h
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
Acc = ":"
Zeichen
R1=4dh
R5=8
pruf
Acc=0?
Fehler
Acc="2"
Zeichen
R0=4ch
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
R0=49h
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
Acc = " "
Zeichen
R0≠5bh?
43.sek
R0=58h
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
R0=5ah
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
Acc = "."
Zeichen
R0≠62h?
50.sek
R0=62h
Acc=@R0
Acc += 30h
Zeichen
R0=61h
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
Acc = "."
Zeichen
R0≠68h?
56.sek
R1=6bh
R5=23
pruf
Acc=0?
Fehler
Acc="3"
Zeichen
R0=6ah
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
R0=66h
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc += 30h
Zeichen
Zeile 2
R0=5dh
Acc=@R0
Acc << 1
R0 -= 1
Acc += @R0
Acc << 1
R0 -= 1
Acc += @R0
Acc≠0?
Acc≠1?
Acc≠2?
Acc≠3?
Acc≠4?
Acc≠5?
Acc = "S"
Acc = "M"
Acc = "D"
Acc = "i"
Acc = "o"
Acc = "r"
Acc = "F"
Acc = "a"
Acc = "S"
Acc = "o"
Acc = "i"
Acc = "M"
Acc = "D"
Acc = "o"
Acc = "r"
Acc = "F"
Zeichen
Stop