5. Einige Details der Ports

Als **Ports** werden die **"Ein- und Ausgabekanäle"** eines Mikrocontrollers bezeichnet. Sie sind nur für **Digitalsignale** geeignet, lediglich **Port 6** dient zur **Analog-Eingabe für den integrierten A - D Wandler**.

Achtung: weiterhin gelten folgende Einschränkungen:

a) **Port 0 und Port 2 sind bereits belegt**, da sie den **Adress- und Datenbus** zur Kommunikation mit den Speicherbausteinen oder irgendwelcher Peripherie darstellen.

b) **2 Leitungen von Port 3** bilden die **Serielle Schnittstelle** , deshalb können nur noch einzelne Bits dieses Ports für andere Zwecke verwendet werden. Weitere 2 Pins liefern die Signale #WRITE und #READ zum Ansteuern des XRAMs. Ein Pin ist durch das Signal „PSEN“ zum Auslesen des EPROMs belegt.

c) Damit stehen dem Anwender nur die **Ports 1, 4 und 5** zum beliebigen Gebrauch in voller Breite zur Verfügung....

d) Außerdem sollte man wissen, dass bei **mehreren Ports noch eine “Drittfunktion**” vorgesehen ist (z. B. Timer- oder Zähleingang, externe Interrupts usw.) Hier hilft das Datenblatt bzw. der Stromlaufplan des Boards weiter!

Details eines Ports im “Normalbetrieb”:
Jeder Port ist so breit wie der Datenbus (hier: 8 Bit = 1 Byte) und besteht grundsätzlich aus 2 Teilen:

a) einem **8-Bit-Register** im SFR-Bereich des Controllers ab Adresse 80H, bei dem sich die **einzelnen Bits** setzen oder löschen lassen (= "bitadressierbarer Bereich").

b) einer **kleinen Elektronik** an jedem einzelnen "Portpin", d. h. zwischen dem entsprechenden IC-Beinchen und den internen Leitungen des Controllers.

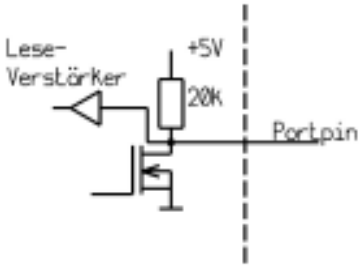
Die **Ausgabe eines Bytes** bei einem solchen Port erfolgt mit dem Befehl

MOV P1,A (= gebe den Akkuinhalt an Port 1 aus).

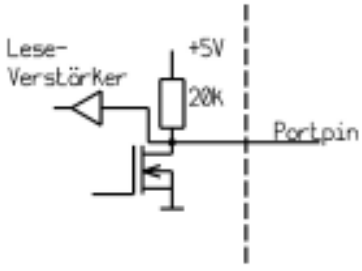
Dadurch wird die Information erst in das entsprechende SFR-Register geladen und anschließend über die Pin-Elektronik an die Anschlüsse des Controllers durchgeschaltet.
(Fast) ebenso einfach verläuft das **Einlesen** einer Information, die einem Port zugeführt wird. Die Einschränkung "fast" ist deshalb nötig, weil hierbei eine kleine **Vorsichtsmaßnahme** nötig ist:

Die Ausgangsstufe bei der Portpin-Elektronik enthält nämlich einen Feldeffekt-Transistor, der für die Ausgabe von "LOW-Pegel" eingeschaltet wird.

Ausgabe von HIGH



Ausgabe von LOW



Legt man nun von außen her HIGH-Pegel an diesen Pin, wenn der Port LOW-Pegel ausgibt, so passiert grundsätzlich ein **Unglück**:
Oft wird der FET (und meist die halbe Controller-Elektronik) zerstört, weil **über den durchgeschalteten FET ein zu großer Strom nach Masse fließt (Kurzschluss!).** Ist eine Strombegrenzung vorhanden, dann bleibt wenigstens der Controller ganz. Aber sicherlich funktioniert zumindest der gewünschte Programmschritt nicht...
Der **sichere Weg zum richtigen Einlesen** besteht deshalb (in Assembler!) aus **zwei** Befehlen:

```
MOV P1,#11111111B   ;Schreibe lauter Einsen in Port 1, dadurch sperren alle Ausgangs-FETs
MOV A,P1            ;Lese jetzt die Information in den Akku, die an Port 1 anliegt
```

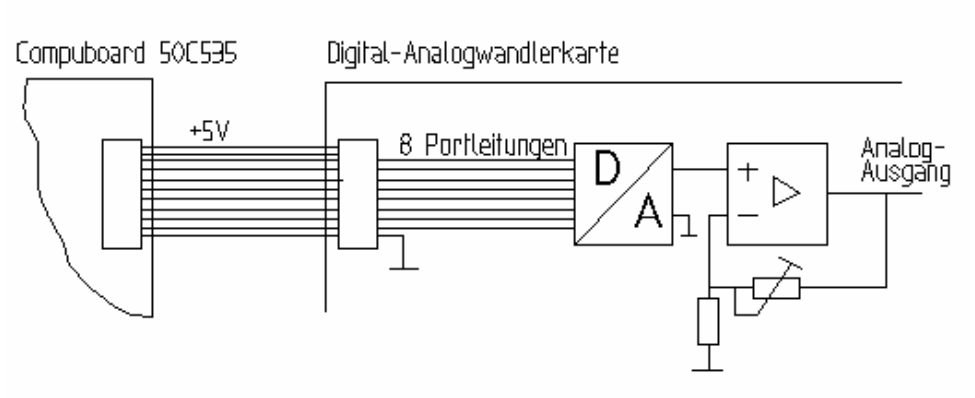
In „C“ sind ebenfalls zwei Zeilen in der „main“-Funktion nötig, wenn Port P1 in die Variable „x“ kopiert werden soll:

```
P1=0xFF;
X=P1;
```

6. Erzeugung von Analogsignalen mit dem Mikrocontroller

6.1. Grundlagen

Als Eigenentwicklung steht in der EST eine “Analogplatine” zur Verfügung. Sie wird über ein 10-poliges Flachbandkabel an einen Controllerport angeschlossen und enthält einen 8 Bit-Digital-Analog-Wandler-IC mit nachfolgendem Pufferverstärker:



Der Digital-Analogwandler selbst besitzt eine Auflösung von

10 Millivolt pro Bit.

Damit erhält man bei einer Datenbusbreite von 8 Bit an seinem Ausgang eine Gleichspannung, die sich in Schritten von 10 mV zwischen

Null x 10mV = Null Volt **und** **255 x 10mV = 2,55 V**

ändern lässt. Der nachfolgende Pufferverstärker liefert schließlich beim Verdrehen des Potis eine Verstärkung zwischen V = 1 und V = 3.

Gibt man nun am Controllerport irgendeine Hexadezimalzahl aus, so erhält man nach etlichen Mikrosekunden den zugehörigen Gleichspannungswert.

Aufgabe: Die D-A-Platine wird an Port 1 über ein 10poliges Flachbandkabel mit 2 Pfostenfeldsteckern angeschlossen. Anschließend wird ein neues µvision2-Projekt „Analogsignale“ angelegt und darin das “kürzeste Assemblerprogramm der Welt” erstellt.
Bitte dieses Programm unter dem Namen **„da_test1.a51“** darin speichern, assemblieren, linken, testen und auf dem Oszilloskopschirm seine Aufgabe beobachten.

Programmkern:

```
loop1:    inc P1
          jmp loop1
```

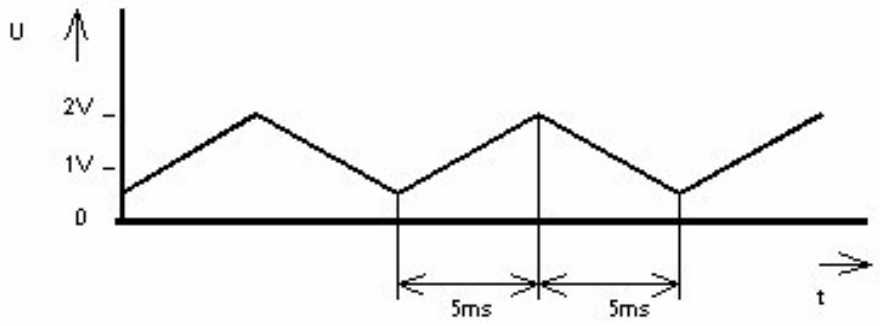
Wiederholen Sie diese Übung mit einem passenden C-Programm!

6.2. Programmierung einer Dreieckspannung

Aufgabe: Schreiben Sie ein Programm, das eine Dreieckspannung mit folgenden Eigenschaften am Ausgang des Digital-Analog-Wandlers erzeugt:

U_{min} ca. 0,5V U_{max} ca. 2V f = 100Hz

Lösen Sie dieses Problem in Assembler und in „C“!



Dreieckspannung / Lösungsvorschlag in Assembler:

```
$nomod51                                ;8051- Standard – Modus ausschalten
$include(REG515.INC)                    ;dafür 80515 bzw. 80535 wählen

OFFSET      EQU 4000h                   ;Adressoffset von 4000h bei unserem Compuboard nötig

ausgang     EQU   P1                    ;Port P1 heißt nun „ausgang“

?Stack      SEGMENT IDATA               ;Definiere den Stack und reserviere 5 Bytes dafür
            RSEG ?STACK
            DS 5

CSEG AT 0+OFFSET                        ;Hauptprogramm soll bei Adresse 4000h beginnen
            jmp START                   ;reset location (= Einsprung beim Reset oder Neustart)

DREIECK     SEGMENT CODE                ;Hauptprogramm soll “Dreieck” heißen
            RSEG DREIECK                ;”Dreieck” aufrufen und Programmcodes dafür schreiben

START:      mov SP,#?STACK-1            ;Am Anfang steht immer die Stack-Initialisierung
            mov ausgang, #50            ;Schicke die Zahl 50 als Minimalspannung zum Ausgang

Label3:     mov R7, #150                ;Sehe 150 Wiederholungen des folgenden Programmteiles vor

Label1:     inc ausgang                 ;Inkrementiere den Ausgang und
            call zeit                   ;rufe die Zeitschleife auf
            djnz R7, Label1             ;Dekrementiere R7 und prüfe, ob schon Null erreicht

            mov R7, #150                ;Sehe wieder 150 Wiederholungen vor
Label2:     dec ausgang                 ;Dekrementiere den Ausgang und
            call zeit                   ;rufe die Zeitschleife auf
            djnz R7, Label2             ;Dekrementiere R7 und prüfe, ob schon Null erreicht

            jmp Label3                  ;Wiederhole alles

zeit:       mov R6, #12                 ;Unterprogramm „Zeit“
            djnz R6, $                  ;Lade R6 mit der Zahl 12 und zähle den Inhalt bis auf Null herunter
            nop                         ;Tue 1 Mikrosekunde lang nix
            ret                         ;Ende des Unterprogramms
end
```

Dreieckspannung / Lösungsbeispiel in C:

```
#include <reg515.h>                     // 80535-Registersatz als header bereitstellen
#include <stdio.h>                      // Universal – C – header bereitstellen und einbinden

sfr ausgang=0x90;                       // Port P1 heißt nun „ausgang“

void zeit(void);                        // Zusatzfunktionen außer „main“ als Prototyp anmelden

void main(void)                         // Hauptfunktion heißt immer „main“
{   int x;                              // Integer-Variable x definieren
    while(1)                            // Endlosschleife
    {   for(x=50;x<=150;x++)            // x von 50 bis 150 inkrementieren und
            {   ausgang=x;              // jedes Mal den Wert von x auch den Ausgang ausgeben
                zeit();                 // und jedes Mal hinterher eine Pause einlegen
            }
        for(x=150;x>=50;x--)            // x von 150 bis 50 dekrementieren und
            {   ausgang=x;              // jedes Mal den Wert von x auch am Ausgang ausgeben
                zeit();                 // und jedes Mal hinterher eine Pause einlegen
            }
    {
}

void zeit()                             // Funktion  Zeitverzögerung
{   int y;                              // Integer-Variable y einfach nur hochzählen
    for(y=0;y<=1000;y++);
}
```

6.3. Erzeugung periodischer Kurvenformen mit Hilfe von Wertetabellen

Wenn die Frequenz bzw. Amplitude der erzeugten Analogsignale (bei unveränderter Kurvenform) leicht veränderbar sein soll oder wenn die Kurvenform selbst recht kompliziert ist, dann arbeitet man lieber mit folgender Methode:

Man "**digitalisiert" den Verlauf der Kurve**, d. h., man zerlegt sie in viele Einzelabschnitte und bestimmt in jedem Abschnitt den genauen Amplitudenwert. Die so entstehende Tabelle **brennt man** hinter den Programmcodes ebenfalls mit in das **EPROM**. Von dort werden die einzelnen Werte mit dem Befehl

movc a,@a+dptr

in den Akkumulator geholt und können bei Bedarf **maximal drei Behandlungen unterzogen werden**:

a) Zur Vergrößerung oder Verkleinerung der **Ausgangsspannungsamplitude** wird jeder abgeholte Wert mit einem **bestimmten Faktor multipliziert**, der die **Amplitude** angibt und durch die Software vorgegeben werden muss.

b) Für eine **bestimmte Wiederholfrequenz** ist bei jedem Einzelschritt eine kurze **Warteschleife** nötig. Soll die **Frequenz veränderbar** sein, dann muss sich die **Wartezeit** für jeden Schritt per Software **neu einstellen** lassen.

c) Möchte man zusätzlich noch die **"Mittellinie verschieben"** (das wäre bei einer Mischspannung der Gleichspannungs- oder DC-Anteil), dann wird bei jedem Schritt vor der Ausgabe an den DA-Wandler eine **konstante Zahl addiert oder subtrahiert**.

Diese Tabelle wird nun einfach immer wieder vom Anfang bis zum Ende ausgelesen, um ein periodisches Signal zu erzeugen.

6.3.1. Treppenspannung

Beispiel: Ein Analogsignal soll aus 10 abgespeicherten Werten bestehen, die nacheinander aus einer Tabelle ausgelesen und an Port P1 ausgegeben werden. Legen Sie nach jeder Werteausgabe eine Wartezeit von 500 Mikrosekunden ein. Schreiben Sie das Programm erst in Assembler, dann in C.

Die abgespeicherten Werte des Signals lauten: 0,25,50,75,100,125,150,175,200,225

a) Assembler.-Lösung:

```
;analog1.a51
;Erstellt am 01. Januar 2005 durch G. Kraus
;-------------------------------------------------------------------------------
$NOMOD51                        ; 8051-Standard-Modus abschalten
$INCLUDE (REG515.INC)           ; dafür 80535-Registersatz bereitstellen

NAME    ANALOG1                 ; Dieser Modul soll "ANALOG1" heißen

OFFSET  EQU 4000h               ; Compuboard-RAM beginnt bei 4000h
ausgang EQU P1                  ; Port P1 heißt nun "ausgang"

?STACK  SEGMENT IDATA           ; Stack soll im IDATA-Bereich sein.
        RSEG ?STACK             ; Schalte auf ?Stack-Segment um.
        DS 5                    ; Reserviere 5 Bytes für den Stack.

CSEG    AT 0+OFFSET             ; Programm-Codes beginnen bei 4000h
        jmp   START             ; reset location (jump to START)
;===============================================================
;Erst hier beginnen die Assembler-Programmteile (= Codes)
;===============================================================
teil1       SEGMENT   CODE      ; Erster Programmteil heißt "teil1"
            RSEG teil1          ; Rufe dieses Codesegment zum Schreiben der Programmschritte auf


START:      mov   SP,#?STACK-1  ; Zuweisung der Stack-Adresse
            mov dptr,#tabelle   ; Anfangsadresse der Tabelle in DPTR laden
loop2:      mov r7,#10          ; Register R7 mit der Zahl 10 laden
            mov r6,#0           ; Register R6 löschen
loop:       mov a,r6            ; Register R6 in den Akku kopieren
            movc a,@a+dptr      ; Ersten Tabellenwert in den Akku holen
```

```
                mov ausgang,a           ; und an Port 1 ausgeben
                call zeit               ; Zeitschleife 500 Mikrosekunden aufrufen
                inc r6                  ; inkrementiere R6, um den nächsten Wert auszugeben
                djnz r7,loop            ; Schon 10 Werte ausgegeben?
                jmp loop2               ; Nach 10 Werten wieder Tabelle neu auslesen
;-------------------------------------------------------------------------
;Unterprogramme
;-------------------------------------------------------------------------
teil2           SEGMENT CODE    ; Zweiter Programmteil heißt "teil2"
                RSEG teil2      ; Schalte auf dieses Codesegment

zeit:           mov r5,#250     ; Register R5 mit der Zahl 250 laden
                djnz r5,$       ; R5 dekrementieren, bis Null erreicht
                ret             ; Das ergibt 500 Mikrosekunden!

tabelle:        db 0,25,50,75,100,125,150,175,200,225
;
end
```

---

b) Lösung in C:

```
/*-----------------------------------------------------------------------------------
Programmbeschreibung
-----------------------------------------------------------------------------------
Name:       anal1_c.c
Funktion:   Erzeugung eines Analogsignals anhand einer      Wertetabelle und Ausgabe an Port P1

Datum:      01.01.2005
Autor:

-----------------------------------------------------------------------------------
Deklarationen und Konstanten
-----------------------------------------------------------------------------------*/

#include <reg515.h>
#include <stdio.h>
sfr ausgang=0x90;                                       //Port P1 als Ausgang
/*-----------------------------------------------------------------------------------
Hauptprogramm
-----------------------------------------------------------------------------------*/

void zeit(void);                            //Alle benützten Funktionen außer „main“ als Prototypen anmelden

void main(void)
{
    unsigned char tabelle[10]={0,32,64,128,255,128,64,32,0,0};      // Array im internen RAM1 anlegen

    while(1)                                                        // Endlosschleife
    {
        unsigned char x;                                            // Variable x dient als Pointer
          for(x=0;x<=9;x++)                                         // Ausgabe von 10 Werten programmieren
        {
            ausgang=tabelle[x];                                     // Aktuellen Arraywert zum Ausgang schicken
            zeit();                                                 // Wartezeit aufrufen
          }
    }
}
/*-----------------------------------------------------------------------------------
Zusatzfunktionen
-----------------------------------------------------------------------------------*/
void zeit(void)                                                     // Zeitschleife
{
    int y;                                                          // y als Zählvariable
    for(y=0;y<=25000;y++);                                          // y wird einfach hochgezählt und verbraucht
}                                                                   // dadurch wird Zeit gewonnen
```

6.3.2. EKG-Signal

Aufgabe aus der Medizintechnik: Erzeugen Sie mit dem Mikrocontroller ein **"EKG - Signal**" für 60 Pulsschläge in der Minute und mit einem Spitze-Spitze-Wert vom 3,5 Volt. Erstellen Sie ein Assembler- und ein C-Programm!

Anleitung:

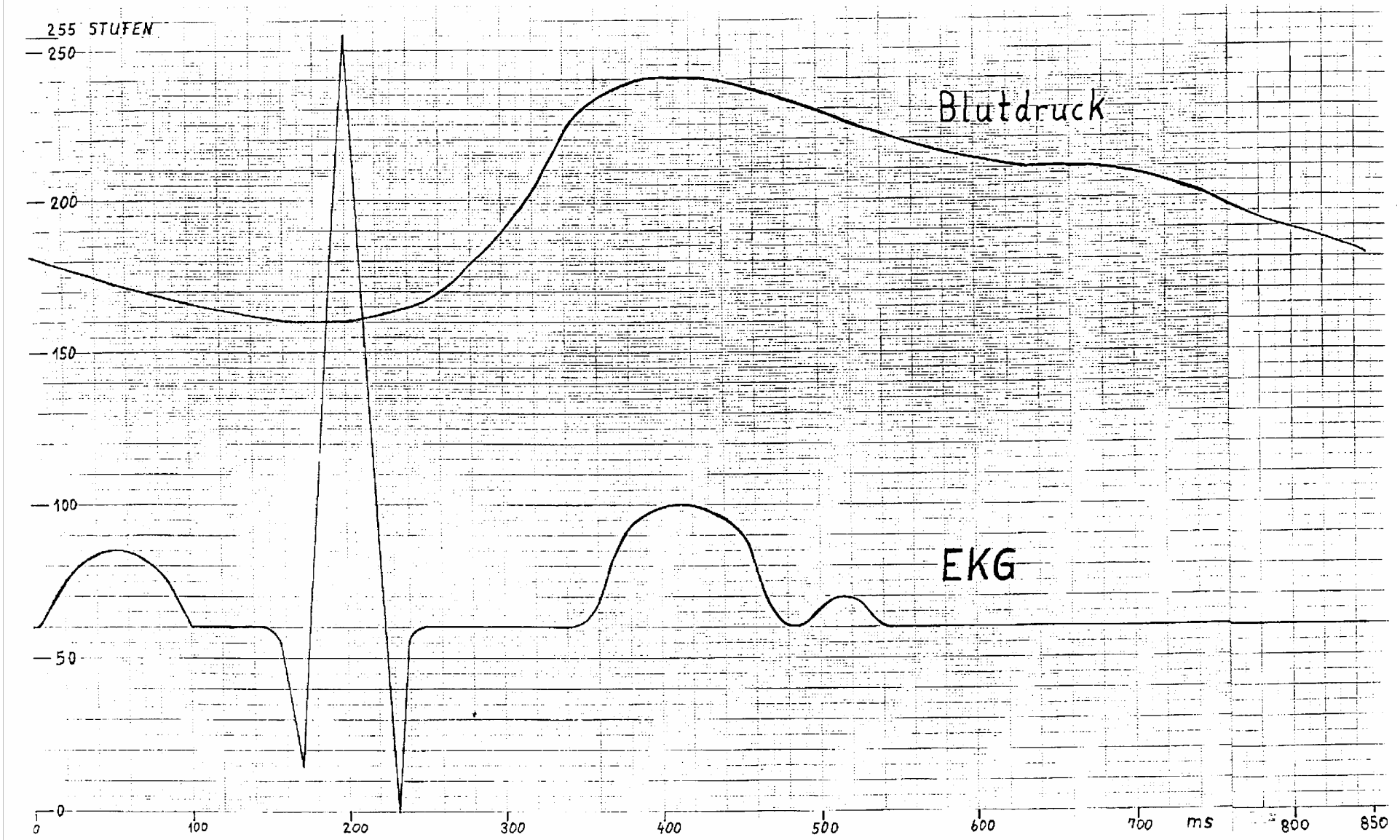
a) Digitalisieren Sie dazu "von Hand" den Verlauf der beigefügten Musterkurve mit einer Auflösung von 8 Bit (= 256 Stufen) bei der Amplitude und 100 Messwerten pro Periode. Tragen Sie die erhaltenen **Bitwerte** nach dem folgendem Schema in die Tabelle ein. Die Tabelle wird später mit einem passenden Label (z. B. "tabelle1:") und der Direktive "**DB**" (= define Byte) in ein Assemblerprogramm eingebunden.

Hinweis: Wenn beim Eintippen der Tabelle in den Editor eine Zeile auf dem Bildschirm nicht ausreicht, dann beginnen Sie bitte die nächste Zeile wieder mit der Direktive „DB".

b) Erstellen Sie ein Hauptprogramm, durch das immer ein Messwert aus der Tabelle geholt und anschließend die passende Wartezeit erzeugt wird.

c) Schreiben Sie das Unterprogramm für die nötige Zeitverzögerung bei jedem Schritt.

Messpunkt:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Messwert:															
Messpunkt:	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Messwert															
Messpunkt:															
Messwert															
Messpunkt:															
Messwert															
Messpunkt:															
Messwert															
Messpunkt:															
Messwert															
Messpunkt:															
Messwert															


255 STUFEN
250
200
150
100
50
0
Blutdruck
EKG
0
100
200
300
400
500
600
700
ms
800
850

EKG-Lösungsvorschlag in Assembler:

```
;ekg.a51
$nomod51
$include(REG515.INC)

OFFSET      EQU   4000h
ausgang     EQU   P1
?Stack      SEGMENT IDATA
            RSEG ?STACK
            DS 5

CSEG AT 0+OFFSET
            jmp START

teil1       SEGMENT CODE          ; Erstes verwendetes Codesegment heißt “teil1”
            RSEG teil1            ; Rufe dieses Segment zum Schreiben der Programmschritte auf

START:      mov SP,#?STACK-1      ; Stack initialisieren
            mov dptr,#tabelle     ; Startadresse der Tabelle in den Datapointer laden
loop1:      mov r7,#100           ; 100 Werte für die Ausgabe vorsehen (= Schleifenzähler)
            mov r6,#0             ; Offset-Adresszähler auf „Null“ stellen
loop:       mov a,r6              ; Ersten Wert der Tabelle in den Akku holen
            movc a,@a+dptr
            mov ausgang,a         ; Ersten Wert der Tabelle am „ausgang“ ausgeben
            call zeit             ; Zeitverzögerung aufrufen
            inc r6                ; Offset-Adresszähler inkrementieren
            djnz r7,loop          ; Falls Schleifenzähler noch nicht Null: nochmals alles wiederholen
            jmp loop1             ; Endlosschleife: nach 100 Werten wieder von vorn anfangen



teil2       SEGMENT CODE          ; Zweites verwendetes Codesegment heißt „teil2“
            RSEG teil2            ; Rufe dieses Segment zum Schreiben der Programmschritte auf

zeit:       mov r4,#20            ; Sehe 20 Wiederholungen vor
loop2:      mov r5,#250           ; Lade Register R5 mit der Zahl 250
            djnz r5,$             ; Zähle Register R5 bis auf Null herunter
            djnz r4,loop2         ; Wiederhole diesen Zählvorgang 20 Mal
            ret                   ; Zurück zum Hauptprogramm


tabelle:    db 65,75,80,82,85,84,82,78,70,60,60,60,60,60,58
            db 45,15,105,170,255,165,80,0,50,60,60,60,60,60,60
            db 60,60,60,60,62,70,82,92,96,99,100,99,97,95,91
            db 80,66,60,60,65,70,69,65,60,60,60,60,60,60,60
            db 60,60,60,60,60,60,60,60,60,60,60,60,60,60,60
            db 60,60,60,60,60,60,60,60,60,60,60,60,60,60,60
            db 60,60,60,60,60,60,60,60,60,60

end
```

EKG-Lösungsvorschlag in C:

```
/#include <reg515.h>                          // Header
#include <stdio.h>

sfr ausgang=0x90;                            // Spezialfunktions-Register P1 heißt nun "ausgang"

void zeit (void);                            // Prototyp anmelden

void main (void)
{
    code unsigned char EKG [100]=                                   // Array wird im EPROM angelegt
        {
            65,75,80,82,85,84,82,78,70,60,60,60,60,60,58,
            45,15,105,170,255,165,80,0,50,60,60,60,60,60,60,
            60,60,60,60,62,70,82,92,96,99,100,99,97,95,91,
            80,66,60,60,65,70,69,65,60,60,60,60,60,60,60,
            60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,
            60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,
            60,60,60,60,60,60,60,60,60,60
        };

    while (1)
        {
            unsigned char x;
            for(x=0; x<=99; x++)                    // Schleife 100mal wiederholen
            {
                ausgang=EKG[x];                     // x als Adresse des ausgegebenen Tabellenwertes
                zeit ();                            // Aufruf der Wartezeit nach jedem Hochzählen
            }

        }
}

void zeit (void)                                    // Einstellung der Wartezeit

{
    unsigned int y;                                 // y als Zählvariable
    for(y=0; y<=1000; y++);                         // Zeit verbrauchen
}
```

/

7. Wichtige Praxis-Anwendung: Ansteuerung von “Intelligenten LCD-Displays”

7.1 Grundlegende Informationen

In der Industrieanwendung der Mikrocontroller nimmt die Anzeige von Messergebnissen, die “Bedienerführung” und die Ausgabe von Kommentaren usw. mit den “Intelligenten LCD-Displays” einen immer größeren Umfang an. Als Standard haben sich für die meisten Anwendungen die ein- bis vierzeiligen Displays mit dem integrierten Hitachi-Controller HD44780 (bzw. entsprechenden Lizenzfertigungen) durchgesetzt.
Sie besitzen insgesamt **80 Speicherplätze** in ihrem RAM, wobei pro Platz ein ASCII-Zeichen (oder ein selbstentworfenes Symbol) abgelegt werden kann. Die verschiedenen Displays unterscheiden sich lediglich in der Zahl der gleichzeitig anzeigbaren Zeichen, sie sind in folgenden Ausführungen erhältlich:

2 x 16 Zeichen / 2 x 20 Zeichen / 2 x 24 Zeichen / 2 x 40 Zeichen / 4 x 16 Zeichen / 4 x 20 Zeichen,

wobei auch Versionen mit Hintergrundbeleuchtung lieferbar sind.

Bedarf an “Leitungen”:
Grundsätzlich wird jedes Zeichen als **8-Bit-Wert (= 1 Byte) vom Mikrocontroller an das Display übergeben**. Da zusätzlich noch **3 Steuerleitungen nötig** sind, haben wir insgesamt drei verschiedene Möglichkeiten zur Kommunikation zwischen Display und Controller:

A) Man verwendet **einen Mikrocontrollerport** für die Ausgabe des Zeichenbytes, muss aber noch **zusätzlich** von einem weiteren Port **drei Leitungen** für die Display-Steuerung belegen.

B) Man arbeitet nur mit “**4-Bit-Daten-Übertragung**” und kommt dadurch **mit 4 + 3 = 7 Leitungen**, also einem **einzigen Port**, aus. Aber nun müssen das obere und untere Nibble **nacheinander übertragen** werden, was etwas mehr Zeit kostet.

C) Man benützt den **Adress- und Datenbus** des Controllers und muss dann mit **Hilfe von zusätzlichen Logikbausteinen erst mal 4 “I/O-Adressen” decodieren**, nämlich:

Schreibe Daten zum Display	(= Übergabe eines Zeichens an die Anzeige)
Lese Daten vom Display	(= Ausgabe von Speicherinhalten der Anzeige an den MC)
Schreibe Code-Bytes zum Display	(dient zur Einstellung oder Umschaltung der Betriebsart)
Lese Code-Bytes vom Display	(damit kann z. B. das BUSY-Flag abgefragt und erkannt werden, ob der Anzeigecontroller schon für die Aufnahme des nächsten Zeichens bereit ist). Diese Methode ist vom Hardwareaufwand her aufwendig, aber natürlich bei der Programmierung sehr übersichtlich.

Wir werden uns deshalb an die Methode B) der 4-Bit-Übertragung halten und nur den Port 5 dafür spendieren!

7.2 Überblick über die Anschlussleitungen der LCD-Displays

Jedes Display weist insgesamt 14 Leitungen auf, die in einer Reihe oder in zwei Reihen angeordnet sind.
Es gilt immer:

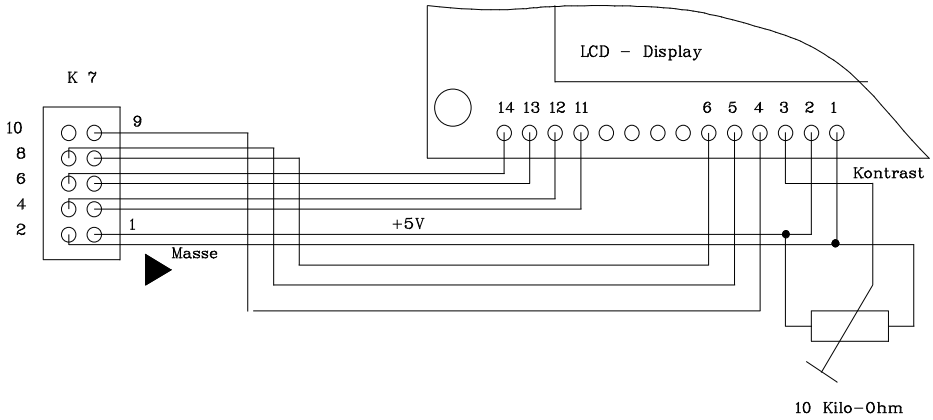
Pin - Nr:	1	2	3	4	5	6	7	8	9	10
Funktion:	Masse	+5V	Kontrast	RS	R/W#	EN	D0	D1	D2	D3

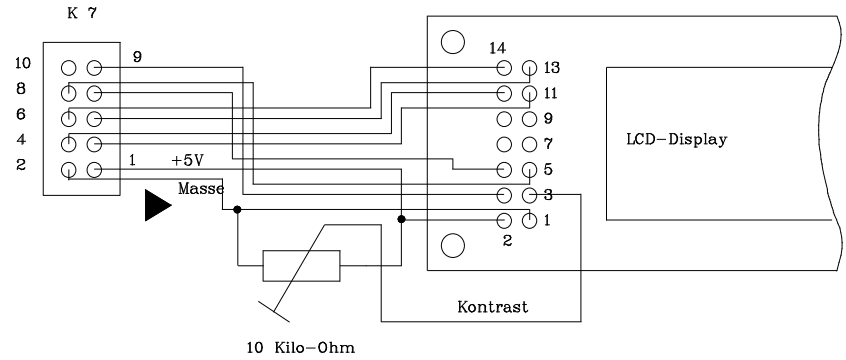
11	12	13	14
D4	D5	D6	D7

Bedeutung: **RS**.....Register Select (log 1: Datenübertragung. log 0: Codebyte-Übertragung)
R/W#.....READ bzw. WRITE NOT (log 1: Lesen. log 0: Schreiben)
EN.....ENABLE (Aktivierung des LCD-Controllers durch Highpegel-Impuls mit der Pulslänge 1 Mikrosekunde. Anschließend sind aber bei verschiedenen Fällen Wartezeiten bis zu 5 Millisekunden nötig).

D0....D7 sind die Datenleitungen, wobei die 4-Bit-Übertragung nur die Anschlüsse 11....14 verwendet! (Anschluss 7....11 werden unbeschaltet gelassen).

Verbindung zwischen Controller und den verschiedenen Displaytypen:





Hinter dieser “Verdrahtungsvorschrift” steckt folgende Zuordnung:

Portpin		
	P5.6 →	Register Select
	P5.5 →	Read/Write#
	P5.4 →	Enable
	P5.3 →	DBit7
	P5.2 →	DBit6
	P5.1 →	DBit5
	P5.0 →	Dbit4

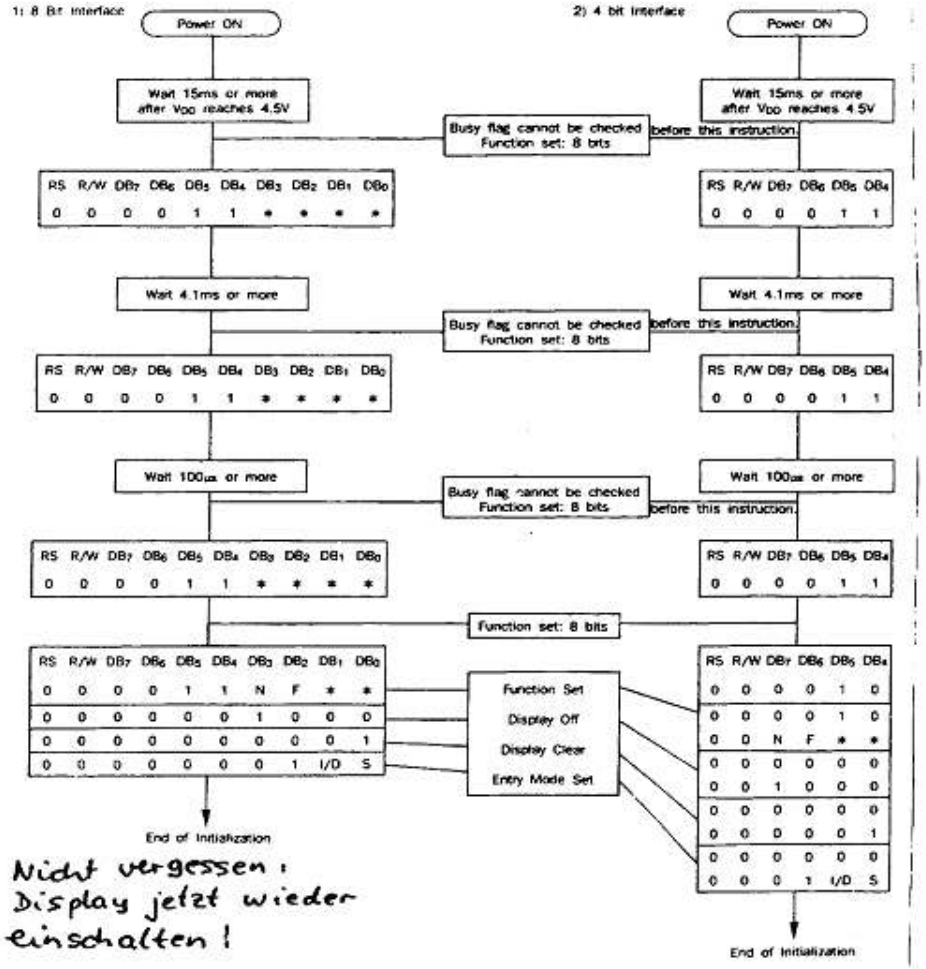
Der letzte Portpin P5.7 bleibt unbenutzt, mit ihm kann man z. B. einen Lautsprecher über einen Trennverstärker ansteuern.

7.3 Einschaltroutine und Function Set

Jedes dieser LCD-Displays muss beim **Einschalten** eine **streng festgelegte Prozedur** durchlaufen, da es sonst entweder **nicht richtig oder überhaupt nicht funktioniert**. Man lässt üblicherweise das Mikrocontrollerprogramm mit dieser Prozedur beginnen, damit das Display später zu jeder beliebigen Zeit benützt werden kann.

Auf dem nächsten Blatt ist diese “Einschaltvorschrift” sowie eine Übersicht über die verschiedenen Steuersignale aus dem Datenblatt dieses LCD-Controllers (HD 44780) abgedruckt.

Function Set



Nicht vergessen: Display jetzt wieder einschalten!

Nicht vergessen: Display jetzt wieder einschalten!

Code – Tabelle

Instruction	Code										Description	Execution Time (max) (when fcp or fosc is 250 KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM anddress 0 in address counter.	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	*	Sets DD RAM address 0 in address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 µs
Display On / Off Control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of entire display (D). Cursor ON/OFF (C), and blink of cursor position character (B).	40 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor & shifts display without changing DD RAM contents.	40 µs
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (L) and character fonts (F).	40 µs
Set CG RAM Address	0	0	0	1	ACG						Sets CG RAM address. CG RAM data is sent and received after this setting.	40 µs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. CG RAM data is sent and received after this setting.	40 µs
Read Busy Flag and Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM	40 µs
Read Data from CG or DD RAM	1	1	Read Data								Reads data into DD RAM or CG RAM	40 µs
	I/D = 1 : Increment I/D = 0 : Decrement S = 1 : Accompanies display shift S/C = 1 : Display shift S/C = 0 : Cursor move R/L = 1 : Shift to the right R/L = 0 : Shift to the left DL = 1 : 8 bits, DL = 0 : 4 bits N = 1 : 2 lines, N = 0 : 1 line F = 1 : 5 × 10 dots, F = 0 : 5 × 7 dots BF = 1 : Internally operating BF = 0 : Can accept instruction										DD RAM : Display data RAM CG RAM : Character generator RAM ACG: CG RAM address ADD: DD RAM Address : Corresponds to cursor address AC: Address counter used for both DD and CG RAM address.	Execution time changes when frequency changes Example: When fcp or fosc is 270 kHz: $40\ \mu s \times \frac{250}{270} = 37\ \mu s$	

* No effect

7.4. Ein Modul für alle Fälle

Wir wollen zuerst auf der Festplatte einen Projektordner „LCD“ anlegen. In ihm wird der nachfolgende Modul „**LCD_CTRL.a51“** abgelegt. Er enthält alle erforderlichen Unterprogramme, die wir zur Programmierung des Displays jemals brauchen werden:

Die **Einschaltroutine** wird durch den Programm-Modul “**LCDEIN**” verwirklicht, während die “**Betriebsarten-Einstellung**” des **Displays** durch eine darauffolgende **“Initialisierungsroutine LCDINI1**” erfolgt.
Außerdem gibt es das Steuerprogramm „**ANZZEICH**“ zur Darstellung **eines einzigen ASCII-Zeichens** und das Steuerprogramm „**ANZTEXT“ für fortlaufenden Text.**
Für die Übertragung vom Microcontroller zur Display dient ein kurzes “**LCDCOM-Programm**”, dazu werden noch zwei verschieden lange Wartezeiten (**ZEITL** ergibt 5 Millisekunden, **ZEITS** ergibt 100 Mikrosekunden) benötigt. Zur Ansteuerung der verschiedenen Display-Varianten stehen noch die **Umschaltroutinen Z1 ....Z4** für einzeilige bis vierzeilige Darstellung zur Verfügung.

Achtung:
a) Alle diese Unterprogramme werden mit der Direktive „PUBLIC“ für die Verwendung in anderen Modulen zugänglich gemacht.
b) **In diesem Modul dürfen keine Adressanweisungen (CSEG.. oder ORG) für den Linker enthalten sein, damit die Unterprogramme an beliebigen Stellen in andere Module eingebunden werden können!**

```
;lcd_ctrl.a51
;
;Technische Erläuterungen:
;          a) Das Display wird über ein 10poliges Flachbandkabel an
;               Port 5 (Stecker K7) angeschlossen. Davon werden 9
;               Leitungen für die Übertragung zum Display benötigt.
;               Pin 1 des Steckers K7 führt die Betriebsspannung von
;               +5V, Pin 2 dagegen Massepotential.
;               Der letzte Stift Nr. 10 von Stecker K7 ist nicht belegt
;               und kann z. B. mit ein Trennverstärker zur
;               Ansteuerung des Lautsprechers verbunden werden.
;
;             b) Deshalb wird nur mit "4 Bit - Übertragung" vom
;               Controller zum Display gearbeitet.
;
;             c) Dabei dienen die 4 Portpins P5.0 .....P5.3 (das sind
;               die Stifte 3,4,5,6 des Steckers K7) als
;               Datenleitungen, sie werden mit den Anschlüssen D4.....D7
;               des LCD-Panels verbunden (das sind die Anschlusspins
;               11....14 des LCD-Displays).
;
;             d) Für die Steuersignale gilt:
;               Portpin P5.4 (das ist Stift 7 des Steckers K7)
;               steuert ENABLE und wird deshalb mit Pin 6 des
;               LCD-Displays verbunden)
;
;               Portpin P5.5 (das ist Stift 8 des Steckers K7) steuert
;               READ / WRITE und wird deshalb mit Pin 5 des
;               LCD-Displays verbunden.
;
;               Portpin P5.6 (das ist Stift 9 des Steckers K7) steuert
;               REGISTER SELECT "RS" und wird deshalb mit Pin 4 des
;               LCD-Displays verbunden.
;
;             e) Die Übertragung eines alphanumerischen Zeichens zum
;               Display dauert ca. 100 Mikrosekunden!

;-------------------------------------------------------------------------------
$NOMOD51                        ; 8051-Standard-Modus abschalten
$INCLUDE (REG515.INC)           ; 80535-Registersatz bereitstellen
;===============================================================================
;Assembler-Direktiven und Deklarationen
;===============================================================================
PUBLIC lcdein
PUBLIC lcdini1
PUBLIC anzzeich
PUBLIC anztext
PUBLIC lcdcom
```

```
PUBLIC zeitl
PUBLIC zeits
PUBLIC z1
PUBLIC z2
PUBLIC z3
PUBLIC z4


ENABLE      BIT P5.4                ; Deklaration von "Enable"

?STACK      SEGMENT IDATA           ; Stack soll im IDATA-Bereich liegen.
            RSEG ?STACK             ; Schalte auf ?Stack-Segment um.
            DS 5                    ; Reserviere 5 Bytes für den Stack.
;===============================================================================================
;Erst hier beginnt unser Programm (= assembler instructions)
;===============================================================================================
EINSCHALTEN SEGMENT   CODE          ;Erstes Codesegment heißt "EINSCHALTEN"
            RSEG EINSCHALTEN        ;Schalte auf dieses Codesegment um
            USING 3                 ;Registerbank 3 soll verwendet werden

lcdein:     call zeitl              ;15 Millisekunden warten nach dem Start
            call zeitl
            call zeitl
            mov P5,#0000011B        ;function - set = 3 Durchgänge mit
            setb ENABLE             ;demselben Codewort
            nop                     ;ENABLE auf HIGH und nach 1 Mikrosekunde
            clr ENABLE              ;wieder auf LOW zurück
            call zeitl              ;5 Millisekunden warten
            setb ENABLE             ;
            nop
            clr ENABLE
            call zeitl              ;2. Durchgang: 5 Millisekunden warten
            setb ENABLE             ;
            nop
            clr ENABLE
            call zeitl              ;3. Durchgang: 5 Millisekunden warten
            ret


lcdini1:    mov P5,#0000010B        ;"function set" auf 4 Bit - Darstellung
                                    ;umschalten
            call lcdcom             ;Code zum Display schicken
            mov P5,#0000010B        ;Oberes Nibble für
                                    ;"4Bit / 2 Zeilen / 5x7 dots" losschicken
            call lcdcom
            mov P5,#0001000B        ;Unteres Nibble für
                                    ;"4Bit / 2 Zeilen / 5x7 dots" losschicken
            call lcdcom
            mov P5,#0000000B        ;Oberes Nibble von "Display OFF"
            call lcdcom
            mov P5,#0001000B        ;Unteres Nibble von "Display OFF"
            call lcdcom
            mov P5,#0000000B        ;Oberes Nibble von "Clear all"
            call lcdcom
            mov P5,#0000001B        ;Unteres Nibble von "Clear all"
            call lcdcom
            call zeitl              ;Beim Löschen 5 Millisekunden warten
            mov P5,#0000000B        ;Oberes Nibble von "Entry mode""
            call lcdcom
            mov P5,#0000110B        ;Unteres Nibble von "Entry mode (Cursor
                                    ;wandert nach rechts, Display steht still)"
            call lcdcom
            mov P5,#0000000B        ;Oberes Nibble von "Display ON, Cursor OFF,
                                    ;Kein Blinken"
            call lcdcom
            mov P5,#0001100B        ;Unteres Nibble von "Display ON"
            call lcdcom
            ret
;-------------------------------------------------------------------------------

ANZEIGE     SEGMENT CODE            ;Dieses CODE-Segment heißt "Anzeige"
```

```
                RSEG ANZEIGE        ;Schalte auf dieses Codesegment um
                USING 3             ;Registerbank 3 soll verwendet werden

anzzeich:       push psw            ;psw retten
                orl psw,#00011000b  ;auf Registerbank 3 umschalten
                mov r7,a            ;Akku-Inhalt nach R7
                mov a,r7            ;Zeichen in den Akku holen
                swap a              ;oberes und unteres Nibble vertauschen
                anl a,#0FH          ;nur oberes Nibble nötig
                orl a,#01000000B    ;Zusätze für Übertragung zum Display
                mov P5,A            ;Akkuinhalt an Port 5 übergeben
                call lcdcom         ;Übertragungsroutine aufrufen
                mov a,r7            ;nochmals Anzeigewert holen
                anl a,#0FH          ;jetzt nur unteres Nibble nötig
                orl a,#01000000B    ;Zusätze für Übertragung
                mov P5,a            ;Akkuinhalt an Port 5 übergeben
                call lcdcom         ;Übertragungsroutine aufrufen
                pop psw             ;psw zurückladen
                ret

anztext:        push psw            ;psw retten
                orl psw,#00011000b  ;auf Registerbank 3 umschalten
                mov r7,a
anz1:           clr a
                MOVC A,@A+DPTR      ;Tabellenwert entsprechend Offset holen
                SWAP A              ;oberes und unteres Nibble vertauschen
                ANL A,#0FH          ;nur oberes Nibble nötig
                ORL A,#01000000B    ;Zusätze für Übertragung zum Display
                MOV P5,A            ;Akkuinhalt an Port 5 übergeben
                CALL LCDCOM         ;Übertragungsroutine aufrufen
                clr a
                MOVC A,@A+DPTR      ;nochmals Codebyte aus Tabelle holen
                ANL A,#0FH          ;jetzt nur unteres Nibble nötig
                ORL A,#01000000B    ;Zusätze für Übertragung
                MOV P5,A            ;Akkuinhalt an Port 5 übergeben
                CALL LCDCOM         ;Übertragungsroutine aufrufen
                INC DPTR            ;Zeichenadresse um 1 erhöhen
                DJNZ R7,ANZ1        ;Zeichenzähler um 1 erniedrigen
                pop psw
                RET                 ;

LCDCOM:         SETB ENABLE         ;ENABLE auf ON
                NOP                 ;1 Mikrosekunde warten
                NOP
                CLR ENABLE          ;ENABLE wieder OFF
                CALL ZEITS          ;100 Mikrosekunden warten
                RET
;
ZEITL:          push psw            ;psw retten
                orl psw,#00011000b  ;auf Registerbank 3 umschalten
                MOV R2,#10          ;Zeitschleife mit 10x500 Mikrosekunden vorsehen
ZEITL1:         MOV R1,#250         ;250 Durchgänge zu je 2 Mikrosekunden ergeben 500 Mikrosekunden
ZEITL2:         DJNZ R1,ZEITL2      ;dieser Befehl dauert 2 Mikrosekunden
                DJNZ R2,ZEITL1      ;
                pop psw
                RET                 ;
;
ZEITS:          push psw            ;psw retten
                orl psw,#00011000b  ;auf Registerbank 3 umschalten
                MOV R1,#50          ;50 Durchläufe programmieren
ZEITS1:         DJNZ R1,ZEITS1      ;sie ergeben 2 x 50 = 100 Mikrosekunden
                pop psw
                RET

Z1:             MOV P5,#0001000B    ;Umschaltung auf Zeile 1
                CALL LCDCOM         ;(1. Zeichen am linken Rand)
                MOV P5,#0000000B    ;
                CALL LCDCOM         ;
                RET
;
;
```

```
Z3:     MOV P5,#0001001B    ;Umschaltung auf Zeile 3 (1. Zeichen
        CALL LCDCOM         ;am linken Rand)
        MOV P5,#0000100B    ;
        CALL LCDCOM         ;
        RET                 ;

Z2:     MOV P5,#0001100B    ;Umschaltung auf Zeile 2
        CALL LCDCOM         ;(1. Zeichen am linken Rand)
        MOV P5,#0000000B    ;
        CALL LCDCOM         ;
        RET

Z4:     MOV P5,#0001101B    ;Umschaltung auf Zeile 4 (1. Zeichen
        CALL LCDCOM         ;am linken Rand)
        MOV P5,#0000100B    ;
        CALL LCDCOM         ;
        RET                 ;

;-------------------------------------------------------------------------
;Programm - Ende
;-------------------------------------------------------------------------
end
```

Hinweise: a) Die Codes für alle darstellbaren Zeichen finden sich auf dem nächsten Blatt. Sie sind zum größten Teil identisch mit dem "AMERICAN STANDARD CODE For INFORMATION INTERCHANGE (= amerikanischer Standardcode für Informationsaustausch = ASCII). Das bedeutet, dass **leider die deutschen Umlaute nicht einfach mit dem sonst üblichen Kommando**

DB '.....' **(bedeutet: "define** Byte")

im Assemblerprogramm aufgerufen werden können. Man muss dazu das Datenwort aus der **Tabelle** des nächsten Blattes fischen und bewusst zum Display senden.

b) Um irgendwelche **Messwerte** anzeigen zu können, muss man natürlich erst mal ein Unterprogramm besorgen, das die vom Mikrocontroller verwendeten **Hexadezimalzahlen in BCD - Zahlen umwandelt**. Der Rest ist dann einfach:

vor jede Ziffer wird eine "3" gesetzt und damit ist das jeweilige, zu sendende Datenbyte schon fertig.

Higher 4 bit / Lower 4 bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
××××0000	CG RAM (1)												
××××0001	(2)												
××××0010	(3)												
××××0011	(4)												
××××0100	(5)												
××××0101	(6)												
××××0110	(7)												
××××0111	(8)												
××××1000	(1)												
××××1001	(2)												
××××1010	(3)												
××××1011	(4)												
××××1100	(5)												
××××1101	(6)												
××××1110	(7)												
××××1111	(8)												

7.5. Anzeige eines einzigen Zeichens auf dem Display

Aufgabe: Schreiben Sie einen Programm-Modul, mit dem Sie ein einzelnes Zeichen aus dem Zeichenvorrat des Displays anzeigen können. Binden Sie dabei alle erforderlichen Unterprogramme mit der Direktive

EXTRN CODE(lcdein, lcdini1, anzzeich, lcdcom, zeitl, zeits,)

ein. Das anzuzeigende Zeichen muss dabei im Akku stehen. Anschließend soll bei diesem Beispielprogramm eine Endlosschleife aufgerufen werden. Das Programm beginnt bei der EPROM-Adresse 4000h. Organisieren Sie dann Ihr Projekt so, **dass Ihr neuer Modul gemeinsam mit dem bereits vorhandenen Modul „lcd_ctrl.a51“ verwendet** und vom Linker verknüpft wird. Testen Sie anschließend die korrekte Funktion mit der Hardware.

Lösungsbeispiel für die Anzeige eines Zeichens:

```
;anz1.a51

$NOMOD51                        ; 8051-Standard-Modus abschalten
$INCLUDE (REG515.INC)           ; 80535-Registersatz bereitstellen

EXTRN CODE (lcdein, lcdini1, anzzeich, lcdcom, zeitl, zeits)
OFFSET EQU 4000h

?STACK  SEGMENT IDATA           ; Stack soll im IDATA-Bereich liegen.
        RSEG ?STACK             ; Schalte auf ?Stack-Segment um.
        DS 5                    ; Reserviere 5 Bytes für den Stack.

CSEG AT 0 + OFFSET
ljmp start

HAUPTPR1  SEGMENT CODE
          RSEG HAUPTPR1

start:    mov SP,#?STACK-1      ;Stack-Adresse zuweisen
          call lcdein               ;Einschaltroutine ausführen
          call lcdini1              ;Display initialisieren
          mov a,#033h               ;ASCII-Zeichen in den Akku laden
          call anzzeich             ;Anzeigeroutine aufrufen
          jmp $                     ;Endlosschleife für Demoprogramm
END
```

7.6. Anzeige von zweizeiligem Text auf dem Display

Aufgabe: Schreiben Sie ein Programm, mit dem Sie sowohl in Zeile 1 wie auch in Zeile 2 einen Text ausgeben können.

Dabei müssen Sie bei jeder Zeile folgende Prozedur durchführen:

a) In das Register R7 wird die Anzahl der anzuzeigenden Zeichen geschrieben.

b) Der Datapointer wird mit der Startadresse (# Label) der Textzeile geladen.

c) Dann wird auf die gewünschte Zeile umgeschaltet und „anztext“ aufgerufen.

Der Text selbst wird z. B. folgendermaßen programmiert:

```
text1: db 'LCD_Anzeige'                          ;das ergibt ASCII-Text

text2: db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h   ;damit werden Einzelzeichen
                                                    ;dargestellt. Hier sehen wir die
                                                    ;Zahlen von 0....9.
```

Die Startadresse ist 4000h, es ist wieder eine Endlosschleife vorzusehen. Verwenden Sie die Unterprogramme aus „lcd_ctrl“ über die EXTRN-Direktive.

Achtung:
Vergessen Sie bei der Organisation des Projektes (und vor dem Assemblieren) aber nicht, das Programm der vorigen Aufgabe mit „remove“ aus der Projektliste zu entfernen....

Lösungsbeispiel für zweizeilige Anzeige:

```
$NOMOD51                                ; 8051-Standard-Modus abschalten
$INCLUDE (REG515.INC)                   ; 80535-Registersatz bereitstellen

EXTRN CODE (lcdein, lcdini1, anzzeich, lcdcom, zeitl, zeits, z1, z2, anztext)

OFFSET EQU 4000h
?STACK      SEGMENT IDATA               ; Stack soll im IDATA-Bereich liegen.
            RSEG ?STACK                 ; Schalte auf ?Stack-Segment um.
            DS 5                        ; Reserviere 5 Bytes für den Stack.

CSEG AT 0+OFFSET
jmp START
;=============================================================================================
;Erst hier beginnt unser Programm (= assembler instructions)
;=============================================================================================
HAUPTPR1    SEGMENT CODE
            RSEG HAUPTPR1


START:      mov SP,#?STACK-1            ;Stackadresse zuweisen

            call lcdein                 ;Einschaltroutine
            call lcdini1                ;Function set
            mov a,#16                   ;Zeichenzähler auf 16 setzen
            mov dptr,#text1z            ;Startadresse des Textes in DPTR
            call z1                     ;auf 1. Zeile umschalten
            call anztext                ;Text zum Display senden

            mov a,#16
            mov dptr,#text2z
            call z2                     ;auf 2. Zeile umschalten
            call anztext                ;Text anzeigen
            jmp $                       ;Endlos-Schleife wegen Demoprogramm


text1z: db 'LCD-Anzeige     '
text2z: db  'funktioniert   '

END
```

Aufgabe: Ändern Sie Ihr Programm so um, dass sich eine „blinkende Anzeige“ auf dem Display ergibt.

(Lösungsweg: Zeigen Sie abwechselnd den gewünschten Text und einen leeren Bildschirm an. Sorgen Sie dafür, dass sich beide Anzeigen im Rhythmus von 0,5....1 Sekunde abwechseln).

7.7. LCD-Display-Programierung in C

Dazu wurde das vorige Assembler-Steuerprogramm „lcd_ctrl.a51“ (mit 4 Bit-Datenübertragung) in C umgeschrieben. Es steht nun als „LCD_CTRL.c“ zur Verfügung.

Es bleibt weiterhin ein selbständiger Modul, der vom Linker an das von Ihnen zu schreibende Anwendungsprogramm angehängt wird und deshalb in der Projektliste von „µvision1“ ausgeführt werden muss.

Deshalb darf in „LCD_CTRL.c“ auch keine „main“-Funktion hineingeschrieben werden, sobald es in dieser Form eingesetzt wird.

„LCD_CTRL.c“ arbeitet mit Pointer - Adressierung und Parameter-Übergabe. Das bedeutet für Sie, dass Sie Ihren anzuzeigenden Text als „Charakter-Zeichenkette“ (= string) anlegen und der Anzeigefunktion mit dem dafür gewählten Namen übergeben müssen.

Bitte achten Sie jedoch hierbei unbedingt auf folgende Spielregeln:

a) Sie müssen alle von „LCD_CTRL.c“ benützten Unterprogramme **zusätzlich** zu Beginn Ihres Hauptprogramms als Prototypen deklarieren.

b) Allgemeinen Text (der sich im Betrieb nicht ändern wird -- z: B. Begrüßungen oder Warnmeldungen oder Menuführungen --) packen Sie bitte unbedingt in das **EPROM**. Das geschieht z. B. mit folgender Anweisung:

```
code unsigned char warnung[21] = {“Sofort ausschalten! “};
```

Dieses Feld wird noch ***vor Aufruf von „main“ global*** deklariert.

Dann könnten Sie durch folgenden Aufruf (entweder in „main“ oder auch in einer weiteren verwendeten Funktion) diese Warnung in Zeile 1 anzeigen lassen:

```
switch_z1();                    // Display auf Zeile 1 umschalten

show_text(warnung);             // Warntext anzeigen lassen
```

c) Wollen Sie dagegen Messwerte speichern und ausgeben, so muss der String ***innerhalb*** derjenigen Funktion stehen, von der aus die Anzeige aufgerufen wird. Das kann „main“ sein, aber auch z. B. eine Interrupt-Service-Routine. Das hat folgenden Grund: solche Werte stehen normalerweise im internen RAM 1 des Controllers -- also im „DATA-Bereich“ -- und nur auf diese Weise wird dort die Adressierung korrekt durchgeführt. Also darf hier das Wort „code“ nicht verwendet werden.
In dieses Array schreiben Sie dann die aktuellen Messwerte und übergeben wie vorhin den Array-Namen an die Funktion „show_text()“.

Beispiel:

```
void main (void)
{
        unsigned char Spannung[21] = {“Spannung = 5,32 V“};
        lcd_ein();
        lcd_ini1();
        switch_z1();
        show_text(Spannung);
        .....
}
```

Nun folgt ein Demoprogramm „LCD_DEMO“ zur Demonstration der Anwendung, gefolgt von „LCD_CTRL_new.c“

```
/*---------------------------------------------------------------------
```
LCD_DEMO
02.01.2005 / G.Kraus

Angezeigt wird in der ersten Zeile ein Text, der im EPROM gespeichert ist. In der zweiten Zeile wird ein im Controller-RAM angelegter Charakter-String ausgegeben, gefolgt von einem einzigen Zeichen (beispielsweise ein Messwert als Variable y).

```
---------------------------------------------------------------------*/
#include <reg515.h>
#include <stdio.h>

void zeit_s(void);
void zeit_l(void);
void lcd_com(void);
void lcd_ein(void);
void lcd_ini1(void);
void switch_z1(void);
void switch_z2(void);
void show_text(char *ptr);
void show_char(char *zeichen);


code unsigned char zeile_1[21]={"LCD-Display in C    "};

void main(void)
{               char zeile_2[21]={"Messwert: "};
                unsigned char y='5';
                lcd_ein();                      // Einschaltroutine
                lcd_ini1();                     // Function Set
                switch_z1();                    // Schalte auf Zeile 1
                show_text(zeile_1);             // Text in Zeile 1 zeigen
                switch_z2();                    // Schalte auf Zeile 2
                show_text(zeile_2);             // Text in Zeile 2 zeigen
                show_char(&y);                  // y zusätzlich anzeigen

                while(1);                       // Endlosschleife
}

//-----------------------------------------------------------------------------------------------------------------------------------------
```

```
/*-------------------------------------------------------------
LCD_CTRL_new.c
C-Programm-Modul  zur Ausgabe von Texten auf dem LCD-Display
02.03.2005 / G. Kraus
-------------------------------------------------------------*/
#include <reg515.h>
#include <stdio.h>

void zeit_s(void);
void zeit_l(void);
void lcd_com(void);
void lcd_ein(void);
void lcd_ini1(void);
void switch_z1(void);
void switch_z2(void);
void switch_z3(void);
void switch_z4(void);
void show_text(char *ptr);
void show_char(char *zeichen);

sfr ausg_lcd=0xf8;                      // Port 5 als Display-Ausgang

sbit enable=ausg_lcd^4;                 // Portpin P5.4 ist "enable"
sbit RW_LCD=ausg_lcd^5;                 // Portpin P5.5 ist "READ - WRITE"
sbit RS_LCD=ausg_lcd^6;                 // Portpin P5.6 ist "Register - Select"

/*-------------------------------------------------------------------------
Zusatzfunktionen
-------------------------------------------------------------------------*/

void zeit_s(void)                       // Funktion „kurze Wartezeit“ gibt etwa 100 Mikrosekunden
{
        unsigned char x;
        for(x=0;x<=30;x++);
}


void zeit_l(void)                       // Funktion „lange Wartezeit“ ergibt 4ms
{       unsigned int x;
        for(x=0;x<=1000;x++);
}


void lcd_ein(void)                      // Einschaltroutine für Display
{       unsigned char x,y;
        for(x=0;x<=2;x++)               // mindestens 3 x 4ms = 12 ms warten
        {       zeit_l();
        }

        ausg_lcd=0x03;                  // Dreimal nacheinander dasselbe Codewort
        for(y=0;y<=2;y++)               // “0011B”  =  0x03 mit Wartepausen dazwischen zum Display schicken
        {       enable=1;               // Damit wird der “Function set” verwirklicht
                for(x=0;x<2;x++);
                enable=0;
                zeit_l();
        }
}


void lcd_ini1(void)                     // Initialisierung des Displays
{       ausg_lcd=0x02;                  // „0010B“  = 0x02 ergibt 4-Bit-Darstellung beim Display

        lcd_com();                      // Code senden

        ausg_lcd=0x02;                  // Oberes Nibble für „4 Bit / 2 Zeilen / 5x7 dots“

        lcd_com();                      // losschicken

        ausg_lcd=0x08;                  // Unteres Nibble für „4 Bit / 2 Zeilen / 5x7 dots“

        lcd_com();                      // losschicken
```

```
        ausg_lcd=0x00;              // Oberes Nibble von „Display OFF“

        lcd_com();                  // losschicken

        ausg_lcd=0x08;              // Unteres Nibble von „Display OFF“

        lcd_com();                  // losschicken

        ausg_lcd=0x00;              // Oberes Nibble von „Clear all“

        lcd_com();                  // losschicken

        ausg_lcd=0x01;              // Unteres Nibble von „Clear all“

        lcd_com();                  // losschicken

        zeit_l();                   // Beim Löschen ist hier eine lange Wartezeit nötig

        ausg_lcd=0x00;              /* Oberes Nibble von „Entry mode“ (Cursor steht, Display wandert
                                       nach rechts */

        lcd_com();                  // losschicken

        ausg_lcd=0x06;              /* Unteres Nibble von „Entry mode“ (Cursor steht, Display wandert
                                       nach rechts */

        lcd_com();                  // losschicken

        ausg_lcd=0x00;              // Oberes Nibble von „Display ON, Cursor OFF, Kein Blinken“

        lcd_com();                  // losschicken

        ausg_lcd=0x0c;              // Unteres Nibble von „Display ON, Cursor OFF, Kein Blinken“

        lcd_com();                  // losschicken
}


void lcd_com(void)                  // Übertragungs-Routine für Daten zum Display
{       unsigned char x;
        enable=1;                   // ENABLE für einige Mikrosekunden auf HIGH
        for(x=0;x<2;x++);
        enable=0;                   // ENABLE wieder zurück auf LOW
        zeit_s();                   // Kurze Wartezeit aufrufen
}


void switch_z1(void)                // Schalte auf Zeile 1 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x08;              // Erforderliches Codewort hat als oberes Nibble „1000B“ = 0x08
        lcd_com();                  // losschicken
        ausg_lcd=0x00;              // Erforderliches Codewort hat als unteres Nibble „0000B = 0x00
        lcd_com();                  // losschicken
}


void switch_z2(void)                // Schalte auf Zeile 2 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x0c;              // Erforderliches Codewort hat als oberes Nibble „1100B“ = 0x0C
        lcd_com();                  // losschicken
        ausg_lcd=0x00;              // Erforderliches Codewort hat als oberes Nibble „0000“ = 0x00
        lcd_com();                  // losschicken
}

void switch_z3(void)                // Schalte auf Zeile 3 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x09;              // Erforderliches Codewort hat als oberes Nibble „1001B“ = 0x09
        lcd_com();                  // losschicken
        ausg_lcd=0x04;              // Erforderliches Codewort hat als unteres Nibble „0100B = 0x04
        lcd_com();                  // losschicken
}
```

```
void switch_z4(void)                    // Schalte auf Zeile 4 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x0D;                  // Erforderliches Codewort hat als oberes Nibble „1101B“ = 0x0D
        lcd_com();                      // losschicken
        ausg_lcd=0x04;                  // Erforderliches Codewort hat als oberes Nibble „0100“ = 0x04
        lcd_com();                      // losschicken
}


void show_text(char *ptr)               // Anzeige eines Textes, als Array vorgegeben
{
        while(*ptr)                     /* Startadresse des Arrays wird übergeben. Schleife wird solange
                                           wiederholt, bis Code „0x00“ (entspricht dem Zeichen „\0“) gefunden
                                           wird.
        {
                ausg_lcd=(*ptr/0x10)|0x40;      /*  Oberes Nibble des Zeichen-Bytes wird um 4 Stellen nach rechts
                                                    geschoben und anschließend um die erforderlichen Steuersignale
                                                    ergänzt. */

                lcd_com();                      //  Oberes Nibble losschicken

                ausg_lcd=(*ptr&0x0f)|0x40;      /*  Unteres Nibble des Zeichen-Bytes wird durch Löschen des
                                                    oberen Nibbles gewonnen und anschließend um die erforderlichen
                                                    Steuersignale ergänzt. */

                lcd_com();                      //  Unteres Nibble losschicken

                ptr++;                          //  Nächstes Zeichen des Arrays adressieren
        }
}


void show_char(char *ptr)               //  Anzeige eines einzigen ASCII-Zeichens. Adresse wird übergeben.
{
        ausg_lcd=(*ptr/0x10)|0x40;      /*  Oberes Nibble des Zeichen-Bytes wird um 4 Stellen nach rechts
                                            geschoben und anschließend um die erforderlichen Steuersignale
                                            (für Datenübertragung) ergänzt. */

        lcd_com();                      //  Oberes Nibble losschicken

        ausg_lcd=(*ptr&0x0f)|0x40;      /*  Unteres Nibble des Zeichen-Bytes wird durch Löschen des
                                            oberen Nibbles gewonnen und anschließend um die erforderlichen
                                            Steuersignale (für Datenübertragung) ergänzt. */

        lcd_com();                      //  Unteres Nibble losschicken
}
```

7.8. Testprogramm zur Ausgabe der verfügbaren Zeichen bei einem LCD-Modul

Auf dem Markt gibt es unzählige LCD-Display-Modulversionen, die mit dem Etikett „HD4780-kompatibel“ versehen sind. Diese Eigenschaft bedeutet aber nicht, dass sämtliche Module dieselben Zeichen anzeigen werden, da nur ein bestimmter Teil des Vorrats (Zahlen, Buchstaben…) identisch ist. Mit dem folgenden Programm ist es möglich, die Hex-Codes von 0x00 bis 0xFF zum eigenen Display zu senden. Dann kann dort das zugehörige Zeichen bewundert werden.

```
/*-------------------------------------------------------------------
Programmbeschreibung
Demo-Programm zur Ausgabe der verfügbaren Zeichen bei einem LCD-Display
01.12.2005 / Gunthard Kraus

Angezeigt wird in der ersten Zeile das Wort "Hex: "gefolgt von einer Hex-Zahl. Diese wird von 0x00 bis
0xFF hoch gezählt und zwischen jeden Zählschritt eine Pause gelegt. In der zweiten Zeile steht "gibt:  ", gefolgt vom
ASCII-Zeichen, das bei diesem Display zu der
jeweiligen Hex-Zahl gehört.
-------------------------------------------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------------------------------------------*/
#include <reg515.h>                                          //Header
#include <stdio.h>
/*-------------------------------------------------------------------------------------------------------
Prototypen, Deklarationen, Hauptprogramm, Zusatzfunktionen
-------------------------------------------------------------------------------------------------------*/
void zeit_s(void);                                           // Prototypen für "LCD_CTRL.C" - Funktionen
void zeit_l(void);
void lcd_com(void);
void lcd_ein(void);
void lcd_ini1(void);
void switch_z1(void);
void switch_z2(void);
void show_text(char *ptr);
void show_char(char *zeichen);

code unsigned char hex_ascii[17]={"0123456789ABCDEF"};       // Array mit ASCII-Zeichen von 0x0 bis 0xF

void wait(void);                                             // Prototyp für Warteschleife

void main(void)
{   unsigned char zeile_1[5]={"Hex "};                       // Erste Zeile
    unsigned char zeile_2[7]={"gibt: "};                     // Zweite Zeile
    unsigned char x=0;                                       // Inkrementierte Hex-Zahl

    lcd_ein();                                               // Einschaltroutine für LCD-Display
    lcd_ini1();                                              // Function Set für LCD-Display

    while(1)
    {       unsigned char upper_nibble;                      // Oberes Nibble der Hex-Zahl
            unsigned char lower_nibble;                      // Unteres Nibble der Hex-Zahl

            upper_nibble=hex_ascii[x/16];                    // Berechnung des Oberen Nibbles
            lower_nibble=hex_ascii[x&0x0F];                  // Berechnung des Unteren Nibbles
            switch_z1();                                      // Schalte auf Zeile 1
            show_text(zeile_1);                              // Text in Zeile 1 zeigen
            show_char(&upper_nibble);                        // Zeige ASCII-Version des Oberen Nibbles
            show_char(&lower_nibble);                        // Zeige ASCII-Version des Unteren Nibbles

            switch_z2();                                     // Schalte auf Zeile 2
            show_text(zeile_2);                              // Text in Zeile 2 zeigen
            show_char(&x);                                   // Zeige das zugehörige Zeichen auf dem Display

            x++;                                             // Hex-Zahl inkrementieren
            wait();                                          // warten
    }
}


void wait(void)                                              // Warte-Funktion
{       unsigned int y;
        for(y=0;y<=30000;y++);
}
```