Paul Bauriedl

Maschinenspracheprogramme und Hardware-Erweiterungen für Schneider CPC's

2. Auflage

für 464, 664 und 6128

IDEA

Paul Bauriedl

Maschinenspracheprogramme und Hardware-Erweiterungen für Schneider CPC's

2. Auflage

IDEA

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Bauriedl, Paul:
Maschinenspracheprogramme und Hardware-Erweiterungen
für Schneider CPC's/Paul Bauriedl.
Puchheim: IDEA, 1988 . ISBN 3-88793-147-5

ISBN 3-88793-147-5

Herstellung: Gräbner, Altendorf b. Bamberg
Printed in Germany

Inhaltsverzeichnis

Vorwort

Dieses Buch stellt keine Einführung in die Z-80- Maschinensprache dar, es ist mehr eine notwendige Bausteinsammlung für Ihre Maschinensprache-Bibliothek. Deshalb empfehle ich auch die Lektüre von z.B Rodnay Zaks Standardwerk "Programmierung des Z 80", Sybex-Verlag, wenn Sie sich bei dem einen oder anderen Befehl doch nicht so sicher sind.

Durch die fehlende Einführung zur Maschinensprache können wir sofort an die "große Programmierung" gehen und haben Zeit für Schneider-Spezifisches und komplette Problemlösungen.

Zum ersten Teil: Die Werkzeugkiste.

Hier habe ich wichtige Routinen des Schneider-Roms mit ihren Ein- und Aussprungsbedingungen beschrieben .
Dazu gehört auch eine eingehende Erklärung dessen, wie ihr Computer Strings, Fließkomma- und Integerzahlen verarbeitet, so daß Sie dann auch in Maschinensprache diese Möglichkeiten nützen können.

Zum zweiten Teil: Routinen und Basicerweiterungen.

Fertige nützliche Programme und einige Basicbefehle mehr kann man immer gebrauchen und diese finden Sie hier. Dabei soll auch gezeigt werden, wie die ROM-Routinen eingesetzt werden können.
Auch das sehr umfangreiche Schneider-Basic kann Erweiterungen vertragen wie z.B. CIRCLE, PAUSE, COPY, TEXT, TAUSCH, um nur einige zu nennen.

Im letzten Teil dann: Hardwareerweiterungen.

Wert wurde daraufgelegt, einige zusätzliche Hardware-Komponenten sowie deren Treiberprogramme vorzustellen. Hier finden Sie D/A- und A/D-Wandler, eine RS-232- und 8-Bit-Centronics-Schnittstelle, einen 24-poligen Ein-/Ausgabeport, einen Timer sowie einen EPROM - Programmierer.

Noch ein Wort zu den Schneider-Computern selber:

Mittlerweile gibt es drei Systeme, den CPC-464, CPC-664 und den CPC-6128, diese drei Typen sind aber nicht restlos miteinander kompatibel. Wenn man von der jeweiligen Speichergröße und dem eingebauten Diskettenlaufwerk absieht, ist da noch ein erweitertes BASIC, das ein geändertes ROM benötigt. Für den BASIC-Programmierer ist das nicht weiter tragisch, es gibt ja bestenfalls mal einen SYNTAX-ERROR, aber der Rechner stürzt deshalb nicht ab. Auch für den Maschinensprache-Programmierer gibt es solange keine Probleme, wie er die Sprungtabelle des RAM's benützt, mit einer kleinen Einschränkung allerdings: Die Tabelle der Fließkommaroutinen befindet sich an einer etwas anderen Stelle, und die Tabelle für die Integerverarbeitung fehlt ganz (zu Gunsten der neuen Befehle).

Es ist aber alles nicht so schlimm, in der "Werkzeugkiste" sind die Unterschiede aufgezeigt.

Alle Programme in diesem Buch sind auf dem CPC-464 ohne Änderung lauffähig, bei Verwendung auf den anderen Systemen sind nur die Einsprünge zu den Rechen-Routinen, wie jeweils angegeben, etwas zu verändern.

Ich wünsche Ihnen nun viel Spaß beim Programmieren und Erweitern Ihres Rechners.

Paul Bauriedl

Werkzeugkiste

Dieses Kapitel ist zu Ihrer Unterstützung gedacht. Hier werden einige wichtige Routinen des Schneider - Betriebssystems mit ihren parametertragenden Registern erläutert. Alle folgenden Einsprungadressen sind -wenn nicht anders angegeben- auf den Schneider Computern 464,664 und 6128 gleich.

Zum größten Teil sind diese Routinen ja selbsterklärend, aber wo das nicht zutrifft, gebe ich Hilfestellung. Die Zeile in Klammern gibt die Register an, die die Unterprogramme selbst verwenden. Sie sollten sie also PUSHen, wenn sie für Sie wichtige Inhalte tragen.

****

Fangen wir gleich bei den wichtigsten Aufrufen an, dazu gehören das Warten auf eine Eingabe von der Tastatur, eine bestimmte Taste prüfen und die Ausgabe eines Zeichens auf dem Bildschirm. Diese Unterprogramme habe ich gleich darauffolgend in kleine Maschinenprogramme eingebaut.

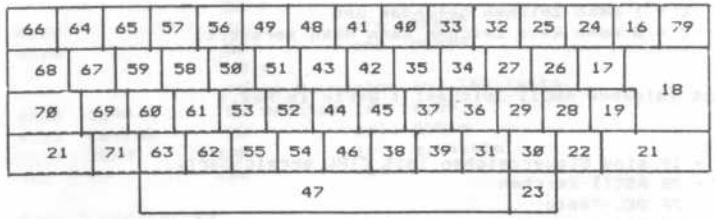
Allgemeine Routinen

* Prüfe auf gedrückte Taste mittels Tastennummer.
(Vergleichbar mit dem Basic-Befehl INKEY.)
(Diese Routine wartet nicht !)

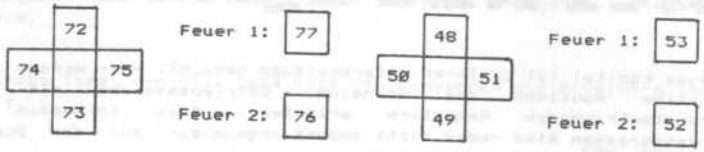
(AF,BC,HL)

```
LD A,Tastennummer
CALL BB1E
Flag's bei Aussprung
Z = Ø, wenn Taste gedrückt wurde und
Z = 1, wenn nicht.
C-Register hat nun folgende Bedeutung:
Bit-7 = 1 wenn mit Controltaste geschehen.
Bit-5 = 1 wenn mit Shifttaste geschehen.
```

Dazu gleich alle möglichen Tastennummern:



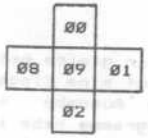
Joystick Ø: ---------- Joystick 1: ----------



Funktionstastenfeld:

10	11	Ø3
20	12	Ø4
13	14	Ø5
15	Ø7	Ø6

Cursortasten:



* Warten auf Taste mit ASCII- oder Erweiterungscode.
(Vergleichbar mit Basic-Befehl INPUT.)
(Diese Routine wartet !)

```
( AF )

CALL BB06
A enthält das Zeichen
```

* Prüfe Taste auf ASCII- oder Erweiterungscode.
(Vergleichbar mit Basic-Befehl INKEY$.)
(Diese Routine wartet nicht !)

```
( AF )

CALL BB09
A enthält das Zeichen
Flag's:
C = 1 wenn Zeichen gefunden und
C = Ø wenn kein Zeichen dann Akku zerstört.
```

Es gibt folgende ASCII-Zeichen: (Werte in Hex.)

```
  Ø - 1F sind Steuerzeichen (mit CTRL erreichbar).
 20 - 7E ASCII-Zeichen
      7F DEL-Taste
      A3 Pfund-Zeichen
```


Ferner folgende Schneider-Spezial-Tasten-Rückgabewerte:

```
 Ø8 - 89 Funktionstasten Ø-9
      8A Funktionstaste Full Stop
      8B      "         ENTER ohne CONTROL
      8C      "         ENTER mit  CONTROL
      E0 COPY-Taste
      E1 TAB -Taste
      F0 Schreib-Cursor rauf
      F1     "     "    runter
      F2     "     "    links
      F3     "     "    rechts
      F4   Copy -Cursor rauf
      F5     "     "    runter
      F6     "     "    links
      F7     "     "    rechts
      F8 Schreib-Cursor zum Textanfang
      F9     "     "     "  Textende
      FA     "     "     "  Zeilenanfang
      FB     "     "     "  Zeilenende
      FC Break
      FD CAPS -Lock-Wert
      FE SHIFT-Lock-Wert
      FF keine Taste
```

* Schreibe ein Zeichen an Bildschirm oder führe Steuercode aus.
(Vergleichbar mit Basic-Befehl PRINT.)

```
( -- )
A enthält das Zeichen (Ø-255)
CALL BB5A
```

* Schreibe ein Zeichen auf Bildschirm. Steuercodes werden ebenfalls nur ausgedruckt nicht ausgeführt !

```
( AF,BC,DE,HL )

(A enth. das Zeichen (Ø-255,wichtig aber nur Ø-31).
CALL BB5D
```

Jetzt folgen einige kleine Beispielprogramme, in denen obige ROM-Routinen verwendet werden:

```
Pass 1 errors: Ø0

                    10 ;Pruefe auf gedrueckte Taste
                    20 ;mittels Tastennummer.
                    30 ;
4E20                40          org  20000
4E20                50          ent  20000
                    60 ;
                    70 ;z.B. warten auf TAB-Taste.
4E20  3E44          80 waitta:  ld   a,68
4E22  CD1EBB        90          call #bb1e
4E25  28F9         100          jr   z,waitta
4E27  C9           110          ret

Pass 2 errors: Ø0
```

```
Pass 1 errors: 00

                      10 ;Warten auf Taste und Echo auf Bildschirm
                      20 ;Abbruch bei ESC-Taste
                      30 ;
4E20                  40         org  20000
4E20                  50         ent  20000
                      60 ;
                      70 ;Steuerzeichen werden ausgedruckt !
4E20  CD06BB          80         call #bb06
4E23  FEFC            90         cp   #fc
4E25  C8             100         ret  z
4E26  CD5DBB         110         call #bb5d
4E29  C9             120         ret
                     130 ;
                     140 ;Steuerzeichen werden ausgefuehrt !
4E2A  CD06BB         150         call #bb06
4E2D  FEFC           160         cp   #fc
4E2F  C8             170         ret  z
4E30  CD5ABB         180         call #bb5a
4E33  C9             190         ret

Pass 2 errors: 00
```

```
Pass 1 errors: 00

                      10 ;Warten auf Taste und Echo auf Bildschirm
                      15 ;wenn Zeichen >=32 dez. war.
                      16 ;Abbruch bei ESC-Taste
                      17 ;
4E20                  20         org  20000
4E20                  30         ent  20000
                      35 ;
4E20  CD09BB          40 loop:   call #bb09
4E23  30FB            50         jr   nc,loop
4E25  FEFC            60         cp   #fc
4E27  C8              70         ret  z
4E28  FE20            80         cp   " "
4E2A  38F4            90         jr   c,loop
4E2C  CD5ABB         100         call #bb5a
4E2F  18EF           110         jr   loop

Pass 2 errors: 00
```

* Zeichenausgabe als Grafikzeichen oder als Steuercodes.

```
        ( AF )

      LD A,FF = Grafikausgabe ein
oder  LD A,00 = Grafikausgabe aus
      CALL BB63
```


* Lies Zeichen von Bildschirm (Vergleicht Matrixen).

```
        ( AF )

        CALL BB60
        A enthält das Zeichen wenn lesbar sonst
        A = 0.
        Flag's:
        C = 1 wenn lesbares Zeichen gefunden sonst
        C = 0.
```

* Joysticks einlesen.

```
        ( AF,HL )

        CALL BB24
        A und H enthalten das Joystick-0-Byte
        L       enthält   das Joystick-1-Byte
```

Das Byte stetzt sich wie folgt zusammen:

```
        Bit 0     Auf
        Bit 1     Ab
        Bit 2     Rechts
        Bit 3     Links
        Bit 4     Feuer 2
        Bit 5     Feuer 1
        Bit 6     NC (Nicht verwendet)
        Bit 7     logisch-0
```

```
*************************************************************
*                                                           *
*              Alle folgende Routinen beziehen              *
*                    sich auf den aktuellen                 *
*           Stream, d.h. auf das angewählte Fenster.        *
*                                                           *
*************************************************************
```

* Aktuellen Stream wählen (0-7).

```
        ( AF,HL )

        LD A,0-7
        CALL BBB4
        A enthält vorherigen Stream bei Aussprung !
```

* Fenster löschen.

```
        ( AF,BC,DE,HL )

        CALL BB6C
```

* Fenster setzen.

(AF,BC,DE,HL)

```
LD H,Spalte einer Ecke
LD D,Spalte der anderen Ecke
LD L,Reihe einer Ecke
LD E,Reihe der anderen Ecke
CALL BB66
```

* Cursor positionieren.

(AF,HL)

```
LD H,neue Spalte
LD L,neue Reihe
CALL BB75
```

* Hole Cursorposition.

(AF,HL)

```
CALL BB78
Reg. H enthält Spalte
Reg. L enthält Reihe
```

* Anwendercursor ein.

(AF)

```
CALL BB7B
```

* Anwendercursor aus.

(AF)

```
CALL BB7E
```

* Setze Textink auf neue Farbe.
(Inknummer:Ø-1 bei Mode 2,Ø-3 bei Mode 1,Ø-15 bei Mode Ø). Ink vorher definieren, sonst werden die Standardfarben verwendet. Das gleiche gilt auch bei allen Aufrufen mit eingesetzter Ink-Nummer.

(AF,HL)

```
LD A,Inknummer
CALL BB9Ø
```

* Hole aktuelle Inknummer. (Gegenteil der vorigen Routine)

(AF)

```
CALL BB93
A enthält Inknummer des Textes
```


* Besetze Inknummer mit neuer Farbe. (Voraussetzung der vorigen vier Unterprogramme).

(AF,BC,DE,HL)

```
LD A,Inknummer
LD B,Farbe 1 ( Ø-26 ) siehe Handb.Anhang IV/S.6
LD C,Farbe 2 ( Ø-26 )   "     "     "     "
CALL BC32
```

* Hintergrund eines Zeichens durchsichtig ein/aus.

(AF,HL)

```
LD A mit Ø, dann undurchsichtig
LD A <> Ø, dann durchsichtig
CALL BB9F
```

* Hole die Adresse der Zeichenmatrix.

(AF,HL)

```
LD A mit Zeichen (Ø-255).
CALL BBA5
HL enthält die Adresse des Zeichens.
Flag's bei Aussprung:
C=1 wenn Matrix im RAM und
C=Ø wenn Matrix im ROM gefunden wurde.
```

* Text invers darstellen bis zum erneuten Aufruf.
(Hintergrund mit Vordergrund tauschen.)

(AF,HL)

```
CALL BB9C
```

* Grafikfenster Grundeinstellung. (Fenster wird nicht gelöscht,nimmt wieder den ganzen Bildschirm ein,es wird ein ORIGIN Ø,Ø und MOVE Ø,Ø durchgeführt und Vorder-/ Hintergrund auf INK 1 / INK Ø gesetzt).

(AF,BC,DE,HL)

```
CALL BBBA
```

* Move absolut.

(AF,BC,DE,HL)

```
LD DE,neue X-Koordinate ( Breite )
LD HL,neue Y-Koordinate ( Höhe )
CALL BBCØ
```

* Move relativ.

(AF,BC,DE,HL)

```
LD DE,neue X-Koord. mit Vorzeichen
LD HL,neue y-Koord.      "
CALL BBC3
```

* Origin setzen.

(AF,BC,DE,HL)

```
LD DE,neue X-Koord.
LD HL,neue Y-Koord.
CALL BBC9
```

* Hole Origin.

(DE,HL)

```
CALL BBCC
DE enthält X-Koord.
HL enthält Y-Koord.
```

* Setze neue Paperfarbe (als Inknummer).

(AF,HL)

```
LD A,Inknummer
CALL BB96
```

* Hole aktuelle Paperfarbe. (als Inknummer).

(AF)

```
CALL  BB99
A enthält Inknummer des Hintergrundes
```

* Hole aktuelle Position des Grafikcursors.

(AF)

```
CALL BBC6
DE enthält X-Koord.
HL enthält y-Koord.
```

* Setzen eines Grafikfensters.
(Die Koordinaten müssen nicht in der Reihenfolge kleiner / größer eingegeben werden, aber folgende Bedingung erfüllen).

	Linker Rand:	Rechter Rand:
Mode Ø:	vielfaches von 2	vielfaches von 2 - 1
Mode 1:	" " 4	" " 4 - 1
Mode 2:	" " 8	" " 8 - 1

(AF,BC,DE,HL)

```
LD DE,X-Koord.1
LD HL,X-Koord.2
CALL BBCF
LD DE,Y-Koord.1
LD HL.Y-Koord.2
CALL BBD2
```

* Lösche das Grafikfenster.

(AF,BC,DE,HL)

```
CALL BBDB
```

* Setze Grafikink. (Inknummer).
Es werden nur so viele Inknummern berücksichtigt, wie im momentanen Modus darstellbar sind !

(AF)

```
LD A,Inknummer
CALL BBDE
```

* Hole aktuelle Grafikink. (Inknummer).

(AF)

```
CALL BBE1
A enthält Inknummer
```

* Setze Grafikhintergrund auf neue Ink. (Inknummer).

(AF)

```
LD A, Inknummer
CALL BBE4
```

* Hole aktuellen Grafikhintergrund. (Inknummer).

(AF)

```
CALL BBE7
A enthält Inknummer
```

* Plotte Punkt absolut.

```
        ( AF,BC,DE,HL )

        LD DE,X-Koord.
        LD HL,Y-Koord.
        CALL BBEA
```

* Plotte Punkt relativ.

```
        ( AF,BC,DE,HL )

        LD DE,X-Koord. mit Vorzeichen
        LD HL,Y-Koord.        "
        CALL BBED
```

* Teste einen Punkt absolut.

```
        ( AF,BC,DE,HL )

        LD DE,X-Koord.
        LD HL,Y-Koord.
        CALL BBFØ
        A enthält Inknummer von Punkt od. Inknr. v. Paper
```

* Teste einen Punkt relativ.

```
        ( AF,BC,DE,HL )

        LD DE,X-Koord. mit Vorzeichen
        LD HL,Y-Koord.        "
        CALL BBF3
        A enthält Inknummer von Punkt od. Inknr. v. Paper
```

* Zeichne Linie absolut. (Basic-Befehl DRAW)

```
        ( AF,BC,DE,HL )

        LD DE,X-Koord.
        LD HL,Y-Koord.
        CALL BBF6
```

* Zeichne Linie relativ. (Basic-Befehl DRAWR)

```
        ( AF,BC,DE,HL )

        LD DE,X-Koord.
        LD HL,Y-Koord.
        CALL BBF9
```



```
*****************************************************************
*                                                               *
*         Wenn Sie nur waagrechte oder  senkrechte              *
*         Linien ziehen wollen, gibt es dafür zwei              *
*              wesentlich schnellere Routinen.                  *
*                                                               *
*****************************************************************
```

* Ziehe waagrechte Linie.

```
        ( AF,BC,DE,HL )

        LD A,Inknummer
        LD DE,Anfangs-X-Koord.
        LD BC,     End-X-Koord.
        LD HL,          Y-Koord.
        CALL BC5F
```

* Ziehe senkrechte Linie.

```
        ( AF,BC,DE,HL )

        LD A,Inknummer
        LD HL,Anfangs-Y-Koord.
        LD BC,     End-Y-Koord.
        LD DE,          X-Koord.
        CALL BC62
```

* Ein Zeichen auf Grafikcursorposition ausgeben.
(Vergleichbar mit BASIC-Befehlen TAG/TAGOFF.)

```
        ( AF,BC,DE,HL )

        A enthält das Zeichen (Ø-255)
        CALL BBFC
```

* Frage nach Zeichen pro Zeile. (Je nach Mode verschieden.)

```
        ( AF,BC )

        CALL BC17
        B enthält Spalten (19,39,79 bei Modis Ø,1,2).
        C enthält Zeilen (immer 24).
```

* Invertiere eine Zeichenposition.

```
        ( AF,BC,DE,HL )

        LD B,verschlüsselte Ink 1 (siehe nächste Routine)
        LD C,verschlüsselte Ink 2 (  "     "      "   )
        LD H,Spalte
        LD L,Zeile
        CALL BC4A
```

* Bildschirmmodus setzen (Ø,1 oder 2).

```
( AF,BC,DE,HL )

LD A, modus
CALL BCØE
```

* Bildschirmmodus holen.

```
( AF )

CALL BC11
A enthält je nach Mode Ø,1 oder 2.
```

* Bildschirm löschen mit Ink Ø.

```
( AF,BC,DE,HL )

CALL BC14
```

* Inknummer verschlüsseln.

```
( AF )

LD A,Inknummer
CALL BC2C
A enthält verschlüsslte Ink ( sowie sie die modeabhängige
  Bildschirmdarstellung einer Farbe benötigt.
```

* Ganzen Bildschirm um eine Zeile nach oben oder unten rollen.
Hardwarescroll-Routine, d.h. der Zeiger für den Anfang des Bildschirmspeicher steht nicht mehr auf &C000.

```
( AF,BC,DE,HL )

B = Ø, dann nach unten rollen
B <>Ø, dann nach oben rollen
A enthält verschlüsselte Ink für neu eingefügte
          Zeile. ( siehe vorige Routine.)
CALL BC4D
```

* Rolle einen Bildschirmteil um eine Zeile nach oben oder unten.
Softwarescroll-Routine, d.h. der Bildschirm wird wirklich in den angegeben Grenzen gerollt. Dies ist zwar nicht so schnell, dafür werden aber keine Zeiger verstellt. Wichtig bei manchen Hardcopy-Programmen.

```
( AF,BC,DE,HL )

B = Ø, dann nach unten rollen
B <>Ø, dann nach oben rollen
A enthält verschlüsselte Ink (wie vor).
H enthält linke  Begrenzung
D    "    rechte      "
L    "    obere       "
E    "    untere      "
CALL BC5Ø
```


Disketten- und Kassettenroutinen

Im folgenden möchte ich die Kassetten- und Disketten- Sprungvektoren kurz vorstellen . Da alleine dieser Teil das ganze Buch füllen könnte, wollte man die Routinen bis zum letzten erklären, verweise ich -um Wiederholungen zu vermeiden- an die im Anhang vorgestellte Literatur.

Doch den Begriff "Header", der gleich öfter auftaucht, möchte ich eingehender erläutern.

Allgemein: Ein Header ist ein 64 Byte langer Block, bei dem allerdings nur die ersten 28 Bytes vom AMSDOS verwendet sind, der Rest steht dem Anwender zur vollen Verfügung. Ferner ist dabei der Aufbau bei Kassetten- und Diskettenbetrieb gleich. Dieser Header oder Kopfsatz wird bei Kassettenbetrieb vor jedem neu aufgezeichneten Block mit abgespeichert und bei der Verwendung einer Floppy nur einmal am Anfang des Aufzeichnungsvorgangs. Die Ausnahme von der Regel bilden dabei die ASCII- und COM- Dateien, diese werden nämlich Headerlos abgespeichert. Weshalb sich auch beide Dateiarten sehr ähnlich sind. Wenn Sie also COM-Dateien unter normalen Umständen (ohne CP/M !) bearbeiten wollen, kommen Sie also am besten zu Rande, wenn Sie sie in Maschinensprache wie ASCII-Dateien behandeln.

Und nun der Aufbau des Headers in Bytes:

```
   ( Alle mit "!" gekennzeichneten Bytes haben bei der Disk
   keine Bedeutung !)

   Ø-15  Dateiname, auf jeweiliges Format (D/K) gebracht.
!    16  Blocknummer.
!    17  Wenn <>Ø dann letzter Block.
     18  Dateityp,(ASCII,Basic,Binär und geschützt ja/nein).
! 19-2Ø  Datenbytes/Satz.
  21-22  Ursprungsadresse, wichtig bei Binär-Files.
!    23  Wenn <>Ø, dann erster Block.
  24-25  File-Länge.
  26-27  Einsprungsadresse bei Maschinen-Code.
  28-63  Frei für Anwender.
```

---

* Initialisiere Kassettenmanager.
(Schließe alle offenen Dateien, setze 1ØØØ Baud, laße Meldungen zu.)

```
( AF,BC,DE,HL )

CALL BC65
```

* Setze Schreibgeschwindigkeit.

```
( AF,HL )

LD HL mit #A7 f. 2ØØØ Baud od. #14D f. 1ØØØ Baud.
LD A  mit #32 "   "  "   "  #19  "   "   " .
CALL BC68
```

* Kassettenmeldungen ein/aus.

(AF)

LD A mit Ø (=ein),<>Ø (=aus)
CALL BC6B

* Recorder-Motor ein.

(AF)

CALL BC6E
A enthält vorigen Motorstatus (#1Ø=ein/#EF=aus)
Flag's:
C = 1, wenn's funktionierte oder
C = Ø wenn Escape-Taste gedrückt war.

* Recorder-Motor aus.

(AF)

CALL BC71
sonst wie oben.

* Motor ein oder aus.

(AF)

LD A mit #1Ø f. Ein oder #EF f. Aus
CALL BC74
sonst wie oben

* Eröffne Eingabedatei auf Kas./Disc.

(AF,BC,DE,HL,IX)

LD B mit Länge des Namens
LD HL mit Adresse von Dateinamen
LD DE mit Adresse eines 2k-Buffers
CALL BC77
HL enthält Adresse von Header.
DE enthält Ursprungsadresse aus Header (wichtig bei
Binär-Dateien, bei Basic-Prog. immer #17Ø.)
BC enthält Dateilänge
A enthält Dateiart (es zählt nur das Low-Nibble!)
Ø = Basic
1 = Basic geschützt
2 = Binär
3 = Binär geschützt
6 = ASCII
Flag's bei Aussprung:
C=1,Z=Ø wenn's funktionierte.
C=Ø,Z=Ø wenn schon eine Eingabedatei offen war.
C=Ø,Z=1 wenn Datei auf Disk nicht gefunden oder
Escape-Taste gedrückt bei Kass.-Betrieb.

* Schließe Eingabedatei normal.

(AF,BC,DE,HL)

CALL BC7A
Flag's bei Aussprung:
C=1 wenn ok und
C=Ø wenn Datei nicht offen war.

* Schließe Eingabedatei unmittelbar.
(Abbruch im Fehlerfall.)

(AF,BC,DE,HL)

CALL BC7D

* Zeichenweise Lesen aus Eingabedatei.
(über 2k-Buffer.)

(AF,IX)

CALL BC8Ø
A enthält Zeichen, wenn Lesen möglich war.
Flag's bei Aussprung:
C=1,Z=Ø wenn Lesen möglich war.
C=Ø,Z=Ø bei Dateiende (EOF) oder nicht geöffneter
Datei.
C=Ø,Z=1 wenn Escape-Taste gedrückt war.

* Lies gesamte Datei direkt in Speicher.
(Nicht über Buffer.)

(AF,BC,DE,HL)

LD HL mit Adresse ab der die Daten abgelegt werden.
CALL BC83
Flag's bei Aussprung:
C=1,Z=Ø wenn's funktionierte.
C=Ø,Z=Ø wenn Datei nicht offen war.
C=Ø,Z=1 bei Escape-Taste.

* Teste auf EOF. (End Of File =#1A)
(Routine vor allem in Verbindung mit zeichenweisem Lesen, um zu verhindern,daß man über's Dateiende liest.)

(AF,IX)

CALL BC89
Flag's bei Aussprung:
C=1,Z=Ø wenn Ende noch nicht erreicht.
C=Ø,Z=Ø wenn Ende erreicht.
C=Ø,Z=1 bei Escape-Taste.

* Eröffne Ausgabedatei.

(AF,BC,DE,HL,IX)

LD B mit Länge des Namens.
LD HL mit Adresse des Namens.
LD DE mit Bufferadresse (2k).
CALL BC8C
HL enthält die Adresse des angelegten Headers.
Flag's bei Aussprung:
C=Ø wenn schon eine Datei offen war.
C=1 alles klar,Datei offen.

* Schließe Ausgabedatei normal.

(AF,BC,DE,HL,IX)

CALL BC8F
Flag's bei Aussprung:
C=1,Z=Ø wenn alles klar.
C=Ø,Z=Ø wenn Datei nicht offen war.
C=Ø,Z=1 wenn Escape-Taste gedrückt war.

* Schließe Ausgabedatei unmittelbar.
(Alle Zeichen, die noch im Buffer stehen und noch nicht ausgegeben waren, sind verloren !)

(AF,BC,DE,HL)

CALL BC92

* Zeichenweises Ausgeben an Datei.
(über Buffer.)

(AF,IX)

A enthält das Ausgabezeichen.
CALL BC95
Flag's bei Aussprung:
C=1,Z=Ø wenn's funktionierte.
C=Ø,Z=Ø wenn Datei nicht geöffnet war.
C=Ø,Z=1 bei Escape-Taste.

* Gib Datei im Ganzen aus.
(Nicht über Buffer.)

(AF,BC,DE,HL,IX)

LD HL mit der Adresse der Daten.
LD DE mit Anzahl der Daten-Bytes.
LD BC mit der Einsprungadresse (Binär-File !)
LD A mit Dateiart.
CALL BC98

Flag's bei Aussprung:
C=1,Z=Ø wenn's funktionierte
C=Ø,Z=Ø wenn Datei nicht offen war.
C=Ø,Z=1 bei Escape-Taste.

* Disc/Kass. Katalog.

(AF,BC,DE,HL,IX)

LD DE mit Adresse eines 2k-Buffers
CALL BC9B
Flag's bei Aussprung:
C=1, dann alles OK sonst
C=Ø wenn eine Eingabedatei offen war.

Sonstige Call's

* Füge neue Basicbefehle (RSX) ins Betriebssystem ein.
(Erweiterung muß im zentralen 32k-RAM liegen.)
(Anwendung siehe auch bei "Neue Basic-Befehle".)

(DE)

LD BC mit Adresse der RSX-Kommandotabelle.
LD HL m.Adresse eines 4-Byte-Buffers f.Verwaltung.
CALL BCD1

* Suche nach der Startadresse irgendeines Befehls.
(Wenn es sich dabei um einen Befehl des Grundbasic's -ohne DOS- handelt, wird er auch ausgeführt, nicht nur gesucht!)

(AF,BC,DE)

Befehl:DEFM "DIS","C"+#8Ø
LD HL,Befehl
CALL BCD4
HL enthält die Adresse der Routine und
C enthält das ROM-Auswahlbyte (bei diesem Bsp.7).
Flag's bei Aussprung:
C=1, wenn Befehl gefunden wurde oder
C=Ø wenn nicht, dann C und HL zerstört.

* Die Zeit bitte !

(DE,HL)

CALL BDØD
DEHL enthalten den 4-Byte-Wert der abgelaufenen Zeit seit dem Einschalten in 3Østel Sec..

* Setze neue Zeit.

(AF)

DEHL enthalten die "Anwender-Zeit"
CALL BD10

* Setze den Indirekten Sprung der Druckerroutine zurück.

(AF,BC,DE,HL)

CALL BD28

* Versuche ein Zeichen an Drucker zu senden.
(Bit 7 wird ignoriert, siehe auch 8-Bit-Ausgang in diesem Buch.)

(AF)

LD A mit Zeichen
CALL BD2B
Flag's bei Aussprung:
C=1 wenn's funktionierte und
C=0 wenn nicht (es wird nur 0.4 sec. versucht ein Zeichen zu senden.)

* Teste Busy Printer.

(-F)

CALL BD2E
Flag's bei Aussprung
C=1 wenn Drucker beschäftigt sonst
C=0 wenn bereit.

* Sende Zeichen an Drucker.

(AF)

CALL BD31
C-Flag immer 1, Zeichen ist verloren, wenn Drucker beschäftigt war.

* Schalte aktives oberes ROM bis zum Widerruf ein.

(AF)

CALL B900
A enthält ROM-Status

* Sperre oberes ROM wieder.

(AF)

CALL B903
A enthält ROM-Status

* Schalte unteres ROM ein.

(AF)

CALL B906
A enthält ROM-Status

* Sperre unteres ROM wieder.

(AF)

CALL B909
A enthält ROM-Status

* Stelle alten ROM-Status her.

(AF)

LD A mit altem ROM-Status aus den vorigen Routinen.
CALL B90C

Indirektions

Die Entwickler des CPC gaben dem Anwender noch eine weitere, sehr trickreiche, Möglichkeit mit auf dem Weg um ins Betriebssystem einzugreifen. Ich meine damit die sogenannten "Indirekten Sprünge" oder kurz "Indirections".
Diese Sprünge der Sorte C3 xx xx (also ganz normale Jumps) haben es in sich, denn mit ihrer Hilfe können Sie den Sinn ganzer Firmware-Routinen ändern. Weshalb das so ist, will ich an einem kurzen Beispiel erklären.
Verfolgen wir dazu den Ablauf eines ganz normalen Vorgangs, das Ausgeben eines Zeichens auf den Drucker:
Es beginnt also mit einem Sprung zur RAM-Adresse &BD2B. An dieser Stelle steht nun ein weiterer Sprung irgendwo ins untere ROM des Computers. Bevor nun im ROM die Bedienung des Centronicsport eingeleitet wird, und damit keine weitere Eingriffsmöglichkeit für uns mehr besteht, wird wieder zurück ins RAM, zur Adresse &BDF1 gesprungen. An dieser Stelle steht nun -vielleicht enttäuschend für Sie- wiederum ein Sprung zurück ins ROM, wo nun wirklich das Zeichen in unserem Beispiel ausgegeben wird.
Diese Umwege über das RAM geben uns aber die Möglichkeit eigene Routinen über die Indirections einzuschleifen und somit, wenigsten zum Teil, das Verhalten der Firmware zu beeinflußen. Wie ? - Na, mit einem ganz einfachen Ändern des Adressteiles im Jump bei Adresse &BDF1 in unserem Beispiel.
Insgesamt gibt es 13 Stück dieser sehr vielseitigen Jumps, ab Adresse &BDCD im RAM des Computers. An dieser Stelle möchte ich davon nur drei, meiner Meinung nach wirklich wichtige herausgreifen.

An Adresse &BDD9 : TEXT OUT AKTION (Zeichen auf Bildschirm geben.)
" " &BDEE : TEST BREAK (Ein RET an dieser Stelle verhindert z.B. ein Unterbrechen des Programmes.)
" " &BDF1 : WAIT PRINTER (Zeichen an Drucker ausgeben.)

(Zwei Programme in diesem Buch machen von diesen Möglichkeiten gebrauch.)

BDF1: JP 077B
BDD9: JP 140C
BDEE: JP 1C27

Fließkomma- und Integerroutinen

Da auch die Arithmetikroutinen der Schneider-Computer über Sprungtabellen integriert sind und daher eine definierte Schnittstelle besitzen, können Sie diese mühelos in ihre Programme einbauen.

Allerdings tauchen hier die ersten Unterschiede der verschiedenen CPC-Rechner auf. Die Fließkommaroutinen sind ja bloß etwas versetzt gegenüber dem 464, die Integerroutinen fehlen in der Sprungtabelle der 664 - und 6128 - Maschinen jedoch ganz. Außerdem sind die Integer-Routinen selbst im Basic-ROM statt -wie beim 464- im Betriebssystem-Rom.

Wieder habe ich die von der Systemroutine benützten Register in Klammer gesetzt. Also PUSHen und POPen nicht vergessen!

** Fließkommaarithmetik **

Eigentlich müssen Sie sich um die interne Darstellung einer Fließkommazahl überhaupt nicht kümmern, denn Sie geben nur die Adressen! der jeweiligen 5-Byte-langen Zahlen an, und alles übrige macht das Betriebssystem.

Die mit den Zeichen *)(* abgegrenzten Adressen geben in der Reihenfolge 664 und 6128 die Einspünge der Nachfolgersysteme von Schneider an.

* Kopiere Fließkommazahl von (DE) nach (HL).

```
(--)

LD DE,Adr. von Fließkommazahl
LD HL,neue Adr.Fließkommazahl
CALL BD3D              > BD5E,BD61 <
```

* Wandle Integerzahl HL nach Fließkommazahl (HL).

```
(AF,DE,HL)

LD HL,Integerzahl (0000-FFFF)
LD DE,Adr.für neue FK-Zahl
CALL BD40              > BD61,BD64 <
HL zeigt nun auf Fließkommazahl
```

* Wandle Fließkommazahl (HL) nach Integerzahl HL.
(Vorzeichen der Fließkommazahl wird nicht berücksichtigt.)

```
(AF,HL,IX)

LD HL,Adr.von FK-Zahl
CALL BD46              > BD67,BD6A <
HL enthält entsprechende positive Integerzahl.
```


* Subtraktion I, (HL)=(HL)-(DE).

```
(AF,BC,DE,IX,IY)

LD HL, Adr.von FK-Zahl-1
LD DE, Adr.von FK-Zahl-2
CALL BD5B              > BD7F,BD82 <
HL zeigt auf Ergebnis
```

* Subtraktion II, (HL)=(DE)-(HL).

```
(AF,BC,DE,IX,IY)

LD HL, Adr.von FK-Zahl-1
LD DE, Adr.von FK-Zahl-2
CALL BD5E              > ----,---- <
HL zeigt auf Ergebnis
```

* Addition (HL)=(HL)+(DE).

```
(AF,BC,DE,IX,IY)

LD HL, Adr.von FK-Zahl-1
LD DE, Adr.von FK-Zahl-2
CALL BD58              > BD79,BD7C <
HL zeigt auf Ergebnis
```

* Multiplikation (HL)=(HL)*(DE).

```
(AF,BC,DE,IX,IY)

LD HL, Adr.von FK-Zahl-1
LD DE, Adr.von FK-Zahl-2
CALL BD61              > BD82,BD85 <
HL zeigt auf Ergebnis
```

* Division (HL)=(HL)/(DE).

```
(AF,BC,DE,IX,IY)

LD HL, Adr.von FK-Zahl-1
LD DE, Adr.von FK-Zahl-2
CALL BD64              > BD85,BD88 <
HL zeigt auf Ergebnis
```

* Zahl mit 2^a multiplizieren, (HL)=(HL)*2^ACCU.

```
(AF,BC,IX)

LD HL, Adr.von FK-Zahl
LD A,  Potenz
CALL BD67              > ----,---- <
HL zeigt auf Ergebnis
```

```
* Zahl mit 10^a multiplizieren.  (HL)=(HL)*10^ACCU.

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        LD A,  Potenz
        CALL BD55           > BD76,BD79 <
        HL zeigt auf Ergebnis

* Logarithmus zur Basis 10.  (HL)=LOG10(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD82           > BDA3,BDA6 <
        HL zeigt auf Ergebnis


* Natürlicher Logarithmus.  (HL)=LOG(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD7F           > BDA0,BDA3 <
        HL zeigt auf Ergebnis


* Exponent  (HL)=EXP(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD85           > BDA6,BDA9 <
        HL zeigt auf Ergebnis

* Wurzelziehen  (HL)=SQR(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD79           > BD9A,BD9D <
        HL zeigt auf Ergebnis

* Cosinus  (HL)=COS(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD8B           > BDAC,BDAF <
        HL zeigt auf Ergebnis

* Sinus  (HL)=SIN(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD88           > BDA9,BDAC <
        HL zeigt auf Ergebnis
```



```
* Tangens  (HL)=TAN(HL).

        (AF,BC,DE,IX)

        LD HL, Adr,von FK-Zahl
        CALL BD8E           > BDAF,BDB2 <
        HL zeigt auf Ergebnis

* Arcustangens  (HL)=ATN(HL).

        (AF,BC,DE,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD91           > BDB2,BDB5 <
        HL zeigt auf Ergebnis


* Hole Konstante PI nach (HL).  (HL)=PI.

        (DE)

        LD HL, freier Speicherplatz für PI
        CALL BD76           > BD97,BD9A <
        PI ist nach (HL) kopiert

* DEG / RAD Umschaltung.

        (-)

        LD A, (00 für RAD oder <>00 für DEG)
        CALL BD73           > BD94,BD97 <
        ( Bei CPC  464 ist die Speicherzelle B8F7  und  bei
        CPC 664/6128 B113 entsprechend geändert.)


* Vorzeichenwechsel der Fließkommazahl (HL).

        (AF,IX)

        LD HL, Adr.von FK-Zahl
        CALL BD6D           > BD8E,BD91 <
        HL unverändert.
        (Zahl hat Vorzeichen von + nach - od. von - nach +
        gewechselt.)

* Vergleiche die Fließkommazahlen (HL) und (DE) miteinander.

        (AF,IX,IY)

        LD HL, Adr.von FK-Zahl-1
        LD DE, Adr.von FK-Zahl-2
        CALL BD6A           > BD8B,BD8E <
        HL und DE unverändert.
        Flag's bei Aussprung je nach Ergebnis:
        (DE) < (HL)         C=0 , Z=0
        (DE) > (HL)         C=1 , Z=0
        (DE) = (HL)         C=0 , Z=1
```

* Stelle Vorzeichen (Signum) fest.

```
(AF,IX)

LD HL, Adr von FK-Zahl
CALL BD7Ø            > BD91,BD94 <
HL unverändert.
Flag's bei Aussprung je nach Ergebnis:
(HL) < Ø          C=1 , Z=Ø
(HL) = Ø          C=Ø , Z=1
(HL) > Ø          C=Ø , Z=Ø
```

****

Dies waren alle Fließkommaroutinen der Schneider-Rechner, und nun noch einige wichtige Konstanten im ROM des 464 mit ihren Adressen und Inhalten. Die 664- und 6128- Computer besitzen diese Konstanten ebenfalls, aber an anderer Stelle. Bedenken Sie, daß bei Aufruf einer Rechen-Routine automatisch das ROM eingeschaltet wird, Sie müssen also nur die Register HL oder DE mit den Adressen laden.

****

** Fließkommakonstanten im ROM des CPC 464 **

Adresse in Hex.	Inhalt in Hex.	Konstante
2F53	ØØ ØØ ØØ 2Ø 84	1Ø
2F58	ØØ ØØ ØØ 48 87	1ØØ
2F5D	ØØ ØØ ØØ 7A 8A	1ØØØ
2F62	ØØ ØØ 4Ø 1C 8E	1ØØØØ
2F67	ØØ ØØ 5Ø 43 91	1ØØØØØ
2F6C	ØØ ØØ 24 74 94	1ØØØØØØ
2F71	ØØ 8Ø 96 18 98	1ØØØØØØØ
2F76	ØØ 2Ø BC 3E 9B	1ØØØØØØØØ
2F7B	ØØ 2Ø 6B 6E 9E	1E9
2F8Ø	ØØ F9 Ø2 15 A2	1E1Ø
2F85	4Ø B7 43 3A A5	1E11
2F8A	1Ø B5 D4 6Ø A8	1E12
2F8F	2A E7 84 11 AC	1E13
3Ø81	34 F3 Ø4 35 8Ø	.7Ø7106781 =1/SQR(2)
3Ø86	F8 17 72 31 8Ø	.693147181 =LOG(2)
3Ø8B	85 9A 2Ø 1A 7F	.3Ø1Ø29996 =LOG1Ø(2)
3ØCC	ØØ ØØ ØØ ØØ 8Ø	Ø.5
31A9	A2 DA ØF 49 82	3.14159265 =PI
3205	A2 DA ØF 49 81	1.57Ø79633 =PI/2
321D	6E 83 F9 22 7F	.318Ø9886 =1/PI
3222	B6 6Ø ØB 36 79	5.55556E-3 =1/18Ø
3227	13 35 FA ØE 7B	1.74533E-2 =PI/18Ø
322C	D3 EØ 2E 65 86	57.2957795 =18Ø/PI
328A	FE FF FF FF 7F	-Ø.5
3332	ØØ ØØ ØØ ØØ 81	1

** Integerarithmetik **

Im Gegensatz zur Fließkommaverarbeitung tragen die Register die Zahlen selbst und keine Adressen mehr!
Bei Berücksichtigung des Vorzeichens stehen die Zahlen -32768 bis +32767 zur Verfügung und ohne Vorzeichen die Zahlenreihe Ø bis 65535. Das höchstwertige Bit (MSB) der Zahl stellt gleichzeitig auch das Vorzeichen dar. Wenn MSB=1, dann ist die Zahl negativ, sonst positiv.

Bei den nunmehr folgenden Routinen besteht allerdings ein großer Unterschied zwischen CPC 464 auf der einen und den Rechnern CPC 664 und 6128 auf der anderen Seite.
Während bei dem 464 alle Routinen ebenfalls über Sprungtabellen laufen, sind bei den anderen Rechnern diese nicht mehr in den Tabellen zu finden. Außerdem sind sie auch noch im Basic-ROM versteckt.

Konkret heißt das, daß Ihre Programme um einen CALL-Befehl länger werden. Anstatt lediglich einem CALL BDAC beim CPC 464 müssen Sie nun die Maschinencodefolge:

```
CALL B9ØØ          ¦Basic-Rom dazuschalten
CALL DD4F          ¦Eigentliche Routine (bei 664)
```

für die Addition von HL- und DE-Register eingeben. Dabei braucht der Aufruf CALL B9ØØ nur einmal am Anfang Ihres Maschinencode-Programmes zu stehen, das Basic-Rom ist dann immer aktiviert. Sie können dann das darunterliegende RAM nur noch mit Schreibzugriffen erreichen.

Die angefügten Adressen gelten wieder für die beiden neuen Rechner und sind mit ">><" kenntlich gemacht, ebenfalls sind die verwendeten Register in Klammer.

---

* Addiere HL=HL+DE.

```
(AF)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDAC            > DD4F,DD4A <
HL enthält das Ergebnis
Flag's bei Aussprung:
C=Ø wenn MSB=1 sonst C=1
```

* Subtrahiere HL=DE-HL.

```
(AF,DE)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDB2            > DD57,DD52 <
HL enthält das Ergebnis
Flag's bei Aussprung:
C=Ø wenn MSB=1 sonst C=1
```

* Subtrahiere HL=HL-DE.

```
(AF)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDAF              > DD58,DD53 <
HL enthält das Ergebnis
Flag's bei Aussprung:
C=Ø wenn MSB=1 sonst C=1
```

* Multipliziere mit Vorzeichen HL=HL*DE.

```
(AF,BC)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDB5              > DD6Ø,DD5B <
HL enthält das Ergebnis
DE unverändert
Flag's nach Aussprung:
S=1 wenn Ergebnis negativ sonst S=Ø
```

* Multipliziere ohne Vorzeichen HL=HL*DE.

```
(AF,BC)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDBE              > DD77,DD72 <
HL enthält das Ergebnis
DE unverändert
Flag's nach Aussprung:
C=1 wenn Ergebnis größer als 65535 (hex FFFF)
```

* Dividiere mit Vorzeichen HL=HL/DE.

```
(AF,BC)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDB8              > DDA1,DD9C <
HL enthält das Ergebnis
DE den möglichen Rest
Flag's bei Aussprung:
S=1 wenn Ergebnis negativ sonst S=Ø.
```

* Dividiere ohne Vorzeichen HL=HL/DE.

```
(AF,BC)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDC1              > DDB3,DDAE <
HL enthalt das Ergebnis
DE den möglichen Rest
```


* Vergleiche die Inhalte von DE-und HL-Register.

```
(AF)

LD HL, Zahl-1
LD DE, Zahl-2
CALL BDC4              > DEØ7,DEØ2 <
HL und DE unverändert
Flag's bei Aussprung:
HL > DE    C=1, Z=Ø
HL = DE    C=Ø, Z=1
HL < DE    C=Ø, Z=Ø
```

* Vorzeichen wechseln.

```
( AF )

LD HL,Zahl
CALL BDC7              > DDF2,DDED <
HL enthält Zahl mit umgekehrten Vorzeichen.
Flag's bei Aussprung:
S = 1, wenn Zahl bei Einsprung positiv war und
S = Ø, wenn Zahl negativ war.
```

Restarts, Lowjumps, Interrupts

Der Z-8Ø besitzt 8 Restart-Adressen im unteren Speicher- bereich, die vorwiegend das Betriebssystem nützt, aber auch vom Anwender sehr vielseitig verwendet werden können. Dazwischen gibt es noch einige Einsprünge, die nicht über RST xx erreicht werden können, sondern die CALL- oder JP-Befehle erfordern und hier als Lowjumps bezeichnet sind.

Schließlich gibt es noch die Interruptservice-Routine bei Adresse ØØ38 hex. (Der Z-8Ø arbeitet beim Schneider im Interruptmodus 1.).

Restart Ø : #ØØØØ

Power-Up-Entry. Einsprung beim Einschalten des Rechners oder einem Reset. Der Speicher wird dabei gelöscht.

Restart 1 : #ØØØ8

Sprung ins untere ROM oder RAM mit einer Inline-Adresse, d.h. dem RST-Aufruf folgt als Datenwort eine 14-Bit-Adresse. Dabei werden keine Register verfälscht ! Die ersten beiden Bits haben eine spezielle Bedeutung.

```
Anwendung :
RST   1  (oder RST #8, je nach Assembler.)
DEFW  Bit15, Bit14 + 14-Bit-Adresse
                Bit14 = 1 dann System-Rom gesperrt
      Bit15           = 1 dann Basic -Rom gesperrt
```

Call #ØØØB :

Wie Restart 1, nur daß hier die Inlineadresse nicht im Speicher, sondern im HL-Register steht. Alle anderen Register sind wieder frei für den Anwender.

Restart 2: #ØØ1Ø

Sprung in ein "Seiten-ROM". Dieser Restart ist nur interessant, wenn eine ROM-Erweiterung nicht mehr in 16k paßt und so der M-Code zwischen mehreren ROM's (max.4) aufgeteilt werden muß. Zu der auf dem Restart folgenden Adresse wird #CØØØ hinzuaddiert, die entsprechende Routine aufgerufen und ein anschließendes RET bringt den Programmcounter wieder zum aufrufenden Programm zurück (einschließlich alten ROM-State).

```
        Anwendung :
        RST   2    (oder RST #1Ø)
        DEFW  Bit15, Bit14, +14-Bit Adresse
              Bit15 +14 können die Werte Ø-3 annehmen und
              dieser Wert wird zur momentanen ROM-Select-
              Adresse hinzuaddiert.
```

Restart 3 : #ØØ18

Der wohl meist verwendete Aufruf. Mit ihm kann man jede Adresse in jedem ROM anspringen bzw. deren Inhalt lesen. Alle Register -mit Ausnahme von IY- werden an die Zielroutine übergeben.

```
        Anwendung :

daten:  DEFW adresse
        DEFB romselect
aufruf: RST 3      (oder RST #18)
        DEFW daten
        .... hier geht's weiter.
```

>adresse< ist ein Wert von ØØØØ bis FFFF, der ROM oder auch RAM anspricht, in den mittleren 32k des Speichers immer RAM. >romselect< kann folgende Werte annehmen:

#ØØ-#FB schaltet das ROM mit der entsprechenden Select-Adresse ein (#Ø7 ist z.B. Floppy-ROM) und sperrt das untere ROM (von #ØØØØ-#8ØØØ).
#FC schaltet das untere und obere ein.
#FD schaltet das untere aus und das obere ein.
#FE schaltet das untere ein und das obere aus.
#FF schaltet beide aus (Zugriff auf ganzes RAM).

Call #ØØ1B :

Wie Restart 3,nur daß hier die Adresse im HL-Register steht und das ROM- Select-Byte im C-Register.

Restart 4: #ØØ2Ø

Dieser Befehl wird verwendet, um bei eingeschalteten ROM's immer aus dem darunterliegenden RAM zu lesen. Entspricht einem "LD A,(HL)"-

Befehl. Das HL=Register enthält also die zu lesende Adresse.

CALL #ØØ23

Ähnlich Restart 3, nur mit dem Unterschied, daß jetzt das HL-Register die Aufgabe von "DEFW daten" übernimmt. Das HL- Register ist also beim Einsprung nicht mehr frei für den Anwender.

```
        Anwendung :

daten:  DEFW adresse
        DEFB romselect (siehe Restart 3)
aufruf: LD HL,daten
        CALL #ØØ23
```

Restart 5 : #ØØ28

Wird hauptsächlich durch die Firmware zum Aufruf einer Routine im unteren ROM verwendet, ihm muß als Datenwort die Adresse folgen. Dieser Restart kehrt nicht von selber zum aufrufenden Programm zurück, entspricht daher eher einem JP-Befehl. Um ihn dennoch zurückkehren zu lassen, muß man den Stack manipulieren wie im Beispiel gezeigt.

```
        Anwendung :

        LD HL,retour
        PUSH HL              ;Rückkehradresse auf Stack
        RST  5    (oder RST #28)
        DEFW adresse
retour: .....weiter im Programm.
```

Restart 6 : #ØØ3Ø

Dieser Aufruf ist vollkommen frei für den Anwender. Es stehen die Adressen #ØØ3Ø-#ØØ37 im RAM bereit zur Aufnahme einer Routine evtl. mit einem JP-Befehl, wenn der Platz nicht reicht. Bei Restart 6 ist immer das untere RAM selektiert.

Interruptservice-Routine #ØØ38

Das Betriebssystem kann unterscheiden zwischen einer Zeitunterbrechung (periodisch vom System selber erzeugt) oder einer externen Unterbrechung (vom Anwender erzeugt). Dies wird erreicht durch eine erneute Freischaltung des Interupts. Liegt das Signal der Unterbrechungsquelle dann immer noch an, wird davon ausgegangen, daß es sich um eine externe Unterbrechung handelt (Ein periodischer Interrupt währe zu diesem Zeitpunkt schon zurückgenommen worden.).

Handelt es sich um einen Anwender-Interrupt, wird der Code ab #ØØ3B im RAM ausgeführt (5 Bytes frei).Eventuell ist ein JP-Befehl ins zentrale RAM notwendig, wenn der Platz nicht ausreicht.Weitere Unterbrechungen sind nun gesperrt, und ROM - Routinen können nicht ausgeführt werden (Der Umschaltmechanismus funktioniert während eines Anwender-Interrupts nicht.). Die Software hierzu muß so schnell wie möglich abgearbeitet sein, um interne Vorgänge nicht vermeidbar zu verzögern, und am Schluß die Interruptquelle löschen.

Diese Löschung wird bei Verwendung von Z-80 I/O-Bausteinen durch den RETI-Befehl erreicht.

Alles in allem stellt dieser Interrupt auf unterster Ebene für den Anwender zwar ein praktikable, aber wenig flexible Methode dar. Eine andere und wesentlich bessere Möglichkeit zur Verarbeitung eines Interrupts bei den CPC's ist das Anstoßen eines Ereignisses. Dabei kann die Bedienroutine so ins Betriebssystem eingebunden werden, daß z.B. alle 1/300 Sekunden ein bestimmter Eingang einer PIO abgefragt wird. In diesem Fall kann auch der 8255 unbeschränkt "Interruptfähig" werden.

Dies möchte ich gleich in einem weiterem Teil der Werkzeugkiste erklären.

Schnelle- und langsame Ereignisse

Um es gleich vorweg zu sagen, die Verarbeitung eines Ereignisses ist kein lupenreiner Interrupt. Denn dann müßte ja auch irgendein externes Gerät (z.B. ein I/O-Port) diese Unterbrechung auslösen können !

Nein, es handelt sich dabei um periodische Unterbrechungen durch das Betriebssystem, die schnellen Ereignisse werden 300 mal pro Sekunde angestoßen und die langsamen in der Sekunde 50 mal.

Um dennoch -wie versprochen- irgendeinen Eingang des 8255-I/O-Ports unterbrechungsfähig zu machen, wird dieser Eingang 300 (oder 50) mal in der Sekunde abgefragt, ob ein bestimmtes Signal anliegt. Die Spezialisten unter uns werden jetzt sagen, daß dies nun eigentlich ein Polling - Verfahren sei (periodisches Abfragen und Warten auf einen Zustand), und sie haben recht ! Nur mit dem feinen Unterschied, daß der Rechner nicht pausenlos in einer Schleife hängt und wartet, und wartet..., sondern daß bei dieser Methode das Betriebssystem nicht nennenswert verzögert wird. Es geschieht ja "nur" alle 300stel (50stel) Sekunden.

Es gibt auch noch die Möglichkeit, die langsamen Ereignisse zusätzlich zu verzögern. Diese führen nämlich einen 16-Bit Zähler mit, der erst auf Null decrementiert sein muß, bevor das Ereignis nun wirklich ausgeführt wird. Damit läßt sich eine Verzögerung von ca. 22 Minuten einbauen. Ähnlichkeiten mit den Basic-Befehlen EVERY und AFTER sind nicht rein zufällig !

Aber jetzt zur Praxis !

** Die schnellen Ereignisse **

Zuerst muß dem Betriebssystem eine Tabelle zur Bedienung des Ereignisses angewiesen werden.

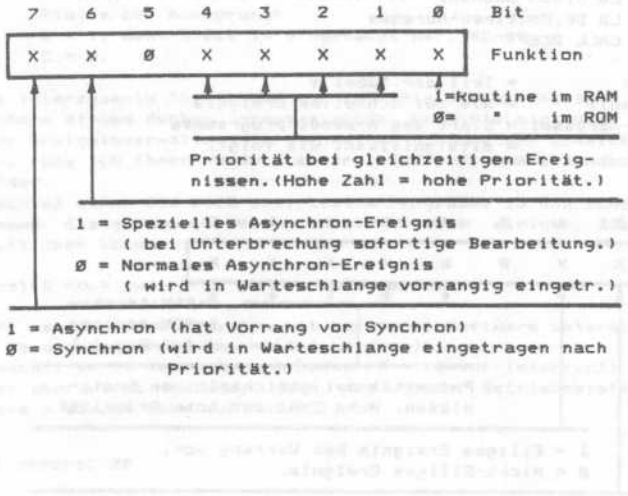
```
Tabelle: DEFW 0,0
         DEFB 0,0
         DEFW 0
         DEFB 0
```

Der Assemblerbefehl DEFS 9 hätte die gleiche Funktion erfüllt.

* Schnelles Ereignis initialisieren und einhängen.

```
( AF,DE,HL )

LD HL,Tabelle
LD DE,Routinen-Adressse
LD B,Klasse
LD C,Romauswahl (kann entfallen w. Routine im RAM.)
CALL BCE0
```

Tabelle: = Anfangsadresse der Tabelle (siehe oben).
Routinen-Adresse: = Startadresse des Anwender Programms.
Romauswahl: = z.B. 7 wenn Floppy-ROM gemeint ist.
Klasse: = Bitsignifikant wie folgt:



* Schnelles Ereignis wieder einhängen.
(Muß schon initialisiert gewesen sein.)

```
( AF,DE,HL )

LD HL,tabelle
CALL BCE3
```

* Lösche schnelles Ereignis aus Liste.

```
( AF,DE,HL )

LD HL,tabelle
CALL BCE6
```

** Die langsamen Ereignisse **

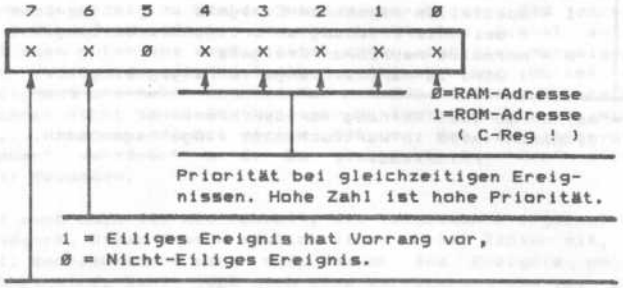
Zuerst wieder die Tabelle für's Betriebssytem, diesmal etwas länger:

```
Tabelle : DEFW Ø,Ø,Ø     oder:    Tabelle :  DEFS 6
Ereignis: DEFW Ø                  Ereignis:  DEFS 7
          DEFB Ø,Ø
          DEFW Ø
          DEFB Ø
```

* Langsames Ereignis initialisieren.
(Wird noch nicht eingehängt !)

```
        ( HL )

        LD HL,Ereignis
        LD B,Klasse
        LD C,ROM-Auswahl
        LD DE,Routinen-Adresse
        CALL BCEF
```

```
Ereignis:           = Teil der Tabelle
ROM-Auswahl:        = wie bei Schnellem Ereignis
Routinen-Adresse:   = Start des Anwenderprogrammes
Klasse:             = Bitsignifikant wie folgt:
```



* Langsames Ereignis einhängen.
(Muß schon mal initialisiert gewesen sein.)

```
        ( AF,BC,DE )

        LD HL,Tabelle
        LD DE,Zählerstart
        LD BC,Ladewert für Zähler
        CALL BCE9
```



```
Tabelle:            = siehe oben
Zählerstart         = Wert in 1/5Ø Sekunden, der angibt, wann
                      das Ereignis zum ersten mal nach dem
                      Einhängen aufgerufen wird. Muß )Ø sein.
                      Ist dabei der Ladewert (in BC) Ø,
                      handelt es sich um ein einmaliges
                      Ereignis.
Ladewert f.Zähler:  = Abstand in 1/5Ø Sekunden von einem Er-
                      eignis zum anderen.
```

* Langsames Ereignis wieder austragen.

```
        ( AF,DE,HL )

        LD HL,Tabelle
        CALL BCEC
        DE enhält den Stand des Zählers, wenn Block einge-
           hängt war sonst DE zerstört.)
        Flag's bei Aussprung:
        C = 1, wenn Block in eingehängt war, sonst
        C = Ø.
```

Dieses interessante Thema könnte noch viele Seiten füllen, was aber den Rahmen dieses Buches sprengen würde. Wenn Sie sich u.a. noch näher mit der Ereignisverwaltung der Schneider - Computer auseinandersetzen wollen, rate ich Ihnen zu der Lektüre des "Firmware-Handbuches" von Schneider.
Im Anschluß sehen Sie noch zwei kleine Programme zu den schnellen und langsamen Ereignissen, die vielleicht noch einige Zusammenhänge verdeutlichen können

Zum Schluß noch zwei Tips zu den Interrupts:

Der NMI ist im Betriebssystem mit keinerlei Software unterstützt, also nicht vorgesehen trotz Anschluß ! Leider !!
Ein Umschalten in den Interupt-Modus 2 (Vektor-Interrupt) ist zwar möglich aber nicht ratsam, denn das ganze Betriebssystem (plus Hardware) beruht eben auf Modus 1.

```
Pass 1 errors: ØØ

4E2Ø              1Ø          org  2ØØØØ
4E2Ø              2Ø          ent  2ØØØØ
                  25 ;Fast Ticker initialisieren
                  27 ;und einhaengen.
4E2Ø  213C4E      3Ø          ld   hl,tabell
4E23  11454E      4Ø          ld   de,adress
4E26  Ø683        5Ø          ld   b,%1ØØØØØ11
4E28  ØE ØØ       6Ø          ld   c,Ø
4E2A  CDE ØBC     7Ø          call #bce0
4E2D  C9          8Ø          ret
                  9Ø ;
                  95 ;Fast Ticker wieder
                  96 ;einhaengen.
4E2E  213C4E      1ØØ         ld   hl,tabell
4E31  CDE3BC      11Ø         call #bce3
4E34  C9          12Ø         ret
```

```
                    130 ;
                    135 ;Fast Ticker austragen.
4E35  213C4E        140         ld   hl,tabell
4E38  CDE6BC        150         call #bce6
4E3B  C9            160         ret
                    170 ;
                    175 ;Tabelle fuer interne
                    176 ;Verwaltung
4E3C  0000          180 tabell: defw 0
4E3E  0000          190         defw 0
4E40  00            200         defb 0
4E41  00            210         defb 0
4E42  0000          220         defw 0
4E44  00            230         defb 0
                    240 ;
                    245 ;Diese Routine wird 300 mal
                    246 ;in der Sekunde ausgefuehrt.
4E45  3E51          250 adress: ld   a,"Q"
4E47  CD5ABB        260         call #bb5a
4E4A  C9            270         ret

Pass 2 errors: 00
Pass 1 errors: 00

4E20                 10         org  20000
4E20                 20         ent  20000
                     30 ;Slow Ticker Ereignis
                     40 ;initialisieren
4E20  21484E         50         ld   hl,ereig
4E23  0680           60         ld   b,%10000000
4E25  0E00           70         ld   c,0
4E27  114F4E         80         ld   de,adress
4E2A  CDEFBC         90         call #bcef
4E2D  C9            100         ret
                    110 ;Slow Ticker Ereignis
                    120 ;einhaengen.
4E2E  21424E        130         ld   hl,tabell
4E31  116400        140         ld   de,100
4E34  016400        150         ld   bc,100
4E37  CDE9BC        160         call #bce9
4E3A  C9            170         ret
                    180 ;Slow Ticker wieder
                    190 ;austragen.
4E3B  21424E        200         ld   hl,tabell
4E3E  CDECBC        210         call #bcec
4E41  C9            220         ret
                    230 ;Tabelle fuer Ereignis-
                    240 ;verwaltung.
4E42                250 tabell: defs 6
4E48                280 ereig:  defs 7
                    330 ;
                    340 ;Dieses Programm wird alle
                    350 ;2 Sekunden ausgefuehrt.
4E4F  3E41          360 adress: ld   a,"A"
4E51  CD5ABB        370         call #bb5a
4E54  C9            380         ret

Pass 2 errors: 00
```


Stringdescriptoren, Variablenpointer und CALL-Befehle

Wenn Sie in Maschinensprache programmieren,wollen Sie sicher bald auf Werte, die Sie in Basic angelegt haben, seien es nun String-, Integer- oder Fließkommavariable, auch von Ihrer Maschinenroutine her zugreifen. Und um sich mit PEEK und POKE abzumühen, gehört wirklich der Vergangenheit an.

Dazu müssen Sie aber wissen, wie Ihr Rechner diese Variable verwaltet und für Sie bereithält. Keine Angst, Ihr Computer unterstützt Sie bei der Verwendung seiner Variablen sehr. Ich erkläre das am besten an ein paar Beispielen, nach dem Motto: "Ein Bild sagt mehr als Tausend Worte".

Zuerst mal ein kleiner Basic-Dreizeiler:

```
10 a$="Die Katze jagd "
20 b$="die Hausmaus."
30 c$=a$+b$
```

Dieses Programm (Das Originalprogramm war nur um eine Ausdruckroutine länger.) lieferte folgende Variablenpointer oder auf deutsch Variablenzeiger:

@a$= 02BB @b$= 02C2 @c$= 02C9.

Diese Werte können Sie sich mit dem Befehl:

z.B. PRINT HEX$(@a$,4)

leicht selber ausdrucken. Das Zeichen @ vor einer Variablen gibt ihre Speicheradresse, nicht deren Inhalt an. Dieser Zeiger auf eine Variable gibt im vorliegenden Fall eines Strings nur dessen Stringdescriptor an, nicht den String. Jetzt haben wir schon wieder ein neues Wort, Sie werden sich jetzt fragen, was nun so ein Stringdingsda eigentlich ist. Dieses neudeutsche Wort kann man auch mit "Zeichenketten-Aufschlüssler" übersetzen und bezeichnet auch dann nichts anderes als eine Zahlenkombination für die Länge und den Auffindungsort eines Strings im Speicher des Rechners. Aber bleiben wir ruhig beim englischen Wort, es klingt ja so toll.

Wahrscheinlich bekommen Sie- wenn Sie meinen Rat folgen, und diese Werte ausdrucken lassen- andere Zahlen, das liegt daran, daß Ihr Programm nicht genau so lang ist wie meines und sich deshalb der Variablenspeicher im CPC etwas verlagert, aber der Effekt ist derselbe.

Beim Ausdruck der Speicherumgebung unseres Variablenpointers sehen wir gleich die Descriptoren selber:

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

02B0  29 2C 19 32 00 00 00 00 00 C1 02 0F 80 01 00 00 ),.2............
02C0  C2 02 0D 9B 01 00 00 C3 02 1C 24 9C 00 C1 02 15 ..........$.....
```

An Speicherplatz 02BB steht 0F (dez.15), dies ist die Länge von a$ einschließlich Leerzeichen (Zählen Sie ruhig nach !). Danach folgt 80 und 01, diese beiden Bytes geben -in der richtigen Reihenfolge gelesen- den Wert 0180 und das ist nun die Adresse des ersten Zeichens von a$. Jetzt wissen wir auch, daß ein Stringdescriptor aus drei Bytes besteht. Für b$ und c$ gilt das gleiche.

Sehen wir auch mal bei Adresse 0180 nach:

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

0180  44 69 65 20 4B 61 74 7A 65 20 6A 61 67 64 20 22 Die Katze jagd "
0190  00 19 00 14 00 03 0C 00 E2 EF 22 64 69 65 20 48 ..........."die H
01A0  61 75 73 6D 61 75 73 2E 22 00 13 00 1E 00 03 13 ausmaus.".......
```

Hier steht tatsächlich in voller Deutlichkeit unser a$ und etwas später auch b$.
Die Abspeicherung von c$ folgt auch noch, denn sie ist aus zwei Gründen interessant.

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

9C20  2D 2D 2D 2D 44 69 65 20 4B 61 74 7A 65 20 6A 61 ----Die Katze ja
9C30  67 64 20 64 69 65 20 48 61 75 73 6D 61 75 73 2E gd die Hausmaus.
9C40  21 0A 9D FE 03 20 0F DD 7E 04 B7 28 09 CB C6 3E !.... ..~..(...>
```

Erstens der Platz selber, direkt unter HIMEM (HIMEM war hier auf 40000 od. 9C40 hex.).
Alle Strings, die irgenwie aus einer Berechnung, Verknüpfung oder sonst einer Manipulation hervorgegangen sind, werden in der Reihenfolge ihres Auftretens direkt unter der Speicherobergrenze abgelegt. Der Stringspeicher wächst also nach unten und kann irgendwann auf das Basic-Programm treffen, mit der Fehlermeldung "Memory full".

Zweitens war ja c$ aus a$ und b$ hervorgegangen und eigentlich nur -zusammengefügt- an einem neuen Platz abgelegt worden.

Das, was hier so unverdächtig klingt, wirft manchmal große Probleme auf. Stellen Sie sich nur mal ein Adressenprogramm vor, in dem ja bekanntlich bei der Sortierung viele Strings gegeneinander vertauscht werden müssen, jede auf diese Art verwurstelte Zeichenkette wird neu angelegt! also an einem neuen Speicherplatz kopiert.
Es versteht sich von selber, daß dabei sehr viel "Stringmüll" anfällt, der nicht mehr gebraucht wird, aber dennoch den Speicher belastet. Irgendwann wird es dem Computer zuviel und er räumt auf! Dieser Vorgang kann einige Minuten in Anspruch nehmen und heißt dann "Carbage Collection". Computer können ja nicht schwitzen, aber vielleicht Sie wenn Sie nicht wissen, ob Ihr gutes Stück sich nun "aufgehängt" hat oder nicht, denn der Rechner gibt währendessen keinerlei Lebenszeichen von sich.

Aber nun wieder zurück zum Thema.
Bei der Dimensionierung eines Stringfeldes stehen im Feld selber nur wieder unsere Descriptoren. Gleich mal ein Beispiel:

```
10 DIM a$(5)
20 FOR n=0 TO 5:a$(n)="Irgendwas":NEXT
```


Dieses Programm liefert als Variablenpointer von @a$(0),02C8 und an dieser Adresse beginnt unser Descriptorenfeld.

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

02C0  00 C1 02 15 00 01 06 00 09 A1 01 09 A1 01 09 A1 ................
02D0  01 09 A1 01 09 A1 01 09 A1 01 00 00 00 00 00 00 ................
```

Sie sehen alle Pointer zeigen auf denselben String an der selben Adresse, Basic macht es sich hier noch leicht. An 01A1 finden wir tatsächlich den String.

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

01A0  22 49 72 67 65 6E 64 77 61 73 22 01 B0 00 18 00 "Irgendwas".....
01B0  1E 00 BF 23 16 2C 20 FF 73 28 40 03 07 00 E1 28 ...#., .s($....(
```

Sobald wir noch folgende Basic-Zeile einfügen,

```
70 a$(3)=a$(3)+" und nochwas"
```

ändert sich das Feld folgendermaßen:

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

02C0  00 C1 02 15 00 01 06 00 09 A1 01 09 A1 01 09 A1 ................
02D0  01 15 2B 9C 09 A1 01 09 A1 01 00 00 00 00 00 00 ..+.............
```

Der Stringdescriptor von a$(3) -der vierte im Feld, da die Null mitzählt- zeigt auf den neuen zusammengefügten String an der Adresse 9C2B, außerdem hat er auch die neue Länge des Strings übernommen.

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

9C20  00 00 00 00 00 00 00 00 00 00 00 49 72 67 65 6E ...........Irgen
9C30  64 77 61 73 20 75 6E 64 20 6E 6F 63 68 77 61 73 dwas und nochwas
```

Soviel zu den Strings, wenden wir uns nun den Fließkomma- und Integervariablen zu. Das gleiche, wie im folgenden für dimensionierte Arrays gezeigte gilt auch für einfache Variable.

Wieder zuerst ein kleines Beispielprogramm:

```
10 DIM a(5)
20 FOR n=0 TO 5:a(n)=10^n:NEXT
```

Es wird also ein Fließkommafeld angelegt, das folgendermaßen aussieht:

```
Adr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF

01F0  00 00 C1 04 21 00 01 06 00 00 00 00 00 81 00 00 ....!...........
0200  00 20 84 00 00 00 48 87 00 00 00 7A 8A 00 00 40 . ....H....z...@
0210  1C 8E 00 00 50 43 91 43 91 00 00 00 00 00 00 00 ....PC.C........
```

An der Speicherstelle Ø1F9 (@a(Ø)) beginnt unser Feld, und wir sehen, daß hier die Fließkommazahlen mit ihren 5 Bytes direkt abgelegt sind, also keine Zeiger mehr.

Ähnlich sieht es auch bei einem Integerfeld aus nach den Basic-Zeilen:

```
1Ø DEFINT a: DIM a(5)
2Ø FOR n=Ø TO 5:a(n)=2Ø48+2^n:NEXT
```

Das entstandene Feld ab der Adresse Ø2ØØ (@a(Ø)):

```
Adr.  ØØ Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8 Ø9 ØA ØB ØC ØD ØE ØF Ø123456789ABCDEF

Ø2ØØ  Ø1 Ø8 Ø2 Ø8 Ø4 Ø8 Ø8 Ø8 1Ø Ø8 2Ø Ø8 Ø8 ØØ 48 87 .......... ...H.
```

Es gilt dasselbe wie bei dem Fließkommafeld Gesagte, nur daß es sich bei den Integerzahlen um Werte zwischen Ø und 65535 handelt und demzufolge diese Zahlen mit zwei Bytes auskommen.

Nach dieser ganzen Theorie um Variable kommen wir zum Grund dieser Ausführungen, dem CALL-Befehl. Vielleicht wissen Sie, daß der CALL-Befehl des Schneider-Basics 32 Parameter mitnehmen kann, das können Integerzahlen sowie die Variablenpointer der String- und Fließkommavariablen sein. Der Befehl könnte deshalb wie folgt lauten:

```
CALL adresse,a%,b%,@c$,@d$,.........,123,65535,@e(Ø)
```

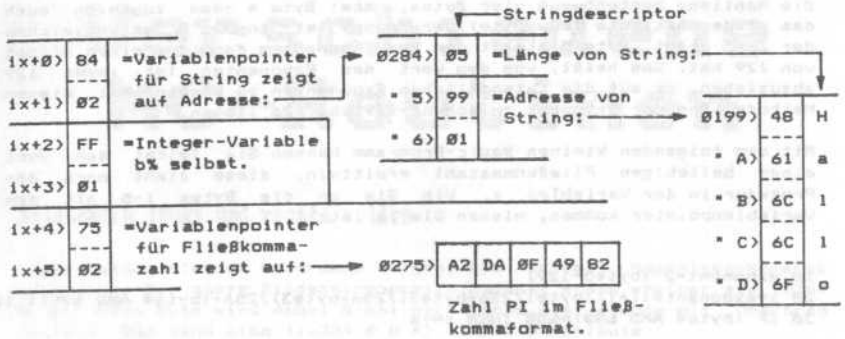
Integerzahlen können auch direkt übermittelt, String- und Fließkommavariable müssen über ihre Variablenpointer verwendet werden, da nur Zwei-Byte lange Werte berücksichtigt werden. Im anschließenden Maschinenspracheteil zeigt das IX-Register auf den letzten Wert in der Kolonne und der Accu enthält die Anzahl der Parameter. Bis hierhin war also der Basic-Interpreter zuständig, alles weitere muß nun Ihr Maschinenspracheteil erledigen.

Aber gehen wir wieder von einem konkreten Beispiel aus:

```
1Ø a=PI
2Ø b%=511
3Ø a$="Hallo"
4Ø CALL adresse,@a,b%,@a$
5Ø 'oder
6Ø !BEFEHL,@a,b%,@a$
```

Der CALL-Befehl nimmt drei Parameter mit, die Variable b% und die Pointer auf die Variablen a und a$. Übrigens kann ein eingebundener Basic-Befehl die gleiche Aufgabe übernehmen. Wie man einen solchen neuen Befehl kreieren kann, zeige ich später.

Nach Aufruf des CALL- oder neuen Basicbefehls sieht die Registerwelt der Z-80-CPU im Schneider so aus, wie auf der nächsten Seite gezeigt wird:

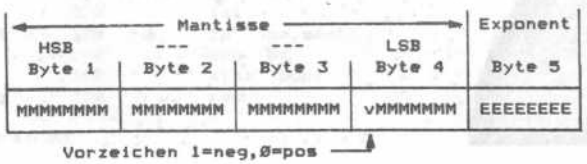


Ich glaube die Darstellung erklärt sich selber. Ein typisches Maschinenprogramm könnte so beginnen:

```
CP   3            ;sind auch wirklich 3 Parameter
RET  NZ           ;vorhanden, wenn nicht zurück zu Basic

LD   H,(IX+1)     ;Adresse des Strindescriptors nach HL
LD   L,(IX+Ø)     ;das erste Byte des Descrip.=Länge a$
LD   A,(HL)       ;Länge in Accu

OR   A            ;Zurück wenn Länge = Ø, also kein
RET  Z            ;String da

INC  HL           ;HL+1, zeigt nun auf Lowbyte Stringadr.
LD   E,(HL)       ;Lowbyte Stringadresse nach E
INC  HL           ;HL+1, zeigt nun auf Highbyte Stringadr.
LD   D,(HL)       ;Highbyte Stringadresse nach D

LD   B,(IX+3)     ;Highbyte von Integerzahl nach B
LD   C,(IX+2)     ;Lowbyte   "      "      "  C

LD   H,(IX+5)     ;Highbyte von Adr.der Fließkommaz.nach H
LD   L,(IX+4)     ;Lowbyte   "   "   "      "     "  L
```

Nach dieser Programmsequenz steht die Länge von a$ im Accu, die Adresse des ersten Zeichens von a$ in DE, die Integerzahl a% in BC und die Adresse des ersten Bytes der Fließkommavariablen in HL.

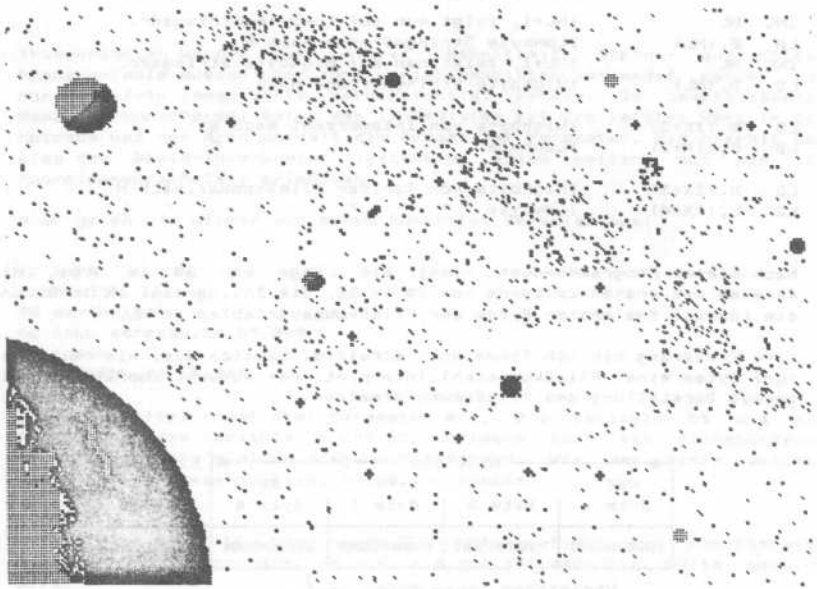
Eine Erklärung bin ich Ihnen noch schuldig, nämlich die, wie Sie die fünf Bytes einer Fließkommazahl interpretieren müssen. Zuerst mal die genaue Darstellung des Fließkommaformates:



Die Mantisse besteht aus vier Bytes, wobei Byte 4 -das zugleich auch das niederwertigste (Low-Byte) der Gruppe ist- in Bit 8 das Vorzeichen der Zahl trägt. Byte 5 stellt den 2er-Exponenten dar, der einen Offset von 129 hat. Das heißt, von dem Wert des Exponenten ist immer 129 abzuziehen, um auf die tatsächlichen Exponenten zu kommen. Mit dieser Methode kann der Exponent zwischen -129 und 126 liegen.

Mit dem folgenden kleinen Basic-Programm können Sie leicht den Wert einer beliebigen Fließkommazahl ermitteln, diese steht nach der Prozedur in der Variablen x. Wie Sie an die Bytes 1-5 mit dem Variablenpointer kommen, wissen Sie ja jetzt!

```
10 exponent=2^(byte5-129)
20 x=exponent*(1+(((byte1/256+byte2)/256+byte3)/256+(byte4 AND &7F))/128)
30 IF (byte4 AND &80)=&80 THEN x=-x
```



Verschiedene MC-Routinen

Bildschirm links und rechts rollen

Diese beiden kleinen Programme zeigen, wie schnell Maschinen-Sprache sein kann. Der ganze Bildschirmbereich umfasst etwas weniger als 16384 Byte. Jedes Byte wird dabei 8-mal nach links oder rechts mit Übertrag rotiert. Das sind also (16384 * 8 =) 131072 Durchläufe !
Eine kleine Besonderheit sind dabei die Verwendungen der LDI und LDD-Befehle, die hier nur zur Zeigerverwaltung dienen.
Der Aufruf ist ganz einfach:

```
CALL &4E20      oder      CALL &4E37
```

```
Pass 1 errors: 00

                    10 ;Den ganzen Bildschirm
                    20 ;rollen.
                    30 ;
                    40 ;Rechts-Scroll
4E20                50          org  20000
4E20                60          ent  20000
                    70 ;
4E20  0608          80          ld   b,8
4E22  C5            90 loop:    push bc
4E23  2100C0       100          ld   hl,#c000
4E26  1100C0       110          ld   de,#c000
4E29  010040       120          ld   bc,#4000
4E2C  CB1E         130 next:    rr   (hl)
4E2E  EDA0         140          ldi
4E30  EA2C4E       150          jp   pe,next
4E33  C1           160          pop  bc
4E34  10EC         170          djnz loop
4E36  C9           180          ret
                   190 ;
                   200 ;Links-Scroll
4E37  0608         210          ld   b,8
4E39  C5           220 loop1:   push bc
4E3A  21FFFF       230          ld   hl,#ffff
4E3D  11FFFF       240          ld   de,#ffff
4E40  010040       250          ld   bc,#4000
4E43  CB16         260 next1:   rl   (hl)
4E45  EDA8         270          ldd
4E47  EA434E       280          jp   pe,next1
4E4A  C1           290          pop  bc
4E4B  10EC         300          djnz loop1
4E4D  C9           310          ret

Pass 2 errors: 00
```

Funktion SCREEN$

Das folgende Programm ruft im wesentlichen nur eine ROM- Routine zum Lesen eines Zeichens an Kursorposition vom Bildschirm auf. Diese Zeichen wird an eine Basic-Stringvariable übergeben, sofern sie die Länge von eins hatte.
Eventuell ist das Programm für Sie so nützlich, daß Sie es mit einer RSX-Erweiterung ins Basic einbauen wollen. Wie das geht, zeige ich später noch. Bei der Namensgebung "SCREEN$" ließ ich mich übrigens durch das Sinclair-Basic leiten, bei dem es die selbe sehr feine Funktion gibt.

Noch ist der Aufruf allerdings nur:

```
CALL 20000,@a$
```

Zuvor den Kursor an die entsprechende Stelle bringen !

```
Pass 1 errors: 00
```

```
                    10 ;Funktion SCREEN$
                    20 ;lesen eines Zeichens vom
                    30 ;Bildschirm und im String
                    40 ;abspeichern.
                    50 ;
4E20                60         org  20000
                    70 ;Subroutine verwendet nur (AF).
4E20  CD60BB        80         call #bb60
4E23  DD6601        90         ld   h,(ix+1)
4E26  DD6E00       100         ld   l,(ix+0)
4E29  46           110         ld   b,(hl)
4E2A  05           120         dec  b
4E2B  C0           130         ret  nz
4E2C  23           140         inc  hl
4E2D  5E           150         ld   e,(hl)
4E2E  23           160         inc  hl
4E2F  56           170         ld   d,(hl)
4E30  12           180         ld   (de),a
4E31  C9           190         ret
```

```
Pass 2 errors: 00
```

Alarmton

Ein Programm, das zehnmal hintereinander einen Piepston von sich gibt, mag zwar auf den ersten Blick ziemlich uninteressant sein, aber man sitzt, während der Computer arbeitet, nicht unentwegt vor dem Bildschirm, dann ist es praktisch, wenn der Computer auf Maschinenebene auf sich aufmerksam machen kann.


```
Pass 1 errors: 00
```

```
4E20                10         org  20000
4E20                20         ent  20000
4E20  060A          30         ld   b,10
4E22  21007D        40 aloop:  ld   hl,32000
4E25  2B            50 bloop:  dec  hl
4E26  7C            60         ld   a,h
4E27  B5            70         or   l
4E28  20FB          80         jr   nz,bloop
4E2A  3E07          90         ld   a,7
4E2C  CD5ABB       100         call #bb5a
4E2F  10F1         110         djnz aloop
4E31  C9           120         ret
```

```
Pass 2 errors: 00
```

Text ausgeben

Dieses Programm soll eigentlich nur das Zusammenwirken einiger Z-80-Befehle mit den ROM-Routinen zeigen. Dabei wird ein beliebiger Text ausgedruckt, einschließlich der Steuerzeichen bis zu einem Textende-Kennzeichen, für das ich das Null-Byte wählte.

```
Aufruf:   CALL 20000
```

```
Pass 1 errors: 00
```

```
                    10 ;Ausdrucken eines Textes auf
                    20 ;Bildschirm an angegebener
                    30 ;Cursorposition.
                    40 ;Byte 0 ist Ende-Kennzeichen
                    50 ;
4E20                60         org  20000
4E20                70         ent  20000
0001                80 spalte: equ  1
000F                90 zeile:  equ  15
                   100 ;
4E20  2601         110         ld   h,spalte
4E22  2E0F         120         ld   l,zeile
4E24  CD75BB       130         call #bb75
                   140 ;
4E27  21334E       150         ld   hl,text
4E2A  7E           160 loop:   ld   a,(hl)
4E2B  B7           170         or   a
4E2C  C8           180         ret  z
4E2D  CD5ABB       190         call #bb5a
4E30  23           200         inc  hl
4E31  18F7         210         jr   loop
4E33  312E205A     220 text:   defm "1. Zeile"
4E3B  0D0A         230         defb #0d,#0a
4E3D  322E205A     240         defm "2. Zeile"
4E45  0D0A00       250         defb #0d,#0a,0
```

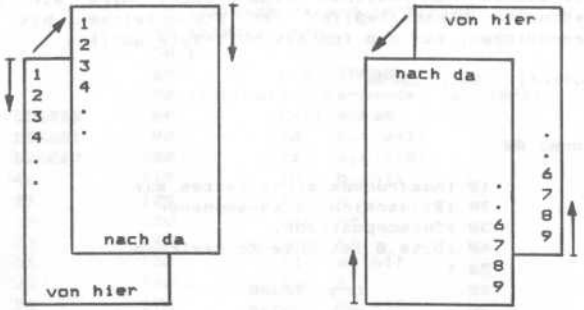
```
Pass 2 errors: 00
```

Die Befehle LDIR und LDDR

Wann immer man Programm- oder Datenbereiche im Speicher transferieren will, wird man zu den leistungsstarken Befehlen LDIR und LDDR des Z-80 greifen. Diese Befehle ersetzen ganze Programmsequenzen auf anderen Prozessoren.
Die Beiden unterscheiden sich in der Abarbeitungsrichtung. LDDR kopiert von hinten nach vorne und LDIR umgekehrt. Wichtig ist dieses verschiedene Verhalten beim Kopieren von sich überlagernden Bereichen. Wenn dabei der falsche Befehl zur Anwendung kommt, wird der Quellbereich vor dem Kopieren überschrieben.

Typische Programmteile und eine kleine grafische Darstellung sollen das verdeutlichen:

```
LD HL,Quellbereich          LD HL,Quellbereich
LD DE,Zielbereich           LD DE,Zielbereich
LD BC,Bereichslänge         LD BC,Bereichslänge
LDIR                        LDDR
RET                         RET
```



ROM lesen

Wenn Sie mal einen Blick in die Schneider-ROM's werfen wollen, dann ist das Programm genau richtig. Mit einer Hilfsroutine, die den RST-Mechanismus der CPC's benützt, um auf die Bytes in den verschiedenen ROM's zugreifen zu können, werden 16k-ROM-Inhalte ins RAM ab 25000 kopiert.

Sehen Sie auch bitte in diesem Zusammenhang in der "Werkzeugkiste" bei "Restarts" nach.

Aufruf des Programms mit:

```
CALL 24950,0,&FC        für unteres ROM
CALL 24950,&C000,&FC    für Basic-ROM
CALL 24950,&C000,&07    für AMSDOS-ROM (wenn da !)
```



```
Pass 1 errors: 00

                    10 ;Beliebiges ROM auslesen
                    20 ;und ab 25000 ins RAM
                    30 ;kopieren.
                    40 ;
6176                50          org  24950
6176  FE02          60          cp   2
6178  C0            70          ret  nz
                    80 ;
6179  DD7E00        90          ld   a,(ix+0)
617C  32BB61        100         ld   (wahl),a
617F  DD6E02        110         ld   l,(ix+2)
6182  DD6603        120         ld   h,(ix+3)
                    130 ;
6185  DF            140         rst  #18
6186  8961          150         defw rom
6188  C9            160         ret
                    170 ;
6189  8C61          180 rom:    defw read
618B  FC            190 wahl:   defb #fc
                    200 ;
618C  010040        210 read:   ld   bc,16384
618F  11A861        220         ld   de,25000
6192  EDB0          230         ldir
6194  C9            240         ret

Pass 2 errors: 00
```

Wandle Dezimal nach Hex

Es ist unter Basic ganz normal, wenn eine Zahleneingabe automatisch Binär im Speicher abgelegt wird, ohne den Interpreter müssen wir aber selber dafür sorgen, daß eine Zahlenfolge in eine dem Computer verständliche Form gebracht wird.
Stellen Sie sich vor, Sie haben irgendwo im Speicher, -evtl. von einer Eingabe her- die Zahlenfolge "54321" in ASCII im Speicher stehen und wollen mit diesem Integerwert weiterarbeiten.
Also müssen Sie diese Zahlenfolge in den entsprechenden Hexadezimalwert "D431" umrechnen, der dann z.B. in ein Doppelregister passt.
Das folgende Konverterprogramm liefert nur richtige Ergebnisse, wenn der Wert zwischen 0 und 65535 liegt.

Das Maschinenprogramm dazu:

```
Pass 1 errors: 00

                    10 ;Wandle ASCII-Zahlenfolge
                    20 ;nach Hexadezimal
                    30 ;
4E20                40          org  20000
4E20                50          ent  20000
                    60 ;
```

```
4E20  21574E      70          ld   hl,hex     ;Loesche Speicher
4E23  3600        80          ld   (hl),0
4E25  23          90          inc  hl
4E26  3600       100          ld   (hl),0
4E28  23         110          inc  hl         ;hl zeigt nun auf
4E29  111027     120          ld   de,10000    ;Anfang der Zahlenfolge.
4E2C  CD444E     130          call wandle     ;Die jeweilige Konstante
4E2F  11E803     140          ld   de,1000    ;laden und konvertieren.
4E32  CD444E     150          call wandle
4E35  116400     160          ld   de,100
4E38  CD444E     170          call wandle
4E3B  110A00     180          ld   de,10
4E3E  CD444E     190          call wandle
4E41  110100     200          ld   de,1       ;nuetze RET aus Upg.
                 210 ;
4E44  7E         220 wandle:  ld   a,(hl)     ;ASCII in Accu.
4E45  D630       230          sub  "0"         ;ASCII Versatz abziehen.
4E47  280C       240          jr   z,noadd    ;Bei 0 keine Addition.
4E49  E5         250          push hl         ;Zeiger aufheben.
4E4A  2A574E     260          ld   hl,(hex)   ;hl ist Arbeitsregister.
4E4D  47         270          ld   b,a        ;Additionszaehler nach b
4E4E  19         280 wloop:   add  hl,de      ;Addieren.
4E4F  10FD       290          djnz wloop
4E51  22574E     300          ld   (hex),hl   ;Im Speicher ablegen.
4E54  E1         310          pop  hl         ;ASCII-Zeiger zurueck
4E55  23         320 noadd:   inc  hl         ;und naechstes Zeichen.
4E56  C9         330          ret
                 340 ;
4E57  0000       350 hex:     defw 0
4E59  35343332   360 zahl:    defm "54321"

Pass 2 errors: 00
```

Wandle Hex nach Dezimal

Wenn Sie mal in die Verlegenheit kommen und einen 16-Bit Binär-Wert dezimal, ohne Vorzeichen -also 0 bis 65535- , darstellen wollen, dann haben Sie jetzt ein Programm dazu. Es hat wenig Sinn, es von Basic aus aufzurufen, denn dann hat das HL-Register ja jedesmal nur 65535 als Inhalt, was hier lediglich als Beispiel gelten soll.

Es wird nicht verraten, wie das Programm arbeitet, da Sie mit Sicherheit selber daraufkommen !

```
Pass 1 errors: 00

                  10 ;Dezimaldarstellung einer
                  20 ;16-Bit Binaerzahl in HL-Reg.
                  30 ;
BB5A              35 print:  equ  #bb5a
4E20              37         org  20000
4E20              38         ent  20000
4E20  21FFFF      40         ld   hl,#ffff
                  45 ;
```



```
4E23  011027      50         ld   bc,10000
4E26  CD3F4E      60         call aus
4E29  01E803      70         ld   bc,1000
4E2C  CD3F4E      80         call aus
4E2F  016400      90         ld   bc,100
4E32  CD3F4E     100         call aus
4E35  010A00     110         ld   bc,10
4E38  CD3F4E     120         call aus
4E3B  7D         130         ld   a,l
4E3C  C34A4E     140         jp   output
                 150 ;
4E3F  AF         160 aus:    xor  a
4E40  5D         170 subtct: ld   e,l
4E41  54         180         ld   d,h
4E42  3C         190         inc  a
4E43  ED42       200         sbc  hl,bc
4E45  30F9       210         jr   nc,subtct
4E47  3D         220         dec  a
4E48  6B         230         ld   l,e
4E49  62         240         ld   h,d
4E4A  C630       250 output: add  a,"0"
4E4C  CD5ABB     260         call print
4E4F  C9         270         ret

Pass 2 errors: 00
```

Hexdump

Dieses erste umfangreichere Programm kann sehr hilfreich sein, wenn Sie mal neugierig sind und dem Rechner auf's Byte sehen wollen. Dieses Programm erzeugt nach dem Aufruf einen Speicher-Dump, wie ihn folgender Musterausdruck zeigt.

```
Adr.    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F     0123456789ABCDEF

9C40    21 0C 9D FE 03 20 0F DD 7E 04 B7 28 09 CB C6 3E     !.... ..~..(...>
9C50    3C 32 0D 9D 18 13 CB 86 3E 14 32 0D 9D CD 11 BC     <2......>.2.....
9C60    FE 02 28 05 3E 02 CD 0E BC CD 12 9D DD 6E 00 DD     ..(.>........n..
9C70    66 01 06 04 CB 3C CB 1D 10 FA 4D 44 DD 6E 02 DD     f....<....MD.n..
9C80    66 03 7D E6 F0 6F E5 7C CD C8 9C 7D CD C8 9C CD     f.}..o.|...}....
9C90    E3 9C C5 06 10 7E CD C8 9C 3E 20 CD F9 9C 23 10     .....~...> ...#.
9CA0    F4 E5 D1 CD E3 9C C1 E1 C5 06 10 7E CD EF 9C 23     ...........~...#
9CB0    10 F9 D5 E1 CD 42 9D C1 0B 78 B1 C8 CD 09 BB FE     .....B...x......
```

Der Aufruf für die Bildschirmausgabe lautet dabei:

```
        CALL 40000,0,Anfangsadresse,Byteanzahl
oder    CALL 40000,Anfangsadresse,Byteanzahl
```

Und für die Druckerausgabe:

```
        CALL 40000,1,Anfangsadresse,Byteanzahl
```

Die erste Zahl ist die Startadresse des Programms, die zweite ist das Druckereinschaltbyte, und die letzten beiden Angaben beziehen sich auf den Speicherbereich, den Sie ansehen möchten.

Das Programm dazu:

```
Pass 1 errors: 00
```

```
                        10 ;       HEXDUMP
                        20 ;SPEICHER IN HEX UND ASCII
                        30 ;     AUSDRUCKEN.
                        40 ;
BC11                    50 GETMOD: EQU  #BC11
BC0E                    60 SETMOD: EQU  #BC0E
BB5A                    70 PRINT:  EQU  #BB5A
BD2B                    80 DRUCK:  EQU  #BD2B
BB09                    90 READCH: EQU  #BB09
BB06                   100 WAITCH: EQU  #BB06
                       110 ;
9C40                   120         ORG  40000
9C40  210C9D           130         LD   HL,DRFLAG
9C43  FE03             140         CP   3          ;Bei weniger als 3 Argte.
9C45  200F             150         JR   NZ,BILDSC  ;ist Bildschirm Default.
9C47  DD7E04           160         LD   A,(IX+4)
9C4A  B7               170         OR   A          ;Bei 3 Argumenten prüfe
9C4B  2809             180         JR   Z,BILDSC   ;Akku auf 0 od.<>0
                       190 ;
9C4D  CBC6             200         SET  0,(HL)     ;Akku war <>0, Drucker-
9C4F  3E3C             210         LD   A,60       ;flag=1,Zeilenzähler =
9C51  320D9D           220         LD   (ZEILE),A  ;60 (pro Seite).
9C54  1813             230         JR   LAB1
                       240 ;
9C56  CB86             250 BILDSC: RES  0,(HL)
9C58  3E14             260         LD   A,20       ;Akku war 0,Druckerflag=0,
9C5A  320D9D           270         LD   (ZEILE),A  ;20 Zeilen für Bildsch.
                       280 ;
9C5D  CD11BC           290         CALL GETMOD
9C60  FE02             300         CP   2
9C62  2805             310         JR   Z,LAB1     ;Bei Bildschirmwahl diesen
9C64  3E02             320         LD   A,2        ;auf Mode 2 setzen wenn
9C66  CD0EBC           330         CALL SETMOD     ;noch nicht.
                       340 ;
9C69  CD129D           350 LAB1:   CALL TITEL      ;Kopfzeile ausgeben.
9C6C  DD6E00           360         LD   L,(IX+0)   ;Anzahl der auszugebenden
9C6F  DD6601           370         LD   H,(IX+1)   ;Bytes holen.Geteilt durch
9C72  0604             380         LD   B,4        ;16 =Zeilenzahl.
9C74  CB3C             390 TEIL16: SRL  H
9C76  CB1D             400         RR   L          ;Anstatt ROM-Integerruot.
9C78  10FA             410         DJNZ TEIL16     ;4* rechtschieben (=/16).
9C7A  4D               420         LD   C,L        ;Zeilenzähler nach BC
9C7B  44               430         LD   B,H        ;bringen.
                       440 ;
9C7C  DD6E02           450         LD   L,(IX+2)   ;Anfangsadresse holen und
9C7F  DD6603           460         LD   H,(IX+3)   ;immer auf volle 16 Byte
9C82  7D               470         LD   A,L        ;abrunden.
9C83  E6F0             480         AND  #F0
9C85  6F               490         LD   L,A
```

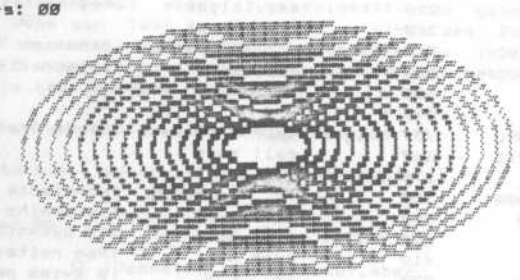


```
                       500 ;
                       510 ;* HAUPTSCHLEIFE *
                       520 ;
9C86  E5               530 WEITER: PUSH HL         ;Anfangsadr retten.
9C87  7C               540         LD   A,H
9C88  CDC89C           550         CALL HEXOUT     ;High-Byte Adr.ausgeben.
9C8B  7D               560         LD   A,L
9C8C  CDC89C           570         CALL HEXOUT     ;Low-Byte Adr.ausgeben.
9C8F  CDE39C           580         CALL SPACE      ;4*Space ausgeben.
9C92  C5               590         PUSH BC         ;Zeilenzähler retten
                       600 ;
9C93  0610             610         LD   B,16       ;16 Byte als Hexcode
9C95  7E               620 LOOP1:  LD   A,(HL)     ;ausgeben mit einem
9C96  CDC89C           630         CALL HEXOUT     ;Zwischenraum.
9C99  3E20             640         LD   A," "
9C9B  CDF99C           650         CALL OUTPUT
9C9E  23               660         INC  HL
9C9F  10F4             670         DJNZ LOOP1
                       680 ;
9CA1  E5               690         PUSH HL         ;Die um 16 erhöhte Adr.
9CA2  D1               700         POP  DE         ;für später nach DE.
                       710 ;
9CA3  CDE39C           720         CALL SPACE      ;4 * Space
9CA6  C1               730         POP  BC         ;BC-Reg ist im Weg !
9CA7  E1               740         POP  HL         ;alten HL-Wert holen.
9CA8  C5               750         PUSH BC         ;BC wieder zurück.
                       760 ;
9CA9  0610             770         LD   B,16       ;Die gleichen 16 Byte
9CAB  7E               780 LOOP2:  LD   A,(HL)     ;nun als ASCII-Werte
9CAC  CDEF9C           790         CALL ASCII      ;ausgeben.
9CAF  23               800         INC  HL
9CB0  10F9             810         DJNZ LOOP2
                       820 ;
9CB2  D5               830         PUSH DE         ;Jetzt den neuen "Anfg."
9CB3  E1               840         POP  HL         ;wieder nach HL.
                       850 ;
9CB4  CD429D           860         CALL LFCR       ;Zeilenschaltung
                       870 ;FERTIG ?
9CB7  C1               880         POP  BC
9CB8  0B               890         DEC  BC         ;Zeilenzähler-1
9CB9  78               900         LD   A,B
9CBA  B1               910         OR   C          ;Wenn 0 dann zum
9CBB  C8               920         RET  Z          ;Basic.
                       930 ;TEST AUF BREAK
9CBC  CD09BB           940         CALL READCH     ;Weiter geht's,aber
9CBF  FEFC             950         CP   #FC        ;zuerst Test auf Break.
9CC1  C8               960         RET  Z
                       970 ;
9CC2  CD4D9D           980         CALL SEITE      ;Prüfe ob Seitenumbruch
9CC5  C3869C           990         JP   WEITER     ;oder Pause notwendig,
                      1000 ; ENDE HAUPTSCHLEIFE    ;sonst weiter.
                      1010 ;
9CC8  F5              1020 HEXOUT: PUSH AF
9CC9  0F              1030         RRCA
9CCA  0F              1040         RRCA            ;Klassischer Hexkonverter
9CCB  0F              1050         RRCA            ;mit schieben, ausblenden
9CCC  0F              1060         RRCA            ;und ASCII dazuadieren.
9CCD  CDD59C          1070         CALL HEX2
```

```
9CD0  F1        1080          POP  AF
9CD1  CDD59C    1090          CALL HEX2
9CD4  C9        1100          RET
9CD5  E60F      1110 HEX2:    AND  #0F
9CD7  C630      1120          ADD  A,"0"
9CD9  FE3A      1130          CP   "9"+1
9CDB  3802      1140          JR   C,ZAHL
9CDD  C607      1150          ADD  A,7
9CDF  CDF99C    1160 ZAHL:    CALL OUTPUT    ;High od. Low-Nibble ausg.
9CE2  C9        1170          RET
                1180 ;
9CE3  C5        1190 SPACE:   PUSH BC        ;Unterprogramm für
9CE4  0604      1200          LD   B,4       ;4* Space ausgeben.
9CE6  3E20      1210          LD   A," "
9CE8  CDF99C    1220 LOOP3:   CALL OUTPUT
9CEB  10FB      1230          DJNZ LOOP3
9CED  C1        1240          POP  BC
9CEE  C9        1250          RET
                1260 ;
9CEF  FE7F      1270 ASCII:   CP   #7F       ;Nicht in ASCII druck-
9CF1  3004      1280          JR   NC,NOASC  ;bare Zahlenwerte durch
9CF3  FE20      1290          CP   " "       ;"." ersetzen.
9CF5  3002      1300          JR   NC,OUTPUT
9CF7  3E2E      1310 NOASC:   LD   A,"."
                1320 ;
9CF9  E5        1330 OUTPUT:  PUSH HL        ;Zeichen ausgeben,ent-
9CFA  210C9D    1340          LD   HL,DRFLAG ;weder auf Bildschirm
9CFD  CB46      1350          BIT  0,(HL)    ;oder auf Drucker
9CFF  E1        1360          POP  HL
9D00  2004      1370          JR   NZ,DRUCK2
9D02  CD5ABB    1380          CALL PRINT
9D05  C9        1390          RET
9D06  CD2BBD    1400 DRUCK2:  CALL DRUCK
9D09  30FB      1410          JR   NC,DRUCK2
9D0B  C9        1420          RET
                1430 ;
9D0C  00        1440 DRFLAG:  DEFB 0
9D0D  00        1450 ZEILE:   DEFB 0
9D0E  4164722E  1460 ADR:     DEFM "Adr."
                1470 ;
9D12  210E9D    1480 TITEL:   LD   HL,ADR     ;*Titelzeile ausgeben.
9D15  0604      1490          LD   B,4       ;Zuerst "Adr." printen.
9D17  7E        1500 LOOP4:   LD   A,(HL)
9D18  CDF99C    1510          CALL OUTPUT
9D1B  23        1520          INC  HL
9D1C  10F9      1530          DJNZ LOOP4
                1540 ;
9D1E  CDE39C    1550          CALL SPACE     ;4 * Space
                1560 ;
9D21  0610      1570          LD   B,16
9D23  AF        1580          XOR  A         ;dann 16 mal 2-stellige
9D24  F5        1590 LOOP5:   PUSH AF        ;Hexzahlen ausgeben.
9D25  CDC89C    1600          CALL HEXOUT
9D28  3E20      1610          LD   A," "
9D2A  CDF99C    1620          CALL OUTPUT
9D2D  F1        1630          POP  AF
9D2E  3C        1640          INC  A
9D2F  10F3      1650          DJNZ LOOP5
```



```
                1660 ;
9D31  CDE39C    1670          CALL SPACE     ;dann 4 * Space
                1680 ;
9D34  0610      1690          LD   B,16
9D36  AF        1700          XOR  A         ;16 mal einstellige Hex-
9D37  F5        1710 LOOP6:   PUSH AF        ;zahlen ausgeben.
9D38  CDD59C    1720          CALL HEX2
9D3B  F1        1730          POP  AF
9D3C  3C        1740          INC  A
9D3D  10F8      1750          DJNZ LOOP6
                1760 ;
9D3F  CD429D    1770          CALL LFCR      ;2* Linefeed ausg. ( Ret
                1780 ;                       ;von Unterprg. nützen).
9D42  3E0D      1790 LFCR:    LD   A,#0D
9D44  CDF99C    1800          CALL OUTPUT    ;LineFeed-Schaltung
9D47  3E0A      1810          LD   A,#0A
9D49  CDF99C    1820          CALL OUTPUT
9D4C  C9        1830          RET
                1840 ;
9D4D  C5        1850 SEITE:   PUSH BC
9D4E  D5        1860          PUSH DE        ;Reg. retten.
9D4F  E5        1870          PUSH HL
9D50  210D9D    1880          LD   HL,ZEILE
9D53  3A0C9D    1890          LD   A,(DRFLAG);Zeilenzähler nach HL
9D56  CB47      1900          BIT  0,A       ;Prüfe ob Drucker od.
9D58  200A      1910          JR   NZ,DRCKER ;Bildschirm
                1920 ;
9D5A  35        1930          DEC  (HL)      ;Nach 20 Zeilen ist
9D5B  2014      1940          JR   NZ,ENDSEI ;Warten notwendig um
9D5D  CD06BB    1950          CALL WAITCH    ;lesen zu können.
9D60  3614      1960          LD   (HL),#14  ;Zähler wieder 20.
9D62  180D      1970          JR   ENDSEI
                1980 ;
9D64  35        1990 DRCKER:  DEC  (HL)
9D65  200A      2000          JR   NZ,ENDSEI ;Nach 60 Zeilen auf
9D67  363C      2010          LD   (HL),60   ;Papier ist Seitenvor-
9D69  3E0C      2020          LD   A,#0C     ;schub notwendig
9D6B  CD069D    2030          CALL DRUCK2    ;und neue Ausgabe der
9D6E  CD129D    2040          CALL TITEL     ;Titelzeile.
                2050 ;
9D71  E1        2060 ENDSEI:  POP  HL
9D72  D1        2070          POP  DE        ;Reg. Zurück.
9D73  C1        2080          POP  BC
9D74  C9        2090          RET

Pass 2 errors: 00
```



Hardcopy für MODE 2

Das nun folgende Programm druckt den Bildschirminhalt nur in Mode 2 aus und soll haupsächlich einige Programmier-Methoden zeigen wie z.B. den direkten Zugriff auf den Bildschirm-Speicher, die Umsetzung der "horizontalen Bildschirmbytes" in "vertikale Druckerbytes", die Verschachtelung der Programmschleifen und den möglichen Aufbau eines solchen Programms im Ganzen.

Aufruf einfach mit CALL 40000.

```
Pass 1 errors: 00

                        10 ;Hardcopyprogramm nur
                        20 ;fuer Bildschirmmodus 2 !
                        25 ;+8-Bit-Centronics !!
                        30 ;
BD2B                    40 toprnt: equ  #bd2b
BD2E                    50 busy:   equ  #bd2e
BC11                    60 getmod: equ  #bc11
BB1E                    70 tstkey: equ  #bb1e
C000                    80 bildan: equ  #c000
                        90 ;
9C40                   100         org  40000
9C40  CD11BC           110         call getmod     ;Bei MODE <>2 zurück.
9C43  FE02             120         cp   2
9C45  C0               130         ret  nz
9C46  11AD9C           140         ld   de,zeiesc  ;Sende Zeilenabstand.
9C49  CD9F9C           150         call send
                       160 ;
9C4C  0619             170         ld   b,25       ;25 Zeilen und &C000
9C4E  2100C0           180         ld   hl,bildan  ;ist Bildanfang (ohne
                       190 ;                       ;Scroll !!!!).
9C51  C5               200 haupt:  push bc
9C52  E5               210         push hl
9C53  CD6B9C           220         call outgra     ;Sende Grafikbytes.
9C56  3E42             230         ld   a,66       ;Teste Escape Taste.
9C58  CD1EBB           240         call tstkey
9C5B  E1               250         pop  hl         ;Hole Register zurück.
9C5C  C1               260         pop  bc
9C5D  C0               270         ret  nz         ;Zum Basic wenn Escape.
9C5E  115000           280         ld   de,80      ;Addiere neuen Zeilen-
9C61  19               290         add  hl,de      ;beginn.
9C62  10ED             300         djnz haupt
9C64  11B19C           310         ld   de,retesc  ;Sende am Schluß Drucker-
9C67  CD9F9C           320         call send       ;einschaltstellung.
9C6A  C9               330         ret
                       340 ;
9C6B  11A89C           350 outgra: ld   de,graesc  ;Sende Grafikvorspann.
9C6E  CD9F9C           360         call send
                       370 ;
9C71  110008           380         ld   de,#0800   ;Konstante für nächste
9C74  0650             390         ld   b,80       ;Bytereihe einer Zeile,
9C76  C5               400 zeile:  push bc         ;80 Bytes/Zeile.
9C77  E5               410         push hl         ;Reg retten.
9C78  0E08             420         ld   c,8        ;8 Bytes pro Zeichenplatz
```



```
9C7A  E1               430 byte:   pop  hl         ;untereinander.
9C7B  E5               440         push hl         ;Stack = Zwischenspeicher
9C7C  0608             450         ld   b,8        ;n-Bits/Byte rotieren.
9C7E  CB06             460 bits:   rlc  (hl)       ;rotiere (HL) und sammle
9C80  17               470         rla             ;Carry's im Akku.
9C81  19               480         add  hl,de      ;Addiere Konstante
9C82  10FA             490         djnz bits       ;       nächste Reihe.
9C84  CD969C           500         call output     ;Gib Akku auf Drucker.
9C87  0D               510         dec  c
9C88  20F0             520         jr   nz,byte
9C8A  E1               530         pop  hl         ;Register zurück und
9C8B  23               540         inc  hl         ;nächsten Zeichenplatz
9C8C  C1               550         pop  bc         ;lesen wenn noch in Zeile.
9C8D  10E7             560         djnz zeile
9C8F  11B49C           570         ld   de,lfcr    ;Sende LineFeed und
9C92  CD9F9C           580         call send       ;Carriage Return.
9C95  C9               610         ret
                       620 ;
9C96  CD2EBD           630 output: call busy       ;Warte bis Drucker auf-
9C99  38FB             640         jr   c,output   ;nahmefähig und sende
9C9B  CD2BBD           650         call toprnt     ;dann.
9C9E  C9               660         ret
                       670 ;
9C9F  1A               680 send:   ld   a,(de)     ;Hilfsprogramm zum
9CA0  B7               690         or   a          ;Senden diverser
9CA1  C8               700         ret  z          ;Druckersteuersequenzen.
9CA2  CD969C           710         call output
9CA5  13               720         inc  de
9CA6  18F7             730         jr   send
9CA8  1B4C8002         740 graesc: defb 27,"L",#80,#02,0
9CAD  1B4108000        750 zeiesc: defb 27,"A",#08,0
9CB1  1B3200           760 retesc: defb 27,"2",0
9CB4  0A0D00           770 lfcr:   defb 10,13,0

Pass 2 errors: 00
```

Schnelle Text-Hardcopy

Dieses Programm ist die schnellste Hardcopy in diesem Buch, kann zwar nur ASCII-Zeichen aber -und das ist was Neues- auch alle Sonderzeichen zu Papier bringen, nur keine Grafik. Die Routine testet ein zu druckendes Zeichen auf seine Zugehörigkeit zum genormten ASCII-Zeichensatz. Wenn der Test positiv war, wird das Zeichen normal ausgedruckt, ansonsten wird die Matrix des Zeichens im RAM und ROM gesucht, der Drucker kurzzeitig in den Grafikmode gebracht und die Originalmatrix des Zeichens geprinted.

Aufruf nur mit: CALL 41500

Routine funktioniert unter allen Bildschirmmodis !

```
Pass 1 errors: 00

                        10 ;Schnelle Texthardcopy,aber
                        20 ;mit Schneider-Sonderzeichen
```

```
                        30 ;
BC11                    40 getmod: equ  #bc11
BB78                    50 getcur: equ  #bb78
BB75                    60 curpos: equ  #bb75
BB1E                    70 tstkey: equ  #bb1e
BB60                    80 readch: equ  #bb60
BBA5                    90 getmat: equ  #bba5
BD2B                   100 print:  equ  #bd2b
BD2E                   110 busy:   equ  #bd2e
                       120 ;
A21C                   130         org  41500
A21C  CD78BB           140         call getcur     ;Kursorposition merken.
A21F  E5               150         push hl
                       160 ;
A220  CD11BC           170         call getmod     ;Im Akku nun 0,1,2 (Mode).
A223  3C               180         inc  a          ;Accu plus 1
A224  47               190         ld   b,a        ;Als Multzähler verwenden.
A225  3E0A             200         ld   a,10       ;Additionskonstante
A227  87               210 mult:   add  a,a        ;Nach Durchlauf im Akku
A228  10FD             220         djnz mult       ;20,40 od.80 (Zeich./Mode)
                       230 ;
A22A  47               240         ld   b,a        ;Neuer Zähler je nach Mode
A22B  0E19             250         ld   c,25       ;25 Zeilen und bei
A22D  2E01             260         ld   l,1        ;Spalte 1 beginnen.
                       270 ;
                       280 ; Hauptschleife
A22F  C5               290 haupt:  push bc         ;Spalte und Zeile retten
A230  2600             300         ld   h,0        ;Zeile 0 (gibt es nicht).
                       310 ;
A232  24               320 zeile:  inc  h          ;Zeile +1.
A233  E5               330         push hl
A234  CD75BB           340         call curpos     ;Kursor auf Spalte/Zeile.
A237  E1               350         pop  hl
A238  CD60BB           360         call readch     ;Zeichen an Kur.Pos. lesen.
A23B  3802             370         jr   c,okzei    ;Im Zeichensatz vorhanden?
A23D  3E20             380         ld   a," "      ;Wenn nicht dann Space.
A23F  CD61A2           390 okzei:  call output     ;Zeichen ausgeben.
A242  10EE             400         djnz zeile      ;Eine Zeile bearbeiten.
                       410 ;
A244  3E0A             420         ld   a,10       ;Zeilenschaltung.
A246  CDBEA2           430         call toprnt
A249  3E0D             440         ld   a,13
A24B  CDBEA2           450         call toprnt
A24E  E5               460         push hl
A24F  3E42             470         ld   a,66       ;Esc. gedrückt ?
A251  CD1EBB           480         call tstkey     ;Reg. zurück
A254  E1               490         pop  hl
A255  C1               500         pop  bc
A256  2004             510         jr   nz,escape  ;Aus wenn Esc.
                       520 ;
A258  2C               530         inc  l          ;nächste Zeile
A259  0D               540         dec  c          ;Zeilenzähler-1
A25A  20D3             550         jr   nz,haupt   ;Ganzer Bildschirm.
                       560 ;
A25C  E1               570 escape: pop  hl         ;Alte Kursorposition
A25D  CD75BB           580         call curpos     ;holen und setzen.
A260  C9               590         ret
                       600 ; Ende Hauptschleife
```



```
                       610 ;
A261  E5               620 output: push hl
A262  C5               630         push bc         ;Reg. aufheben.
A263  CB7F             640         bit  7,a
A265  200A             650         jr   nz,noasc   ;Zeichen >127 dann Grafik.
A267  FE20             660         cp   " "
A269  3806             670         jr   c,noasc    ;Zeichen <32 dann Grafik.
                       680 ;
A26B  CDBEA2           690         call toprnt     ;Normales ASCII-Zeichen
A26E  C1               700         pop  bc         ;ausgeben
A26F  E1               710         pop  hl         ;Reg. zurück.
A270  C9               720         ret
                       730 ;
A271  CDA5BB           740 noasc:  call getmat     ;Matrix von Schneider-
A274  3005             750         jr   nc,rom     ;Sonderzeichen holen
A276  CD83A2           760         call chcopy     ;aus ROM oder RAM.
A279  1819             770         jr   outgra
                       780 ;
A27B  DF               790 rom:    rst  #18        ;Matrix aus ROM holen.
A27C  80A2             800         defw rerom
A27E  1814             810         jr   outgra
                       820 ;
A280  83A2             830 rerom:  defw chcopy
A282  FC               840         defb #fc
                       850 ;
A283  118CA2           860 chcopy: ld   de,buffer  ;Hilfsprogramm zum
A286  010800           870         ld   bc,8       ;Matrix in RAM kopieren.
A289  EDB0             880         ldir
A28B  C9               890         ret
A28C                   900 buffer: defs 8
                       910 ;
A294  3E1B             920 outgra: ld   a,27       ;Esc;"K";8;0 senden.
A296  CDBEA2           930         call toprnt
A299  3E4B             940         ld   a,"K"
A29B  CDBEA2           950         call toprnt
A29E  3E08             960         ld   a,8
A2A0  CDBEA2           970         call toprnt
A2A3  AF               980         xor  a
A2A4  CDBEA2           990         call toprnt
A2A7  0608            1000         ld   b,8
                      1010 ;
A2A9  C5              1020 byte:   push bc         ;Dieses Programm macht aus
A2AA  218CA2          1030         ld   hl,buffer ;"horizontalen"  Zeichen-
A2AD  0608            1040         ld   b,8       ;bytes "vertikale"Drucker-
                      1050 ;                      ;bytes.
A2AF  CB06            1060 bits:   rlc  (hl)
A2B1  17              1070         rla
A2B2  23              1080         inc  hl
A2B3  10FA            1090         djnz bits
                      1100 ;
A2B5  CDBEA2          1110         call toprnt
A2B8  C1              1120         pop  bc
A2B9  10EE            1130         djnz byte
                      1140 ;
A2BB  C1              1150         pop  bc
A2BC  E1              1160         pop  hl
A2BD  C9              1170         ret
                      1180 ;
```

```
A2BE  CD2EBD    1190 toprnt: call busy      ;Wenn Drucker bereit
A2C1  38FB      1200         jr   c,toprnt  ;dann senden.
A2C3  CD2BBD    1210         call print
A2C6  C9        1220         ret

Pass 2 errors: 00
```

BEISPIELAUSDRUCK

```
F m ^ F & 8 ^ ▮ 4 ! Ä ^ 6 ◤ ▬ ↓ ó ↖ ? ▮
← M ↖ ↖ * e H ↖ ; ¢ , ● # G £ ▮ ❯ ▟ ß 6
l ┐ h ♠ p " ! ¢ X ▟ ◀ 9 0 9 n ▮ ◢ r ◢
✲ " ▫ £ ■ T ❮ + ↑ ◢ ƒ ▶ ▮ t 8 ▟ ◘ ◤ D ∝
Ready
call 41500
```

Mini - Monitor

Ein nicht zu unterschätzendes Hilfsmittel zum Testen von mehr oder weniger kleinen Maschinen - Routinen. Das Basicprogramm versteht sich als Grundinstrument und kann jederzeit zu einem umfangreicheren Monitor erweitert werden, evtl. mit dem Hex-Dump-Programm aus diesem Buch.

Die eigentliche Besonderheit ist das dazugehörige Maschinen- Programm, welches eine Registerübernahme von und zum Basic erlaubt. Dabei wird etliche Male der sonst so sorgsam gehütete Stack des Prozessors gezielt ausgetrickst. Wenn ein Register nach einem PUSH in ein anderes zwecks Tausch gePOPt wird, ist das ja noch harmlos. Wenn aber noch mit den Rücksprungadressen gespielt wird, dann wird's ernst.

Erinnern wir uns, ein RET holt vom Stapel eine Adresse und springt zu ihr, diese Adresse wurde vorher durch einen CALL- Befehl auf dem Stapel abgelegt. Soweit, sogut - aber hier ist es anders, die Adresse wird nicht durch einen CALL erzeugt, sondern durch ein offenes PUSH HL ! Deshalb dürfen wir in Zeile 560 auch nicht mit CALL springen -denn dann wäre ja wieder eine neuerer Stapeleintrag erfolgt-, sondern mit RET ! Was in diesem Fall einem CALL gleichkommt. In diesem Programm wird davon gleich zweimal Gebrauch gemacht und zwar sowohl für den Sprung zum Anwenderprogramm nach dem ganzen Register-Laden als auch bei der Rückkehr von diesem zum Programm-Teil, der die Registerinhalte wieder in Basicvariable packt.

Die PUSHes in den Zeilen 130 und 160 sind dafür offen, d.h. sie haben kein gleichwertiges POP im Programm.

Der Aufruf erfolgt mit

```
CALL 40000,Startadr.Anwender,@Register.....
```

Und nun zuerst das Basic-Programm und dann das Maschinen - Programm.


```
10 '****** Mini-Monitor ******
20 '****************************
30 DEFINT a-z:CLS:MEMORY 29999
40 LOAD"minimon.bin",40000
45 'Data-Zeilen mit Ihrem Programm
50 DATA cd,f0,bb,47,e6,0f,fe,02,c9
60 DATA 10^0
70 '
80 adresse=30000
90 WHILE a$<>"1000"
100 READ a$:wert=VAL("&"+a$)
110 POKE adresse,(wert AND 255)
120 adresse=adresse+1
130 WEND
140 POKE adresse,&C9' Noch'n RET -- sicher ist sicher!
150 '
160 'Register mit beliebigen Werten laden.
170 '-------------------------------------
180 a=&FF:f=&0:af$=HEX$(256*a+f,4):af=VAL("&"+af$)
190 bc=&FFFF:de=&0:hl=&0:iy=&0
200 '
210 'Hilfsroutinenaufruf
220 '-------------------
230 CALL 40000,30000,$af,$bc,$de,$hl,$iy
240 '
250 'Ausdruck
260 '--------
270 PRINT"           P"
280 PRINT"           /  "
290 PRINT"      SZ H VNC"
300 PRINT"Flag= "¦BIN$(af AND 255,8)
310 PRINT
320 a$=LEFT$(HEX$(af,4),2)
330 PRINT"Akku=   "¦a$¦" hex",USING "####### dez."¦VAL("&"+a$)
340 PRINT
350 PRINT"BC  = "¦HEX$(bc,4)¦ " hex",USING "####### dez."¦bc
360 PRINT"DE  = "¦HEX$(de,4)¦ " hex",USING "####### dez."¦de
370 PRINT"HL  = "¦HEX$(hl,4)¦ " hex",USING "####### dez."¦hl
380 PRINT"IY  = "¦HEX$(iy,4)¦ " hex",USING "####### dez."¦iy
390 END

Pass 1 errors: 00
```

```
                    10 ¦Hilfsprogramm zum
                    20 ¦Austesten diverser
                    30 ¦Maschinenroutinen
                    40 ¦mit Registeruebergabe
                    50 ¦von und zum BASIC.
                    60 ¦
9C40                70          org  40000
9C40  FE06          80          cp   6
9C42  C0            90          ret  nz
                   100 ¦
                   110 ¦Stack manipulieren!
9C43  21849C       120          ld   hl,rueck
9C46  E5           130          push hl
9C47  DD6E0A       140          ld   l,(ix+10)
9C4A  DD660B       150          ld   h,(ix+11)
9C4D  E5           160          push hl
```

```
                        170 ;
                        180 ;iy-reg laden
9C4E  DD6E00            190         ld   l,(ix+0)
9C51  DD6601            200         ld   h,(ix+1)
9C54  5E                210         ld   e,(hl)
9C55  23                220         inc  hl
9C56  56                230         ld   d,(hl)
9C57  D5                240         push de
9C58  FDE1              250         pop  iy
                        260 ;bc-reg laden
9C5A  DD6E06            270         ld   l,(ix+6)
9C5D  DD6607            280         ld   h,(ix+7)
9C60  4E                290         ld   c,(hl)
9C61  23                300         inc  hl
9C62  46                310         ld   b,(hl)
                        320 ;af-reg laden
9C63  DD6E08            330         ld   l,(ix+8)
9C66  DD6609            340         ld   h,(ix+9)
9C69  5E                350         ld   e,(hl)
9C6A  23                360         inc  hl
9C6B  56                370         ld   d,(hl)
9C6C  D5                380         push de
9C6D  F1                390         pop  af
                        400 ;de-reg laden
9C6E  DD6E04            410         ld   l,(ix+4)
9C71  DD6605            420         ld   h,(ix+5)
9C74  5E                430         ld   e,(hl)
9C75  23                440         inc  hl
9C76  56                450         ld   d,(hl)
9C77  D5                460         push de
                        470 ;hl-reg laden
9C78  DD6E02            480         ld   l,(ix+2)
9C7B  DD6603            490         ld   h,(ix+3)
9C7E  5E                500         ld   e,(hl)
9C7F  23                510         inc  hl
9C80  56                520         ld   d,(hl)
9C81  EB                530         ex   de,hl
9C82  D1                540         pop  de
                        550 ;getrickstes Return ausfuehren
9C83  C9                560         ret
                        570 ;
                        580 ;durch 2.Stackmanipulation
                        590 ;fuehrte ein ret hierher!
                        600 ;
                        610 ;de nach variable
9C84  E5                620 rueck:  push hl
9C85  DD6E04            630         ld   l,(ix+4)
9C88  DD6605            640         ld   h,(ix+5)
9C8B  73                650         ld   (hl),e
9C8C  23                660         inc  hl
9C8D  72                670         ld   (hl),d
                        680 ;hl nach variable
9C8E  D1                690         pop  de
9C8F  DD6E02            700         ld   l,(ix+2)
9C92  DD6603            710         ld   h,(ix+3)
9C95  73                720         ld   (hl),e
9C96  23                730         inc  hl
9C97  72                740         ld   (hl),d
```



```
                        750 ;iy nach variable
9C98  DD6E00            760         ld   l,(ix+0)
9C9B  DD6601            770         ld   h,(ix+1)
9C9E  FDE5              780         push iy
9CA0  D1                790         pop  de
9CA1  73                800         ld   (hl),e
9CA2  23                810         inc  hl
9CA3  72                820         ld   (hl),d
                        830 ;bc nach variable
9CA4  DD6E06            840         ld   l,(ix+6)
9CA7  DD6607            850         ld   h,(ix+7)
9CAA  71                860         ld   (hl),c
9CAB  23                870         inc  hl
9CAC  70                880         ld   (hl),b
                        890 ;af nach variable
9CAD  DD6E08            900         ld   l,(ix+8)
9CB0  DD6609            910         ld   h,(ix+9)
9CB3  F5                920         push af
9CB4  D1                930         pop  de
9CB5  73                940         ld   (hl),e
9CB6  23                950         inc  hl
9CB7  72                960         ld   (hl),d
9CB8  C9                970         ret

Pass 2 errors: 00
```

Data - Generator

Sie kennen sicher das Problem: Sie wollen ein Maschinen - Programm handlich verpackt einem Bekannten zugänglich machen, ohne daß dieser umständliche Ladeprozeduren in Kauf nehmen muß !
Diesem Zweck dient das folgende Programm, welches aus einem Maschinen-Code wieder Basic-Data-Zeilen erzeugt. Viel ist zu diesem Programm nicht zu sagen, nur soviel, daß die Data - Zeilen mit einem kleinen Lader als ASCII-File abgespeichert werden.
Das ist auch der Grund, warum Sie keine Schwierigkeiten mit dem MERGEn in ein bestehendes Basic-Programm haben werden. Die älteren CPC-464 haben ja noch einen Fehler in der MERGE-Routine (Die Token eines Basic-Programms können z.T. als EOF - Bytes interpretiert werden.), der hier aber keine Rolle mehr spielt.

```
100 '     DATA - GENERATOR
110 '-------------------------
120 INPUT"Filename (ohne Extension)";file$
130 IF LEN(file$)>8 THEN PRINT CHR$(7):GOTO 120
140 PRINT
150 INPUT"Anfangsadresse eingeben : ";anf
160 INPUT"Endadresse     eingeben : ";ende
170 IF ende<anf THEN PRINT CHR$(7):GOTO 150
180 '
190 'Basic-File erzeugen
200 '--------------------
```

```
210 OPENOUT file$+".bas"
220 PRINT#9,"10 FOR n="¦anf¦" TO "¦ende
230 PRINT#9,"20 READ a$:wert=VAL(CHR$(38)+a$)"
240 PRINT#9,"30 POKE n,wert:NEXT n"
250 PRINT#9,"40 '"
260 '
270 'DATA-Zeilen erzeugen
280 '--------------------
290 zeile=100:PRINT#9,zeile¦" DATA "¦
300 zaehler=1
310 FOR n=anf TO ende
320 a$=HEX$(PEEK(n),2)
330 IF INT (zaehler/16)<>(zaehler/16) THEN PRINT#9,a$;","; ELSE PRINT#
9,a$:IF n<>ende THEN zeile=zeile+10:PRINT#9,zeile¦" DATA "¦
340 zaehler=zaehler+1
350 NEXT
360 PRINT#9
370 CLOSEOUT
```

Breakpoint

Wenn Sie mal ein Maschinen-Programm geschrieben haben, das auch nach der dritten Nachtschicht einfach nicht laufen will und sich trotz aller Bemühungen mit einem bunten Bildschirm verabschiedet, ist es Zeit für eine Durchforstung nach einem Fehler mittels einer Breakpoint-Verarbeitung.
Nur, so einfach ist das nicht, denn auch in einem funktionierenden Programm ist der Stack nicht zu jeder Zeit ausgeglichen. Also ist ihre rettende Rücksprungadresse zum Basic mehr oder weniger verschüttet.

Da gibt es aber einen relativ einfachen Weg. Wenn der Basic-Interpreter zu einer Anwenderroutine springt (Dabei ist es egal, ob ein CALL- oder RSX-Befehl dies auslöste.), dann ist der oberste Eintrag auf dem Stapel die Rücksprungadresse zur Interpreter-Schleife. Also den Stackpointer, der ja einen Zeiger auf diesen Eintrag bildet, VOR der eigentlichen aufzurufenden Routine in einer Speicherzelle ablegen. Dies übernimmt der Befehl LD (speich),SP - wie im anschließenden Programm aufgeführt.
Wenn Sie nun irgendwo in Ihrem Programm einen CALL zu einer Routine setzen, der den Stapelzeiger wieder auf diesen Eintrag setzt, kommen Sie mit einem anschließenden RETurn wieder ins Basic.

Aber erstens ist es nicht so praktisch, jedesmal den 3-Byte Befehl eines CALLs in die Routine zu packen, und zweitens wollen Sie ja sicher auch die Registerinhalte nach diesem Rücksprung sehen. Es hätte ja keinen Sinn, wieder im Basic-Programm zu landen, ohne zu wissen, weshalb das Programm nicht läuft, und um dies zu erkunden sind die Registerinhalte und vor allem die Flag's sehr wichtig.
Dem ersten Problem begegnen wir durch einen RESTART- Befehl, der wie ein CALL wirkt, aber nur ein Byte verbraucht. Dieser RST-6-Befehl bewirkt zwar einen Sprung zur Adresse #38, aber ein einfacher "JP breakp" bringt uns zur eigenlichen Routine.

Jetzt sind wir endlich im eigentlichen Breakpoint-Programm und gehen an das zweite Problem.

Wenn wir schon den Stapel manipulieren, dann auch gleich richtig ! Zuerst koppeln wir mit DI das Betriebssystem ab. Ab jetzt sind Sie alleiniger Herr und Meister über den Stapel des Rechners !
Dann legen wir mit einem neuen Stapel-Zeiger einen Benutzer-Stapel an, PUSHen die ganzen Register darauf, lassen den Stackpointer auf die Rücksprungadresse zum Basic zeigen, lassen auch das Betriebssytem wieder zugreifen nach dem EI-Befehl und springen zu guter Letzt mit dem anschließendem RET ins Basic. Fertig !

Die Registerinhalte können Sie nun aus dem angewiesenen Pufferbereich von Basic mit PEEK auslesen.

Aber nicht vergessen, vor jedem zu untersuchenden Programm zuerst den Stackpointer retten !

Das Programm:

```
Pass 1 errors: 00

                        10 ;Breakpoint
                        20 ;----------
A604                    30          org  42500
A604  F3                40 breakp:  di
A605  3124A6            50          ld   sp,puffer+12
A608  F5                60          push af
A609  C5                70          push bc
A60A  D5                80          push de
A60B  E5                90          push hl
A60C  DDE5             100          push ix
A60E  FDE5             110          push iy
A610  ED7B16A6         120          ld   sp,(speich)
A614  FB               130          ei
A615  C9               140          ret
A616  0000             150 speich: defw 0
A618  00000000         160 puffer: defw 0,0,0,0,0,0
                       170 ;-----------------------
                       180 ;An Adr. #30 poken !
A624  C304A6           190          jp   breakp
                       200 ;-----------------------
                       210 ;Vor jedes zu testende
                       220 ;Programm setzten !
A627  ED7316A6         230          ld   (speich),sp
                       240 ;-----------------------
                       250 ;Als Aufruf dann :
A62B  F7               260          rst  #30

Pass 2 errors: 00
```

Vier Routinen zur Listenverarbeitung

Programm 1:

Es wird untersucht, ob ein im Akku angegebenes Byte in einer Liste enthalten ist. Der starke Befehlssatz des Z-80 macht es möglich, daß das ganze Suchen mit einem einzigen Befehl erledigt werden kann. Wenn

das Byte in der Liste war, ist nach Beendigung des CPIR-Befehls das Zerro-Flag 1 sonst 0.

Programm 2:

Gibt an, wie oft ein im Akku angegebenes Byte in einer Liste enthalten ist. Nach Aussprung enthält das DE-Register die Anzahl der gefundenen Einträge.
Eine Abart des CPIR- Befehles, der CPI- Befehl besorgt jetzt die Hauptaufgabe. Da dieser Befehl sich nicht automatisch wiederholt, kann er auch in einer Schleife erweitert werden. Dafür muß aber dann auch von außen gesorgt werden, daß sich der Befehl wiederholt, dies geschieht mit einem "JP-Parity-Even-Befehl". Es gibt leider keinen relativen Sprung mit dieser Bedingung.

Programm 3:

Es kann folgendermaßen umschrieben werden: Prüfe eine Liste mit vier Einträgen, ob ein bestimmtes Byte -wieder im Akku- vorhanden ist, merke dir den Platz des Eintrages, hole dann aus einer Tabelle mit Sprungadressen die entsprechende und führe dann ein Anwenderprogramm aus.
In diesem Zusammenhang ist die Methode interessant, wie aus dem gefundenen Listeneintrag eine Adresse berechnet wird, die wiederum auf einen Tabelleneintrag zeigt.

Programm 4:

Ähnlich wie Programm 3 nur ohne Listenverarbeitung. Im Akku steht lediglich eine Nummer eines bestimmten auszuführenden Jumps.

****

Alle vier Programme dienen nur zur Anregung, wie man auf ein Programmierproblem mit einer Maschinen-Code-Routine antworten kann.

****

Anschließend die vier Teillösungen:

```
Pass 1 errors: 00

7530            10         org  30000
0014            20 laenge: equ  20
                30 ;
                40 ; Programm 1
                50 ;-----------
7530  011400      60         ld   bc,laenge
7533  219075      70         ld   hl,liste
7536  EDB1        80         cpir
7538  C9          90         ret
                100 ;
```



```
                110 ; Programm 2
                120 ;-----------
7539  110000    130         ld   de,0
753C  011400    140         ld   bc,laenge
753F  219075    150         ld   hl,liste
7542  EDA1      160 pruef:  cpi
7544  2001      170         jr   nz,nofoun
7546  13        180         inc  de
7547  EA4275    190 nofoun: jp   pe,pruef
754A  C9        200         ret
                210 ;
                220 ; Programm 3
                230 ;-----------
754B  0604      240         ld   b,4
754D  219075    250         ld   hl,liste
7550  BE        260 weiter: cp   (hl)
7551  2804      270         jr   z,found
7553  23        280         inc  hl
7554  10FA      290         djnz weiter
7556  C9        300         ret
                310 ;
7557  05        320 found:  dec  b
7558  68        330         ld   l,b
7559  2600      340         ld   h,0
755B  29        350         add  hl,hl
755C  116575    360         ld   de,tabell
755F  19        370         add  hl,de
7560  5E        380         ld   e,(hl)
7561  23        390         inc  hl
7562  56        400         ld   d,(hl)
7563  EB        410         ex   de,hl
7564  E9        420         jp   (hl)
7565  8D758A75  430 tabell: defw num4,num3
7569  87758475  440         defw num2,num1
                450 ;
                460 ; Programm 4
                470 ;-----------
756D  57        480         ld   d,a
756E  87        490         add  a,a
756F  82        500         add  a,d
7570  5F        510         ld   e,a
7571  1600      520         ld   d,0
7573  217875    530         ld   hl,jumps
7576  19        540         add  hl,de
7577  E9        550         jp   (hl)
7578  C38475    560 jumps:  jp   num1
757B  C38775    570         jp   num2
757E  C38A75    580         jp   num3
7581  C38D75    590         jp   num4
                600 ;
                610 ; Nur fuer Beispielzwecke
                620 ;-----------------------
7584  3E41      630 num1:   ld   a,"A"
7586  C9        640         ret
7587  3E42      650 num2:   ld   a,"B"
7589  C9        660         ret
758A  3E43      670 num3:   ld   a,"C"
758C  C9        680         ret
```

```
758D  3E44      690 num4:   ld   a,"D"
758F  C9        700         ret
                710 ;
7590            720 liste:  defs laenge
75A4            730         end

Pass 2 errors: 00
```

Deutscher Zeichensatz

Selbstverständlich kann man beim Schneider ohne besondere Umschweife den Zeichensatz per Basic mit SYMBOL AFTER... ändern. Aber dann ist es auch manchmal nötig, einen nicht unerheblichen Teil des Zeichensatzes ins RAM zu kopieren, das frißt aber erstens sehr viele Bytes und zweitens ist es nicht immer möglich.
Einen anderen Lösungsweg möchte ich hier aufzeigen. Es wird durch ein Programm geprüft, ob ein bestimmtes zum Ausdruck kommendes Zeichen in einer Austauschliste vorhanden ist, trifft dies zu dann wird dem Akku statt des Originalzeichens der ASCII-Code eines neuen Charakters übergeben. Da Zeichen über 240 automatisch im RAM stehen, können diese Zeichen sehr praktisch als Austauschcharaktere dienen.

Das Programm nützt wieder eine Eigenschaft des Schneiders, der bei manchen Routinen zuerst in der Tabelle der indirekten Sprünge nachsieht ob der Anwender nicht vielleicht etwas anderes von ihm erwartet als das Betriebssystem vorsieht. So wird der indirekte Sprung auf eine eigene Routine "verbogen", die das Auswechseln der Zeichen vornimmt.

```
Pass 1 errors: 00

                 10 ;Deutschen Zeichensatz
                 20 ;ueber indirekten Sprung
                 30 ;einpatchen.
                 40 ;
A028             50         org  41000
A028  2ADABD     60         ld   hl,(#bdd9+1)
A02B  2245A0     70         ld   (raus+1),hl
A02E  2135A0     80         ld   hl,routin
A031  22DABD     90         ld   (#bdd9+1),hl
A034  C9        100         ret
                110 ;
                111 ; Ab jetzt laufen alle
                112 ;    Zeichenausgaben
                113 ; ueber diese Routine.
                115 ;
A035  0608      120 routin: ld   b,ende-liste/2
A037  2147A0    130         ld   hl,liste
A03A  BE        140 loop:   cp   (hl)
A03B  23        150         inc  hl
A03C  2805      160         jr   z,found
A03E  23        170         inc  hl
A03F  10F9      180         djnz loop
A041  1801      190         jr   raus
A043  7E        200 found:  ld   a,(hl)
A044  C30C14    210 raus:   jp   #140c
```



```
A047  40F85BF9  220 liste:  defb "@",248,"[",249,"\",250
A04D  5DFB7BFC  230         defb "]",251,"{",252,"!",253
A053  7DFEA3FF  240         defb "}",254,"#",255
A057            250 ende:   end

Pass 2 errors: 00
```

Das Basicprogramm setzt die deutschen Umlaute in den Symbolspeicher ab Charakter 248.

```
10 'Umlaute in den letzten Benutzer-Zeichen
20 SYMBOL 248,30,48,56,108,56,48,240,0
30 SYMBOL 249,102,0,60,102,102,126,102,0
40 SYMBOL 250,198,0,124,198,198,108,56
50 SYMBOL 251,102,0,102,102,102,102,60,0
60 SYMBOL 252,198,0,120,12,124,204,118,0
70 SYMBOL 253,102,0,60,102,102,102,60,0
80 SYMBOL 254,0,102,0,102,102,102,62,0
90 SYMBOL 255,120,198,198,252,198,198,248,192
100 :
110 KEY DEF 17,1,123,91:KEY DEF 19,1,125,93:KEY DEF 24,1,94,126
```

Drucker-Protokoll

Mit diesem Programm wird das, unter CP/M, sehr praktische Kommando "Control-P" auch unter Basic verfügbar. Dieser Befehl schaltet ja bekanntlich den Drucker parallel zur Bildschirmausgabe.
Mit CALL 41000 wird -wie üblich- eine zusätzliche Routine, mit Hilfe der indirekten Sprungtabelle, eingeschleift. Ein CALL &BB51 stellt wieder normale Verhältnisse her.

Folgendes ist bei dieser Routine zu beachten:
Die Basic-Variable "WIDTH", sie bestimmt die Zeilenlänge eines Ausdrucks, wird nur unter Basic richtig verwaltet,
Da wir hier aber nicht auf dem üblichen Wege ein Zeichen an den Drucker übermitteln, muß "WIDTH" extra berücksichtigt werden. Weshalb in diesem Programm auch ein eigener Zeichenzähler eingebaut wurde.
Mit dem Kommando WIDTH 20,40 oder 80 können sie nun sogar die Zeilenbreite, am Ausdruck, dem Bildschirmmodus angleichen.

Bei den CPC's 664 und 6128 wird die Variable "WIDTH" an der Speicherzelle &AC09 abgelegt !!

```
Pass 1 errors: 00

                 10 ;Zusaetzliche Druckerausgabe
                 20 ;ueber indirekten Sprung
                 30 ;einpatchen.
                 40 ;
AC24             45 width:   equ  #ac24
A028             50          org  41000
A028  2ADABD     60          ld   hl,(#bdd9+1)
A02B  223DA0     70          ld   (screen+1),hl
```

```
A02E  213BA0      80          ld   hl,routin
A031  22DABD      90          ld   (#bdd9+1),hl
A034  3EFF        100 restor: ld   a,255
A036  323AA0      110         ld   (zaehl),a
A039  C9          120         ret
A03A  00          130 zaehl:  defb 0
                  140 ;
                  150 ; Ab jetzt laufen alle
                  160 ;    Zeichenausgaben
                  170 ; ueber diese Routine.
                  180 ;
A03B  F5          190 routin: push af
A03C  CD0C14      200 screen: call #140c
A03F  F1          210         pop  af
A040  FE0D        220         cp   #0d
A042  4F          230         ld   c,a
A043  CC34A0      240         call z,restor
A046  79          250         ld   a,c
A047  FE14        255         cp   20
A049  3818        260         jr   c,print
                  265 ;
A04B  F5          270         push af
A04C  213AA0      280         ld   hl,zaehl
A04F  34          290         inc  (hl)
A050  3A24AC      300         ld   a,(width)
A053  BE          310         cp   (hl)
A054  200C        320         jr   nz,inline
A056  3601        330         ld   (hl),1
A058  3E0D        340         ld   a,#0d
A05A  CD63A0      350         call print
A05D  3E0A        360         ld   a,#0a
A05F  CD63A0      370         call print
A062  F1          380 inline: pop  af
                  385 ;
A063  CD2BBD      390 print:  call #bd2b
A066  D8          400         ret  c
A067  18FA        410         jr   print
                  415 ;
                  460 ;Reset Ind.Jump
A069  CD51BB      470         call #bb51
A06C  C9          480         ret

Pass 2 errors: 00
```

Text suchen

Folgendes Programm simuliert, unter Maschinen-Sprache, den Basic-Befehl "INSTR". Es sucht in einem max.64 kByte langen Text, nach einer max. 255 Byte langen Zeichenkette.
Das HL-Register dient in diesem Programm als Zeiger auf den Anfang, des zu untersuchenden Textes und BC enthält die Länge des Textfeldes. DE und B (nach PUSH) zeigen etwas später auf den Anfang des Suchtextes und dessen Länge.
Wenn die Zeichenkette im Text enthalten war, erfolgt der Aussprung mit gesetztem Carry und das HL-Register zeigt auf den Anfang des gesuchten Textteiles (in diesem Beispiel auf &4E52), sonst ist Carry=0.


```
Pass 1 errors: 00
                  10 ;Suchen einer Bytefolge
                  20 ;im Speicher.
                  30 ;
4E20              40          org  20000
4E20  21454E      50          ld   hl,text
4E23  ED4B824E    60          ld   bc,(texlen)
                  70 ;
4E27  C5          80 weiter:  push bc
4E28  E5          90          push hl
4E29  11844E      100         ld   de,sutext
4E2C  3A884E      110         ld   a,(txtlen)
4E2F  47          120         ld   b,a
4E30  1A          130 search: ld   a,(de)
4E31  BE          140         cp   (hl)
4E32  2008        150         jr   nz,unglch
4E34  23          160         inc  hl
4E35  13          170         inc  de
4E36  10F8        180         djnz search
                  190 ;Kommt hier nur an, wenn
                  200 ;alle n-Zeichen gleich waren.
                  210 ;Dann Carry=1 und HL zeigt auf Text.
4E38  E1          220         pop  hl
4E39  C1          230         pop  bc
4E3A  37          240         scf
4E3B  C9          250         ret
                  260 ;
                  270 ;Ein Zeichen war ungleich Suchtext.
4E3C  E1          280 unglch: pop  hl
4E3D  23          290         inc  hl
4E3E  C1          300         pop  bc
                  310 ;Pruefe ob Ende des Textes erreicht,
                  320 ;dann zurueck mit Carry=0.
4E3F  0B          325         dec  bc
4E40  78          330         ld   a,b
4E41  B1          340         or   c
4E42  C8          350         ret  z
4E43  18E2        360         jr   weiter
                  370 ;
4E45  44696573    380 text:   defm "Dies ist ein Test,um eine Zeichen"
4E66  666F6C67    390         defm "folge im Speicher zu suchen."
4E82  3D00        400 texlen: defw $-text
4E84  54657374    410 sutext: defm "Test"
4E88  04          420 txtlen: defb $-sutext&#ff

Pass 2 errors: 00
```

Relocater

Viele in diesem Buch enthaltenen Maschinenprogramme sind sicher in einem Speicherbereich abgelegt, in dem Sie sie nicht gebrauchen können.

Da hilft der Relocater. Er re-lokalisiert den Code, d.h er gibt ihm einen neuen Aufenthaltsort im Speicher. Diese etwas zeitraubende

Vorgehensweise ist nur bei Programmen nötig, die direkte Adressen verwenden, dies trifft aber ohnehin in den meisten Fällen zu.

Sie haben z.B. ein Programm, das ab #8000 abgelegt ist, und Sie wollen es lauffähig zur Adresse #9C40 bringen. Dazu müssen Sie nun alle direkten Sprungadressen und Speicherzeiger auf die entsprechenden Stellen im neuen Bereich zeigen lassen.

Wie gehen Sie nun vor ?

Sie suchen nun die Adressen aller Maschinencodes, in denen solche Zugriffe auftauchen und legen sie in der DATA-Zeile ab. Aber Vorsicht, nicht die Adresse des Operationscodes selbst, sondern erst die seiner Daten. Also wenn z.B. bei Adresse #9D02 der Code "CD 00 82" steht, dann geben Sie die Adresse #9D03 an, ab der in diesem Fall die Sprungadresse steht.

Anschließend geben Sie noch die Menge der Daten in Zeile 35 an. Schließlich geben Sie nun die jetzige Adresse in der Variablen "origadr" an und Ihren gewünschten Bereich in der Variablen "useradr", das Umrechnen aller Adressen geschieht jetzt automatisch nach Start des Hilfs-Programmes.

Nach dieser Prozedur steht das Programm zwar noch immer an der gleiche Stelle, ist aber jetzt nur in dem neuen Speicherbereich lauffähig. Sie müssen es nun nur noch in diesen neuen Bereich praktizieren. Was am einfachsten über Abspeichern auf Datenträger und anschließendes Laden in den neuen Speicherbereich geschieht.

```
10 ' Relocater
15 ' ---------
20 origadr=&8000:useradr=&9C40:' zum Beispiel
25 offset=origadr-useradr
30 DATA &9d00,&9d03:'......Ihre Adressen!
35 anzdat=2:'         entsprechend DATA-Anzahl
40 FOR n=1 TO anzdat:READ adr
45 wert=PEEK(adr)+256*PEEK(adr+1):wert=wert-offset
50 hbyte=INT(wert/256):lbyte=wert-256*hbyte
55 POKE adr,lbyte:POKE adr+1,hbyte:NEXT
```

Integer Bubble-Sort

Das erste Sortierprogramm ist für Ein-Byte-Werte gedacht, die im Speicher ab Label "Liste" abgelegt sind.
Zum besseren Verständnis und zur Erinnerung der Vorgang bei Bubble-Sort:
Die Sortierung erledigen zwei ineinander verschachtelte Schleifen, dabei wird die äußere der beiden solange durchlaufen, bis in der inneren Schleife kein Tausch zweier benachbarter Zellen mehr nötig war.

Die innere Schleife prüft bei jedem Durchlauf die ganze Liste. Dabei werden zwei benachbarte Zellen verglichen und wenn nötig (erster- > zweiter Eintrag) gegeneinander getauscht. Wenn während des Durchlaufes auch nur ein Eintrag getauscht wurde, wird ein Tauschflag gesetzt. Dieses Tauschflag "sagt" dann der äußeren Schleife, daß die Liste noch

nicht ganz sortiert war, und es erfolgt ein erneuter Durchlauf. Das also in groben Zügen zum Sortiervorgang und nun zum Programm.

Das BC-Register enthält die Länge der Liste - 1, DE- und HL- Register bilden die Zeiger auf die beiden benachbarten Zellen und werden laufend incrementiert. Der Akku dient vorrübergehend als Tauschflag, dieses Flag wird aber sofort wieder auf dem Stack abgelegt, um es nicht zu zerstören. Der Wert #FF als Flag wird als "Tausch-erfolgt" interpretiert.

Im Programm ist die Länge der Liste mit 1000 dez. angegeben, sie kann jedoch auch andere Längen annehmen.

```
Pass 1 errors: 00

                        10 ;Bubble-Sort einer
                        20 ;Tabelle mit ein-Byte
                        30 ;Eintraegen.
                        40 ;
03E8                    50 laenge: equ  1000
4E20                    60         org  20000
                        70 ;
4E20  01E703            80 weiter: ld   bc,laenge-1
4E23  11494E            90         ld   de,liste
4E26  214A4E           100         ld   hl,liste+1
4E29  AF               110         xor  a
4E2A  F5               120         push af
                       130 ;
                       140 ;vergleiche zwei Zellen
                       150 ;ob Tausch noetig.
4E2B  1A               160 loop:   ld   a,(de)
4E2C  BE               170         cp   (hl)
4E2D  380E             180         jr   c,notau
4E2F  280C             185         jr   z,notau
                       190 ;
                       200 ;es wird getauscht
                       210 ;(hl) <> (de)
4E31  F1               220         pop  af
4E32  3EFF             230         ld   a,255
4E34  F5               240         push af
4E35  C5               250         push bc
4E36  46               260         ld   b,(hl)
4E37  1A               270         ld   a,(de)
4E38  77               280         ld   (hl),a
4E39  EB               290         ex   de,hl
4E3A  70               300         ld   (hl),b
4E3B  EB               310         ex   de,hl
4E3C  C1               320         pop  bc
                       330 ;
4E3D  23               340 notau:  inc  hl
4E3E  13               350         inc  de
4E3F  0B               360         dec  bc
4E40  78               370         ld   a,b
4E41  B1               380         or   c
4E42  20E7             390         jr   nz,loop
                       400 ;arbeite die Liste
                       410 ;einmal ganz durch.
                       420 ;--------------------
```

```
                        430 ;mach solange weiter
                        440 ;bis Tauschflag nicht
                        450 ;mehr gesetzt war.
4E44  F1                460         pop   af
4E45  B7                470         or    a
4E46  20D8              480         jr    nz,weiter
4E48  C9                490         ret
4E49                    500 liste:  defs 1000
5231                    510         end

Pass 2 errors: 00
```

Fließkommazahlen sortieren

Das folgende Programm ist nicht mehr auf Integerzahlen beschränkt und kann direkt auf ein von Basic mittels DIM-Anweisung angelegtes Array zugreifen !
Um gleich Ihre Neugier zu stillen, das Maschinen-Programm braucht für die Sortierung von 100 Fließkommazahlen nur knapp 3 Sekunden.

Es arbeitet vom Prinzip her genau wie das vorige Programm. Deshalb möchte ich auch nur drei Eigenheiten des Programms besprechen.

Erstens, da ein Basic-Array sehr wohl im RAM-Adress-Bereich bis 16k aufgebaut sein kann -also unter einem ROM- und wir eine Fließkomma-Vergleichsroutine im eben diesem ROM aufrufen, müssen wir zuerst die beiden zu untersuchenden Variablen in einen Bereich schaufeln, auf den auch das ROM zugreifen kann. Denn wenn das ROM aktiviert ist, blendet es das darunterliegende RAM aus und kann so nicht mehr auf das Array zugreifen. Natürlich müssen wir die Variablen später wieder zurückbringen, aber nur wenn sie ausgetauscht wurden.

Zweitens wird -um den Stack zu schonen- das Tauschflag im Interupt-Register abgelegt ! Das geht bei den Schneider- Computern, weil sie alle im Interruptmodus 1 arbeiten und so dieses Register nicht verwenden. Der Anwender bekommt also ein Prozessorregister geschenkt.

Drittens wird der eigentliche Tauschvorgang nun elegant mit dem IY-Register plus einem Offset erledigt, da wir es ja jetzt mit einem 5-Byte-Eintrag zu tun haben.

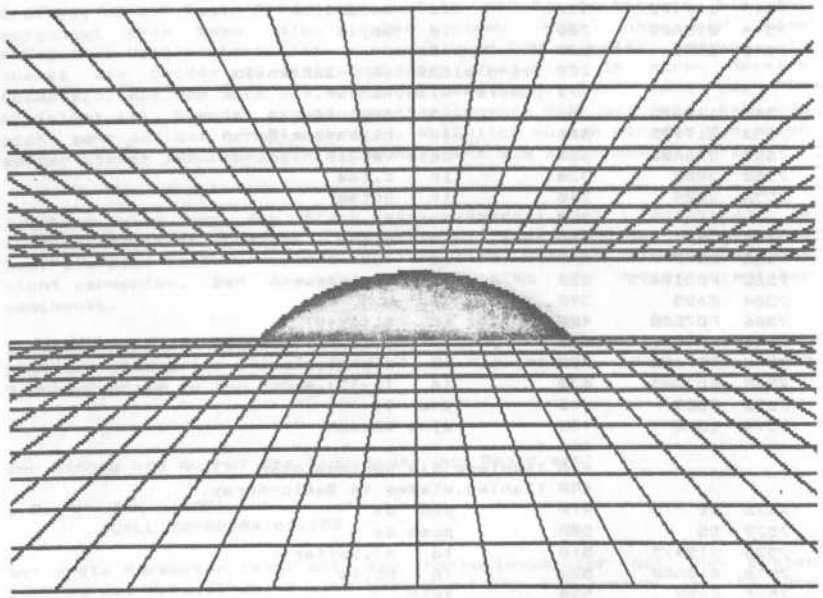
Der Einbau und Aufruf des Programmes von Basic aus:

```
z.B.    DIM a(100)
        CALL 30000,@a(0),100
```

Der erste Parameter zeigt mit dem Variablenpointer auf den ersten Eintrag des Arrays, der zweite gibt die Länge des Arrays an. Da das Programm ja sowieso die Listenlänge abzüglich 1 braucht, muß also nicht als Länge der Wert 101 angegeben werden, welcher ja die tatsächliche Länge des Arrays wäre.


```
Pass 1 errors: 00

                         10 ;Fliesskommazahlen sortieren
                         20 ;muessen in einer Reihe stehen,
                         30 ;so wie es bei einer Basic-DIM-
                         40 ;Anweisung der Fall ist.
                         50 ;
7530                     60         org  30000
BD6A                     70 verglh: equ  #bd6a
                         80 ;
7530  FE02               90         cp   2
7532  C0                100         ret  nz
7533  DD4E00            110         ld   c,(ix+0)
7536  DD4601            120         ld   b,(ix+1)
7539  DD5E02            130         ld   e,(ix+2)
753C  DD5603            140         ld   d,(ix+3)
                        150 ;
                        160 ;fuer Neudurchlauf retten
753F  C5                170 weiter: push bc
7540  D5                180         push de
7541  AF                190         xor  a
7542  ED47              200         ld   i,a
                        210 ;zum Zaehlen auf Stack
7544  C5                220 loop:   push bc
7545  D5                230         push de
                        240 ;kopiere 2 F-Zahlen nach buffer
7546  EB                245         ex   de,hl
7547  119475            250         ld   de,buffer
754A  010A00            260         ld   bc,10
754D  EDB0              270         ldir
                        280 ;vergleiche die F-Zahlen im
                        290 ;Buffer miteinander.
754F  119475            300         ld   de,buffer
7552  219975            310         ld   hl,buffer+5
7555  CD6ABD            320         call verglh
7558  3802              330         jr   c,tau
755A  1824              340         jr   notau
                        350 ;--------------
755C  3EFF              360 tau:    ld   a,255
755E  ED47              370         ld   i,a
7560  FD219475          380         ld   iy,buffer
7564  0605              390         ld   b,5
7566  FD7E00            400 tausch: ld   a,(iy+0)
7569  FD4E05            410         ld   c,(iy+5)
756C  FD7100            420         ld   (iy+0),c
756F  FD7705            430         ld   (iy+5),a
7572  FD23              440         inc  iy
7574  10F0              450         djnz tausch
                        460 ;
                        470 ;kopiere die vertauschten
                        480 ;Zahlen wieder in Basic-Array.
7576  D1                490         pop  de
7577  D5                500         push de
7578  219475            510         ld   hl,buffer
757B  010A00            520         ld   bc,10
757E  EDB0              530         ldir
                        540 ;--------------
7580  D1                550 notau:  pop  de
```

```
7581  C1          560          pop  bc
7582  13          570          inc  de
7583  13          580          inc  de
7584  13          590          inc  de
7585  13          600          inc  de
7586  13          610          inc  de
7587  0B          620          dec  bc
7588  78          630          ld   a,b
7589  B1          640          or   c
758A  20B8        650          jr   nz,loop
                  660 ;sooft mal wiederholen
                  670 ;bis kein Tausch mehr
                  680 ;noetig wurde.
758C  D1          690          pop  de
758D  C1          700          pop  bc
758E  ED57        710          ld   a,i
7590  B7          720          or   a
7591  20AC        730          jr   nz,weiter
7593  C9          740          ret
7594              750 buffer:  defs 10

Pass 2 errors: 00
```



Neue Basic-Befehle

Ihr Computer hat die sehr schöne Eigenschaft, neue Befehle ganz leicht ins bestehende BASIC einbauen zu können, von dieser Möglichkeit machen wir nun Gebrauch. Die Namen der neuen Befehle:

- CIRCLE : Zieht einen Kreis mit verschiedener X-, Y-Ausdehnung, oder auch nur ein Segment davon.
- COPY : Eine Hardcopy mit max. drei Graustufen zuzüglich weiß und schwarz !
- TEXT : Buchstaben und Sonderzeichen so groß wie Sie wollen.
- TAUSCH : Austausch zweier Variablen ohne Garbage- Collektion !
- PAUSE : Ein Computer braucht auch mal eine Pause. Erspart eine leere FOR-NEXT-Schleife.
- DPOKE : Ein 16-Bit-Wort wird in Zilog-Schreibweise abgelegt.
- DPEEK : Das Gegenstück,ein 16-Bit-Wort wird in eine BASIC-Variable geholt.
- RESTORE: Ein Restore mit berechneter Zeilennummer.
- TAST : Wartet ganz einfach auf Tastendruck
- INVERSE: Vertauscht Vorder-und Hintergrundfarbe bis zum erneuten Aufruf.
- OVER1/0: Macht den Hintergrund durchscheinend oder nicht.

Soviel nur um Sie neugierig zu machen. Anschließend wird die ganze Prozedur des Einbindens- und der Aufbau der Befehle sowie deren Syntax erklärt. Fröhliches Abtippen !

Am Anfang die Symboltabelle aller verwendeten ROM-Routinen.

```
Pass 1 errors: 00

0001              10 wahr:   equ  1          ;  ** A C H T U N G **
0000              20 falsch: equ  0          ;
0000              30 sys464: equ  falsch     ;  Diese Maschinen - Pro-
BCD1              40 rsxin:  equ  #bcd1      ;  gramme sind für den CPC
B900              50 basrom: equ  #b900      ;  6128 assembliert.
E85C              60 suchzl: equ  #e85c      ;! Also alle mit ! gekenn-
AE17              70 datazg: equ  #ae17      ;! zeichneten Adressen b.
BD64              80 intflo: equ  #bd64      ;! 464 u. 664 entsprechend
BDAF              90 cos:    equ  #bdaf      ;! ändern.(Siehe Werkzeug-
BDAC             100 sin:    equ  #bdac      ;! kiste.)
BD7C             110 plus:   equ  #bd7c      ;!
                 120 ;----------             ;
BD82             130 minus:  equ  #bd82      ;!
0000             140         if   sys464
53A3             150 newbyt: equ  #37
53A3             160 oldbyt: equ  #3b
53A3             170         else
009A             180 newbyt: equ  #9a
009E             190 oldbyt: equ  #9e
53A3             200         end
                 210 ;----------
```

```
BD94            220 signum: equ  #bd94     ;!BD70
BD97            230 raddeg: equ  #bd97     ;!BD73
BD61            240 copflo: equ  #bd61     ;!BD5D
BD85            250 mult:   equ  #bd85     ;!BD67
BD6A            260 floint: equ  #bd6a     ;!BD46
BD2E            270 busy:   equ  #bd2e
BD2B            280 print:  equ  #bd2b
BBF6            290 draw:   equ  #bbf6
BBC0            300 move:   equ  #bbc0
BBDE            310 setpen: equ  #bbde
BC11            320 getmod: equ  #bc11
BBF0            330 tstpkt: equ  #bbf0
BBE7            340 getpap: equ  #bbe7
BBC6            350 askcur: equ  #bbc6
BBA5            360 getmat: equ  #bba5
BBEA            370 plot:   equ  #bbea
BBDE            380 graset: equ  #bbde
BB09            390 readch: equ  #bb09
BB06            400 waitta: equ  #bb06
BB9C            410 invers: equ  #bb9c
BB9F            420 setgnd: equ  #bb9f
```

Die Erweiterung ist ab Adresse 41300 abgelegt,HIMEM also vor dem Laden auf maximal 41299 setzen.

Zu Beginn (Zeile 480-500) sehen Sie den Teil, der die neuen Basic-Befehle mittels einer ROM-Routine ins System einbindet. Dann folgen die Sprünge zu den Maschinensprache-Teilen mit den BASIC-Namen, das letzte Zeichen eines jeden Namens ist mit #80 oder-verknüpft, und zuletzt ein 4-Byte Speicherbereich für die interne Verwaltung, so wie es der Rechner fordert.

```
                430 #E
A154            440         org  41300
                450 ;
                460 ;Befehle in Basic einbinden.
                470 ;
A154  015DA1    480         ld   bc,jump
A157  21C4A1    490         ld   hl,memory
A15A  C3D1BC    500         jp   rsxin
                510 ;
A15D  83A1      520 jump:   defw namen
A15F  C3C8A1    530         jp   circle
A162  C337A3    540         jp   copy
A165  C364A4    550         jp   text
A168  C35BA5    560         jp   tausch
A16B  C39AA5    570         jp   pause
A16E  C3B4A5    580         jp   dpoke
A171  C3BBA5    590         jp   dpeek
A174  C3D8A5    600         jp   restor
A177  C306BB    610         jp   waitta
A17A  C39CBB    620         jp   invers
A17D  C3E9A5    630         jp   over0
A180  C3EDA5    640         jp   over1
A183  43495243  650 namen:   defm "CIRCL"
A188  C5        660         defb "E"$#80
A189  434F50    670         defm "COP"
A18C  D9        680         defb "Y"$#80
```



```
A18D  544558    690         defm "TEX"
A190  D4        700         defb "T"$#80
A191  54415553  710         defm "TAUSC"
A196  C8        720         defb "H"$#80
A197  50415553  730         defm "PAUS"
A19B  C5        740         defb "E"$#80
A19C  44504F4B  750         defm "DPOK"
A1A0  C5        760         defb "E"$#80
A1A1  44504545  770         defm "DPEE"
A1A5  CB        780         defb "K"$#80
A1A6  52455354  790         defm "RESTOR"
A1AC  C5        800         defb "E"$#80
A1AD  54415354  810         defm "TAST"
A1B1  C5        820         defb "E"$#80
A1B2  494E5645  830         defm "INVERS"
A1B8  C5        840         defb "E"$#80
A1B9  4F564552  850         defm "OVER"
A1BD  B0        860         defb "0"$#80
A1BE  4F564552  870         defm "OVER"
A1C2  B1        880         defb "1"$#80
A1C3  00        890         defb 0
A1C4            900 memory: defs 4
```

Befehl CIRCLE

```
|CIRCLE,(Vonwinkel,Biswinkel),X-Mitte,Y-Mitte,X-Radius,
       Y-Radius,Farbe
```

Zum Beispiel:

```
|CIRCLE,90,270,200,200,150,150,1
```

Dieser Befehl zeichnet ein Kreissegment von Winkelgrad 90-270. Der Mittelpunkt ist dabei in X- und Y-Richtung gleich auch die beiden Radien. Das Segment hat dabei die Farbe 1. Wenn die Angabe Von- und Biswinkel weggelassen wird, wird ein Vollkreis gezeichnet.

Das Programm:

Wenn die Anzahl der Argumente <>7 oder <>5 ist, erfolgt sofort ein Rücksprung nach Basic. Die Parameter werden in die entsprechenden Speicherzellen abgelegt, evtl. erst nach einer Integer zu Fließkommawandlung.
Die Routine kommt in der Hauptschleife ohne die langsamen trigonometrischen Funktionen aus. Am Besten kann man die Arbeitsweise des Programms mit dem folgenden Basic-Algorithmus erklären.

```
110 'Vorgabe der Werte
120 xrad%=150:yrad%=150
130 vonw%=0:bisw%=360
140 xmit%=300:ymit%=200
150 farbe%=1
160 '
170 'Kursor auf Beginn
180 DEG:konst=PI/180
190 x=COS(vonw%):y=SIN(vonw%)
200 GOSUB 340:MOVE xx,yy
210 '
```

```
220 'Hauptschleife
230 FOR n%=1 TO (biswi%-vonw%)/2
240 GOSUB 320:GOSUB 330:GOSUB 340
250 DRAW xx,yy,farbe%
260 GOSUB 330:GOSUB 320:GOSUB 340
270 DRAW xx,yy,farbe%
280 NEXT
290 END
300 '
310 'Unterprogramme
320 x=x-y*konst:RETURN
330 y=y+x*konst:RETURN
340 xx=xmit%+x*xrad%
350 yy=ymit%+y*yrad%
360 RETURN
```

```
                910 *E
                920 ;Befehl CIRCLE
                930 ;
A1C8  F5        940 circle: push af         ;Anzahl der Argumente
A1C9  DD6E02    950         ld   l,(ix+2)   ;(Akku) retten.
A1CC  DD6603    960         ld   h,(ix+3)
A1CF  11F9A2    970         ld   de,yrad
A1D2  CD64BD    980         call intflo
A1D5  DD6E04    990         ld   l,(ix+4)   ;x-rad,y-rad holen und n.
A1D8  DD6605   1000         ld   h,(ix+5)   ;Fließkomma wandeln und
A1DB  11F4A2   1010         ld   de,xrad    ;ablegen.
A1DE  CD64BD   1020         call intflo
               1030 ;
A1E1  DD6E06   1040         ld   l,(ix+6)
A1E4  DD6607   1050         ld   h,(ix+7)
A1E7  2210A3   1060         ld   (ymit),hl  ;y- und x- Mitte ablegen
A1EA  DD6E08   1070         ld   l,(ix+8)
A1ED  DD6609   1080         ld   h,(ix+9)
A1F0  220EA3   1090         ld   (xmit),hl
               1100 ;
A1F3  DD7E00   1110         ld   a,(ix+0)   ;Zeichenfarbe setzen.
A1F6  CDDEBB   1120         call setpen
A1F9  3EFF     1130         ld   a,255      ;DEG einschalten !
A1FB  CD97BD   1140         call raddeg
A1FE  F1       1150         pop  af
A1FF  FE05     1160         cp   5          ;Wenn nur 5 Argumente dann
A201  281D     1170         jr   z,deflt    ;Vollkreis.
               1180 ;
A203  FE07     1190         cp   7          ;Kein Vollkreis ab.7 Argu.
A205  C0       1200         ret  nz         ;sonst Return.
A206  DD6E0A   1210         ld   l,(ix+10)
A209  DD660B   1220         ld   h,(ix+11)  ;hl= biswinkel
A20C  DD5E0C   1230         ld   e,(ix+12)
A20F  DD560D   1240         ld   d,(ix+13)  ;de= vonwinkel
A212  ED52     1250         sbc  hl,de      ;hl= hl-de entspr.Segment-
A214  D5       1260         push de         ;größe. de-Reg aufheben.
A215  CB1C     1270         rr   h
A217  CB1D     1280         rr   l          ;hl=hl/2-hl k. max 180 dz.
A219  7D       1290         ld   a,l        ;sein,desh.reicht A z.
A21A  320DA3   1300         ld   (steps),a  ;Übernahme.In steps abl.
A21D  E1       1310         pop  hl         ;ehem. de-Rg nach hl holen
A21E  1808     1320         jr   beginn     ;und zur Hauptschleife.
```



```
A220  3EB4     1330 deflt: ld   a,180      ;Defaultwerte für Vollkr.:
A222  320DA3   1340        ld   (steps),a ;Kreis hat 180 Schritte u.
A225  210000   1350        ld   hl,0       ;beginnt bei 0 Grad.
               1360 ;
A228  E5       1370 beginn: push hl         ;hl retten.
A229  2183BD   1380        ld   hl,minus+1 ;2te Minusrout.Sprungtab.
A22C  369A     1390        ld   (hl),newbyt;einbauen.(hl)=(hl)-(de).
A22E  E1       1400        pop  hl         ;hl wieder zurück.
               1410 ;
A22F  11FEA2   1420        ld   de,rechrg  ;Inh.hl-Reg(=Vonwinkel) a.
A232  CD64BD   1430        call intflo     ;Flozahl in (rechreg) abl.
A235  EB       1440        ex   de,hl      ;rechreg nach de bringen.
A236  2103A3   1450        ld   hl,xreg    ;(rechreg) nach xreg kop.
A239  E5       1460        push hl         ;Adresse v. xreg aufheben.
A23A  CD61BD   1470        call copflo     ;Flozahl vonwinkel auch n.
A23D  2108A3   1480        ld   hl,yreg    ;(yreg) kopieren.
A240  CD61BD   1490        call copflo
A243  CDACBD   1500        call sin        ;Sinus und Cosinus für
A246  E1       1510        pop  hl         ;Anfangspunkt berechnen,
A247  CDAFBD   1520        call cos        ;in Grafikkoordinaten um-
A24A  CD9AA2   1530        call koord      ;rechnen und Grafikkursor.
A24D  CDC0BB   1540        call move       ;setzen.
               1550 ;
A250  3A0DA3   1560        ld   a,(steps)
A253  47       1570        ld   b,a        ;b= Zähler
A254  F3       1580        di
A255  C5       1590 loop:  push bc         ;Interrupt ausschalten.
A256  CD78A2   1600        call xwert      ;-- HAUPTSCHLEIFE --
A259  CD89A2   1610        call ywert
A25C  CD9AA2   1620        call koord
A25F  CDF6BB   1630        call draw
A262  CD89A2   1640        call ywert      ;Weg.2-maligem Ziehen ein.
A265  CD78A2   1650        call xwert      ;Linie i.einem Durchg.hat
A268  CD9AA2   1660        call koord      ;Vollkreis auch nur
A26B  CDF6BB   1670        call draw       ;180 Schritte statt 360 !
A26E  C1       1680        pop  bc
A26F  10E4     1690        djnz loop       ;sooft wie b-Reg angibt.
A271  FB       1700        ei              ;Interrupt zulassen.
A272  2183BD   1710        ld   hl,minus+1 ;Original-Minusroutine
A275  369E     1720        ld   (hl),oldbyt;zurück.(hl)=(de)-(hl).
A277  C9       1730        ret             ;Nach Basic
               1740 ;
A278  CDDEA2   1750 xwert: call ycopy      ;Fließkommaberechnung:
A27B  1112A3   1760        ld   de,konst
A27E  CD85BD   1770        call mult       ;x=x-(y*Konstante)
A281  EB       1780        ex   de,hl
A282  2103A3   1790        ld   hl,xreg    ;Konstante = 0.5
A285  CD82BD   1800        call minus
A288  C9       1810        ret
               1820 ;
A289  CDD4A2   1830 ywert: call xcopy      ;Fließkommaberechnung:
A28C  1112A3   1840        ld   de,konst
A28F  CD85BD   1850        call mult       ;y=y+(x*Konstante)
A292  EB       1860        ex   de,hl
A293  2108A3   1870        ld   hl,yreg    ;Konstante = 0.5
A296  CD7CBD   1880        call plus
A299  C9       1890        ret
               1900 ;
```

```
A29A  CDDEA2    1910 koord:  call  ycopy     ;Grafikkoordinaten vom
A29D  11F9A2    1920         ld    de,yrad   ;momentanen Mittelp. aus
A2A0  CD85BD    1930         call  mult      ;berechnen.
A2A3  CD94BD    1940         call  signum    ;Nach Formel:
A2A6  F5        1950         push  af        ;hl = ymitte + (yreg*yrad)
A2A7  CD6ABD    1960         call  floint    ;In Klammer-Flozahl-sonst
A2AA  F1        1970         pop   af        ;Integerberechnung.
A2AB  3003      1980         jr    nc,nomin  ;Vorzeichen berücksicht.!
A2AD  CDE8A2    1990         call  chgsig    ;evtl.Vorzeichen wechseln
A2B0  ED5B10A3  2000 nomin:  ld    de,(ymit)
A2B4  19        2010         add   hl,de     ;hl ist nun y-wert
A2B5  E5        2020         push  hl        ;diesen aufheben
                2030 ;
A2B6  CDD4A2    2040         call  xcopy     ;wie vor,jedoch für xwert.
A2B9  11F4A2    2050         ld    de,xrad
A2BC  CD85BD    2060         call  mult      ;Nach Formel:
A2BF  CD94BD    2070         call  signum    ;de = xmitte + (xreg*xrad)
A2C2  F5        2080         push  af        ;In Klammer-Flozahl-sonst
A2C3  CD6ABD    2090         call  floint    ;Integerberechnung.
A2C6  F1        2100         pop   af        ;Vorz.d.Flozahl bestimmen.
A2C7  3003      2110         jr    nc,nomin1 ;u.evtl.Vorzeichen d.Int.-
A2C9  CDE8A2    2120         call  chgsig    ;zahl wechseln.
A2CC  ED5B0EA3  2130 nomin1: ld    de,(xmit) ;Registerinh. vertauschen
A2D0  19        2140         add   hl,de     ;hl zurueck, de u. hl ent-
A2D1  EB        2150         ex    de,hl     ;halten nun x- u. y-Koord.
A2D2  E1        2160         pop   hl
A2D3  C9        2170         ret
A2D4  21FEA2    2180 xcopy:  ld    hl,rechrg ;Unterprogramm für Inhalt
A2D7  1103A3    2190         ld    de,xreg   ;xreg nach Rechenregister
A2DA  CD61BD    2200         call  copflo    ;kopieren (sind 5 Bytes).
A2DD  C9        2210         ret
A2DE  21FEA2    2220 ycopy:  ld    hl,rechrg ;wie vor,jedoch für yreg.
A2E1  1108A3    2230         ld    de,yreg
A2E4  CD61BD    2240         call  copflo
A2E7  C9        2250         ret
                2260 ;
A2E8  AF        2270 chgsig: xor   a         ;Wechsle Vorzeichen.
A2E9  95        2280         sub   l         ;Unterprogramm a.ROM kop.
A2EA  6F        2290         ld    l,a       ;da bei <>464 im Basicrom.
A2EB  9C        2300         sbc   a,h
A2EC  95        2310         sub   l
A2ED  BC        2320         cp    h
A2EE  67        2330         ld    h,a
A2EF  37        2340         scf
A2F0  C0        2350         ret   nz
A2F1  FE01      2360         cp    1
A2F3  C9        2370         ret
                2380 ;
A2F4  00000000  2390 xrad:   defb 0,0,0,0,0
A2F9  00000000  2400 yrad:   defb 0,0,0,0,0
A2FE  00000000  2410 rechrg: defb 0,0,0,0,0
A303  00000000  2420 xreg:   defb 0,0,0,0,0
A308  00000000  2430 yreg:   defb 0,0,0,0,0
A30D  00        2440 steps:  defb 0
A30E  0000      2450 xmit:   defb 0,0
A310  0000      2460 ymit:   defb 0,0
A312  1335FA0E  2470 konst:  defb #13,#35,#fa,#0e,#7b
```


Befehl COPY

```
!COPY
```

Um eine Grauabstufung beim Ausdruck zu erreichen, muß tief in die Trickkiste gegriffen werden.
Um einen eventuellen Mißverständnis vorzubeugen: Mit Grauton meine ich nicht die (unfreiwillige) Verwendung eines alten Farbbandes, sondern das Verhältnis der am Papier gesetzten Punkte zu den Leerstellen !

Aber nun wieder im Ernst !
Im MODE 0 haben Sie fünf Graustufen (einschließlich schwarz und weiß), im MODE 1 immerhin noch vier und im MODE 2 nur noch weiß und schwarz, dafür aber die höchste Auflösung von allen. Das Programm stellt sich automatisch auf die entsprechenden Modi ein, indem es selbsttätig seine Programmierung ändert.

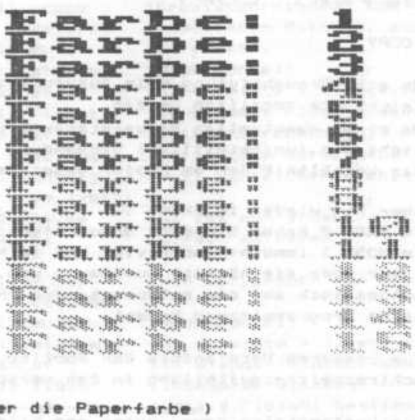
Zum besseren Verständnis der Routine muß ich etwas über die Bildschirmspeicheraufteilung in den verschiedenen Modi anmerken.

Die horizontalen Koordinaten des Bildschirms sind immer in dem Bereich 0-639, aber je nach dem verwendeten MODE ist die Anzahl der tatsächlich darstellbaren Pixel verschieden und zwar 160 (Mode 2),320 (Mode 1) und alle 640 im Mode 2.
Daraus ergibt sich, daß der einzelne Punkt oder Pixel im Mode 0 ganze vier horizontale Bildpunkte verbraucht (640 / 160), im Mode 1 sind's noch zwei und im Mode 2 endlich nur noch einer.
Dieses Verhalten des Schneiders kann man sich auch beim Ausdruck zunutze machen, denn der Drucker kann ebenfalls vier nebeneinander liegende horizontale Punkte für einen ganzen Bildpunkt verwenden. Mit einem unterschiedlichen Setzen dieser einzelnen Punkte lassen sich theoretisch schon vier Graustufen darstellen.
Das ist aber noch nicht alles, wie Sie sicher wissen, benützt der Bildschirm Ihres Computers in der vertikalen Richtung die Koordinaten 0-399, hat aber in Wirklichkeit nur die Auflösung von 200 Pixel, also pro Pixel zwei Koordinatenpunkte. Und dies bleibt konstant in allen Modi !
Auch der Drucker steuert pro Pixel zwei vertikale Nadeln an, um die richtigen Seitenverhältnisse zu bewahren. Damit sind es also vier horizontale und zwei vertikale Druckernadeln pro Pixel im Bildschirmmode 0.
Eigentlich ließen sich damit 2*4, also 8 verschiedene Grauschattierungen erreichen, aber wie gesagt nur theoretisch, denn die tatsächlich sichtbare ! Auflösung eines Matrix-Druckers ist wesentlich geringer, besonders in den dunkel-grauen Tönen. Wir begnügen uns mit fünf Grautönen und liegen damit goldrichtig.

Pixel/Zeichenplatz entsprechend Modus:

```
        ******** ******** ****************
        ******** ******** ****************
        ******** ******** ****************
        ******** ******** ****************
        ******** ******** ****************
        ******** ******** ****************
        ******** ******** ****************
        ******** ******** ****************
        -Mode 2-          ------Mode 1-----
        --------------Mode 0---------------
```

Noch ein bildhafte Darstellung:




```
================================

Drucker-"Pixel" im Mode Ø:

 1111
 1111      entspricht Schwarz

 1Ø1Ø
 111Ø        "        D'Grau

 1Ø1Ø
 Ø1ØØ        "        M'Grau

 1ØØØ
 ØØ1Ø        "        H'Grau

 ØØØØ
 ØØØØ        "        Weiss ( immer die Paperfarbe )

 XX  <-- Diese vier Nadeln werden im Mode 1 verwendet.
 XX  <--

 X   <-- Und nur diese beiden im Mode 2.
 X   <--

======================================================================
```

So, und nach soviel Theorie endlich das Programm:

```
                 2480 *E
                 2490 ;Befehl COPY
                 2500 ;
ØØFF             2510 schwrz: equ  255
ØØAE             2520 dgrau:  equ  %1Ø1Ø111Ø          ;Druckermatrixen
ØØA4             2530 mgrau:  equ  %1Ø1ØØ1ØØ          ;für d.einzelnen
ØØ82             2540 hgrau:  equ  %1ØØØØØ1Ø          ;Graustufen.
ØØØØ             2550 weiss:  equ  Ø
                 2560 ;
A317  AØØØD4A3   2570 modis:  defw 16Ø,sprngØ         ;Tab.f.Programm-
A31B  13131313   2580         defb #13,#13,#13,#13    ;    änderungen.
A31F  Ø4         2590         defb 4                  ;Mode Ø- Werte
                 2600 ;
A32Ø  4ØØ1F1A3   2610 nxtmod: defw 32Ø,sprng1
A324  1313ØØØØ   2620         defb #13,#13,Ø,Ø
A328  Ø2         2630         defb 2                  ;Mode 1-Werte
                 2640 ;
A329  8ØØ2Ø8A4   2650         defw 64Ø,sprng2
A32D  13ØØØØØØ   2660         defb #13,Ø,Ø,Ø
A331  Ø1         2670         defb 1                  ;Mode 2-Werte
                 2680 ;
A332  ØØ         2690 paper:  defb Ø                 ;Zwisch.speich.f.Paper.
A333  ØØØØØØØØ   2700 drbyts: defb Ø,Ø,Ø,Ø           ;u. 4 Druckerbytes.
                 2710 ;
                 2720 ;Programm stellt sich selbst
                 2730 ;auf entsprechenden Modus ein !
A337  ED7351A5   2740 copy:   ld   (return),sp ;f.ESC-Aussprung.
```



```
A33B  2117A3     2750         ld   hl,modis   ;Adresse vorbereiten
A33E  11Ø9ØØ     2760         ld   de,nxtmod-modis
A341  CD11BC     2770         call getmod     ;Mode holen
A344  FEØ1       2780         cp   1
A346  28Ø5       2790         jr   z,lab1
A348  FEØ2       2800         cp   2
A34A  200?       2810         jr   nz,lab2
A34C  19         2820         add  hl,de      ;und daraus Tabellen-
A34D  19         2830 lab1:   add  hl,de      ;Anfang berechnen
A34E  5E         2840 lab2:   ld   e,(hl)
A34F  23         2850         inc  hl
A35Ø  56         2860         ld   d,(hl)       ;Dann folgende Werte
A351  ED538AA3   2870         ld   (bcreg+1),de ;einstellen:
A355  23         2880         inc  hl
A356  5E         2890         ld   e,(hl)     ;bc-Reg in Zeile 315Ø mit
A357  23         2900         inc  hl         ;   Anzahl der Y-Pixel
A358  56         2910         ld   d,(hl)     ;   laden.
A359  ED5396A3   2920         ld   (tocall+1),de;call entsprechend änd.
A35D  23         2930         inc  hl           ;   in Zeile 322Ø
A35E  Ø1Ø4ØØ     2940         ld   bc,4       ;Anzahl d.Decrements fest-
A361  11B1A3     2950         ld   de,decrem  ;   leg.(Z.343Ø-346Ø).
A364  EDBØ       2960         ldir            ;b-Reg  i.Zeile 331Ø laden
A366  7E         2970         ld   a,(hl)     ;   (Zahl d.Nadeln in
A367  32A3A3     2980         ld   (byts+1),a;    der Breite)
                 2990 ;
A36A  FD2132A3   3000         ld   iy,paper
A36E  CDE7BB     3010         call getpap     ;Paperfarbe bestimmen u.z.
A371  FD77ØØ     3020         ld   (iy+Ø),a   ;Vergleich abl.,iy=Zeiger.
                 3030 ;
A374  2155A4     3040         ld   hl,zeiesc  ;Zeilenabstand senden,
A377  CD46A4     3050         call send       ;8-Nadel-Höhe.

                 3060 ;
                 3070 ;Hauptschleife
A37A  219ØØ1     3080         ld   hl,4ØØ     ;hl= 4ØØ (Höhe-Koord.)
A37D  2B         3090 zeil5Ø: dec  hl         ;hl-1 weil Ø-399, ist auch
A37E  E5         3100         push hl         ;achtes dec-hl i.Schleife.
A37F  214FA4     3110         ld   hl,graesc  ;Grafiksequenz senden.
A382  CD46A4     3120         call send
A385  E1         3130         pop  hl
A386  11ØØØØ     3140         ld   de,Ø       ;de=Ø,ganz links beginnen.
A389  Ø1AØØØ     3150 bcreg:  ld   bc,16Ø     ;Pixel/Zeile-entspr.Mode.
A38C  C5         3160 hbzeil: push bc
A38D  Ø6Ø4       3170         ld   b,4
A38F  D5         3180 pixel4: push de         ;Innerste Schleife:
A39Ø  C5         3190         push bc         ;4Pixel untereinand.testen
A391  E5         3200         push hl         ;Grauwerte bestimmen und
A392  CDFØBB     3210         call tstpkt     ;bei Druckerbytes ablegen.
A395  CDD4A3     3220 tocall: call sprngØ
A398  CD13A4     3230         call rotate
A39B  E1         3240         pop  hl
A39C  2B         3250         dec  hl         ;2*dec w.immer 2 vertikale
A39D  2B         3260         dec  hl         ;Koord. ein Pixel ergeben.
A39E  C1         3270         pop  bc
A39F  D1         3280         pop  de
A3AØ  1ØED       3290         djnz pixel4
                 3300 ;
A3A2  Ø6Ø4       3310 byts:   ld   b,4        ;!!Modeabhängig,Anz.d.tat-
A3A4  E5         3320         push hl         ;sächlich z.Ausdruck kom-
```

```
A3A5  CD24A4    3330          call druck      ;menden Druckerbytes.
A3A8  E1        3340          pop  hl
A3A9  23        3350          inc  hl
A3AA  23        3360          inc  hl         ;Da noch immer i.d.selben
A3AB  23        3370          inc  hl         ;Grafikzeile müs.d.2 decr.
A3AC  23        3380          inc  hl         ;der Zeilen 3250/60 nach 4
A3AD  23        3390          inc  hl         ;Durchläufen ausgeglichen
A3AE  23        3400          inc  hl         ;werden.
A3AF  23        3410          inc  hl
A3B0  23        3420          inc  hl
A3B1  13        3430 decrem:  inc  de         ;!!Modeabhängig
A3B2  13        3440          inc  de         ;Anz.d.horizontalen Pixel
A3B3  13        3450          inc  de         ;die zu überspringen sind.
A3B4  13        3460          inc  de
A3B5  C1        3470          pop  bc
A3B6  0B        3480          dec  bc
A3B7  78        3490          ld   a,b
A3B8  B1        3500          or   c
A3B9  20D1      3510          jr   nz,hbzeil  ;Grafikzeile fertig ?
                3520 ;
A3BB  E5        3530          push hl
A3BC  2161A4    3540          ld   hl,linefd  ;Wenn ja Line-Feed senden.
A3BF  CD46A4    3550          call send
A3C2  E1        3560          pop  hl         ;f.neue! Grafikzeile jetzt
A3C3  2B        3570          dec  hl         ;d. 8 Incr.neutralisieren.
A3C4  2B        3580          dec  hl
A3C5  2B        3590          dec  hl         ;hier nur 7*dec hl damit
A3C6  2B        3600          dec  hl         ;Sprungbedingung erfüllt
A3C7  2B        3610          dec  hl         ;sein kann (hl=0) !
A3C8  2B        3620          dec  hl
A3C9  2B        3630          dec  hl
A3CA  7C        3640          ld   a,h
A3CB  B5        3650          or   l          ;n. insges.50 Durchläufen
A3CC  20AF      3660          jr   nz,zeil50  ;fertig (50 Halbzeilen).
                3670 ;                        ;Drucker in Einschaltmodus
A3CE  215BA4    3680          ld   hl,retesc  ;bringen und dieses RET
A3D1  C346A4    3690          jp   send       ;nach BASIC nützen.
                3700 ;ENDE Hauptschleife
                3710 ;
                3720 ;Pixel in MODE 0 lesen ;Pixel m.Paper vergl.,
A3D4  FDBE00    3730 sprng0:  cp   (iy+0)     ;w. wahr d.weiß als Farbe.
A3D7  0600      3740          ld   b,weiss
A3D9  2814      3750          jr   z,setcol   ;Sonst d.Grauw. bestimmen,
A3DB  06FF      3760          ld   b,schwrz   ;4 versch. Farben geben e.
A3DD  FE04      3770          cp   4          ;Grauwert.
A3DF  380E      3780          jr   c,setcol
A3E1  06AE      3790          ld   b,dgrau
A3E3  FE08      3800          cp   8
A3E5  3808      3810          jr   c,setcol
A3E7  06A4      3820          ld   b,mgrau    ;schließlich diesen Grau-
A3E9  FE0C      3830          cp   12         ;wert in Akku laden.
A3EB  3802      3840          jr   c,setcol
A3ED  0682      3850          ld   b,hgrau
A3EF  78        3860 setcol:  ld   a,b
A3F0  C9        3870          ret
                3880 ;
                3890 ;Pixel im MODE 1 lesen ;Gleiche Prozedur i.Model,
A3F1  FDBE00    3900 sprng1:  cp   (iy+0)     ;nur daß hier alle Farben
```



```
A3F4  0600      3910          ld   b,weiss    ;dargestellt werd. können.
A3F6  28F7      3920          jr   z,setcol
A3F8  06FF      3930          ld   b,schwrz
A3FA  FE01      3940          cp   1
A3FC  28F1      3950          jr   z,setcol
A3FE  06A4      3960          ld   b,mgrau
A400  FE02      3970          cp   2
A402  28EB      3980          jr   z,setcol
A404  0682      3990          ld   b,hgrau
A406  18E7      4000          jr   setcol
                4010 ;
                4020 ;Pixel im Mode 2 lesen ;Im Mode 2 gibt's nur
A408  06FF      4030 sprng2:  ld   b,schwrz   ;Vorder- und Hintergrund.
A40A  FDBE00    4040          cp   (iy+0)
A40D  20E0      4050          jr   nz,setcol
A40F  0600      4060          ld   b,weiss
A411  18DC      4070          jr   setcol
                4080 ;
A413  0E02      4090 rotate:  ld   c,2        ;D.Drucker-"Farb"-Wert der
A415  2133A3    4100 l1:      ld   hl,drbyts  ;Reihe nach in die 4
A418  0604      4110          ld   b,4        ;Druckerbytes schieben.
A41A  07        4120 l2:      rlca
A41B  CB16      4130          rl   (hl)       ;(E.Zeichen dafür w.effek.
A41D  23        4140          inc  hl         ;d.Rotate-Befehle des Z-80
A41E  10FA      4150          djnz l2         ;eingesetzt werd. können!)
A420  0D        4160          dec  c
A421  20F2      4170          jr   nz,l1
A423  C9        4180          ret
                4190 ;
A424  2133A3    4200 druck:   ld   hl,drbyts  ;Von 4 bereitgest.Drucker-
A427  7E        4210 l3:      ld   a,(hl)     ;bytes b-viele z. Ausdruck
A428  CD2FA4    4220          call waiprt     ;bringen (je nach Mode).
A42B  23        4230          inc  hl
A42C  10F9      4240          djnz l3
A42E  C9        4250          ret
A42F  F5        4270 waiprt:  push af
A430  CD09BB    4280          call readch     ;Teste auf ESCape, dann
A433  FEFC      4290          cp   #fc        ;zurück nachdem Stack
A435  280A      4300          jr   z,escape   ;ausgeglichen.
A437  F1        4310          pop  af
A438  CD2EBD    4320          call busy       ;Warten auf Busy, damit k.
A43B  38F2      4330          jr   c,waiprt   ;Zeichen verloren geht u.
A43D  CD2BBD    4340          call print      ;dann drucken.
A440  C9        4350          ret
                4360 ;
A441  ED7B51A5  4370 escape:  ld   sp,(return)
A445  C9        4380          ret
                4390 ;
A446  7E        4400 send:    ld   a,(hl)     ;Hilfsprogramm zum Senden
A447  B7        4410          or   a          ;Steuersequenzen an den
A448  C8        4420          ret  z          ;Drucker.
A449  CD2FA4    4430          call waiprt
A44C  23        4440          inc  hl
A44D  18F7      4450          jr   send
A44F  1B4B0802  4460 graesc:  defb 27,"K",#08,#02,0,0   ;Platz reicht
A455  1B410800  4470 zeiesc:  defb 27,"A",8,0,0,0       ;auch f.mehr
A45B  1B320000  4480 retesc:  defb 27,"2",0,0,0,0       ;ESC-Seq.eines
A461  0A0D00    4490 linefd:  defb 10,13,0              ;and. Druckers.
```

Befehl TEXT

```
!TEXT,farbe,breite,hoehe,@String

z.B.
10 a$="Hallo"
20 !TEXT,1,10,4,@a$
```

Der Befehl plottet ein beliebiges Zeichen oder String beliebig groß auf dem Bildschirm. Die Breite und die Höhe sind dabei in Grafikpixeln angegeben.

Das Programm:
Zuerst üblicherweise die Überprüfung auf die Parameteranzahl und eventuell Rücksprung. Dann wird der Grafikkursor in einer Speicherzelle abgelegt. Nun wird das erste Zeichen des Strings geholt, überprüft ob die Matrix dafür im ROM oder RAM zu suchen ist, dann die Matrix in einen RAM-Buffer kopiert (zur einfacheren Handhabung) und schließlich diese Matrix mit den angegebenen Parametern in X- und Y-Richtung vergrößert und gleichzeitig in der gewünschten Farbe geplottet.

Dies geschieht solange, bis der String abgearbeitet ist, dabei werden die Koordinaten immer für das nächste Zeichen aktualisiert.

```
                4500 *E
                4510 ;Befehl TEXT
                4520 ;
A464  FE04      4530 text:  cp   4          ;Wenn weniger als 4 Argum.
A466  C0        4540        ret  nz         ;dann zurueck.
A467  DD6601    4550        ld   h,(ix+1)   ;Stringlänge holen,
A46A  DD6E00    4560        ld   l,(ix+0)
A46D  7E        4570        ld   a,(hl)
A46E  B7        4580        or   a
A46F  C8        4590        ret  z          ;wenn Null dann zurück.
A470  3246A5    4600        ld   (lenstr),a
A473  23        4610        inc  hl         ;Stringdescriptor
A474  5E        4620        ld   e,(hl)     ;aufbereiten.
A475  23        4630        inc  hl
A476  56        4640        ld   d,(hl)
A477  ED5344A5  4650        ld   (adrstr),de
                4660 ;
A47B  DD7E02    4670        ld   a,(ix+2)   ;Höhe und Breite holen
A47E  3242A5    4680        ld   (hoehe),a  ;und ablegen.
A481  3243A5    4690        ld   (hoehe1),a
A484  DD7E04    4700        ld   a,(ix+4)
A487  3241A5    4710        ld   (breite),a
A48A  DD7E06    4720        ld   a,(ix+6)   ;gewünschte Grafikfarbe
A48D  FE1B      4730        cp   27         ;setzen.
A48F  D0        4740        ret  nc
A490  CDDEBB    4750        call graset
                4760 ;
A493  CDC6BB    4770        call askcur     ;Grafikkursor holen
A496  ED5347A5  4780        ld   (xpos),de  ;und ablegen.
A49A  ED5349A5  4790        ld   (xpos1),de
A49E  224DA5    4800        ld   (ypos),hl
A4A1  224FA5    4810        ld   (ypos1),hl
                4820 ;
```



```
                4830 ;programm text
A4A4  2A44A5    4840        ld   hl,(adrstr);Wo ist Matrix vom
A4A7  7E        4850 loop1: ld   a,(hl)     ;aktuellem Zeichen?
A4A8  23        4860        inc  hl
A4A9  E5        4870        push hl
A4AA  CDA5BB    4880        call getmat     ;Im ROM oder RAM.
A4AD  3005      4890        jr   nc,rom     ;Carry=0,dann im ROM
A4AF  CDBCA4    4900        call adr2       ;Copy RAM-Zeichen nach
A4B2  1811      4910        jr   weiter     ;Matrixbytes
                4920 ;
A4B4  DF        4930 rom:   rst  #18        ;Copy ROM-Zeichen nach
A4B5  B9A4      4940        defw rerom      ;Matrixbytes
A4B7  180C      4950        jr   weiter
A4B9  BCA4      4960 rerom: defw adr2
A4BB  FC        4970        defb #fc
                4980 ;
A4BC  1153A5    4990 adr2:  ld   de,matrix  ;Unterroutine zum Kopieren
A4BF  010800    5000        ld   bc,8
A4C2  EDB0      5010        ldir
A4C4  C9        5020        ret
                5030 ;
A4C5  2153A5    5040 weiter: ld  hl,matrix  ;Matrixzeiger auf Anfang
A4C8  2251A5    5050        ld   (matzei),hl;von Matrix.
A4CB  0608      5060        ld   b,8        ;B=8 - (Ein Zeichen hat 8
A4CD  C5        5070 loop2: push bc         ;Bytes),auf Stack.
A4CE  3A42A5    5080        ld   a,(hoehe)  ;Höhe nach B
A4D1  47        5090        ld   b,a
A4D2  2A51A5    5100        ld   hl,(matzei);Akku m. aktuellem Matrix-
A4D5  7E        5110        ld   a,(hl)     ;Byte laden,HL-Zeig.incre-
A4D6  23        5120        inc  hl         ;mentieren u.Zeiger zurück
A4D7  2251A5    5130        ld   (matzei),hl
A4DA  C5        5140 loop3: push bc         ;Höhe auf Stack
A4DB  0608      5150        ld   b,8        ;Ein Zeichen i.im Original
A4DD  C5        5160 loop4: push bc         ;8 Bit breit,dies a.Stack.
                5170 ;
A4DE  07        5180        rlca            ;Akku rotieren durch Carry
A4DF  301A      5190        jr   nc,noplot  ;Bit i.Carry=0,nicht plot.
                5200 ;
A4E1  F5        5210        push af         ;Akku wird noch gebraucht.
A4E2  3A41A5    5220        ld   a,(breite)
A4E5  47        5230        ld   b,a        ;gewählte Breite nach B.
A4E6  7B        5240        ld   a,e
A4E7  2A4FA5    5250        ld   hl,(ypos1);HL+DE mit aktueller
A4EA  ED5B49A5  5260        ld   de,(xpos1);Position laden
A4EE  C5        5270 loop5: push bc         ;Plotroutine
A4EF  E5        5280        push hl         ;Sooft mal in X-Richtung
A4F0  D5        5290        push de         ;plotten w. Breite angibt.
A4F1  CDEABB    5300        call plot
A4F4  D1        5310        pop  de
A4F5  E1        5320        pop  hl
A4F6  13        5330        inc  de
A4F7  C1        5340        pop  bc
A4F8  10F4      5350        djnz loop5
A4FA  F1        5360        pop  af
                5370 ;
A4FB  2A49A5    5380 noplot: ld  hl,(xpos1);für nächste Plotposition
A4FE  0600      5390        ld   b,0        ;xpos1=xpos1+Breite
A500  5F        5400        ld   e,a
```

```
A501  3A41A5   5410          ld   a,(breite)
A504  4F       5420          ld   c,a
A505  7B       5430          ld   a,e
A506  09       5440          add  hl,bc
A507  2249A5   5450          ld   (xposi),hl
               5460 ;
A50A  C1       5470          pop  bc          ;eine Bitreihe abarbeiten.
A50B  10D0     5480          djnz loop4
               5490 ;
A50D  224BA5   5500          ld   (xposn),hl ;xposn=Anfg.des 1.Punktes
A510  2A47A5   5510          ld   hl,(xpos)  ;vom nächsten Zeichen.
A513  2249A5   5520          ld   (xposi),hl ;xposi=xpos für nächste
A516  2A4FA5   5530          ld   hl,(yposi) ;Punktreihe.
A519  2B       5540          dec  hl          ;Höhe-1,(ypos)
A51A  224FA5   5550          ld   (yposi),hl
A51D  C1       5560          pop  bc          ;sooft m. d.gleiche Punkt-
A51E  10BA     5570          djnz loop3       ;reihe untereinandersetzen
               5580 ;                         ;wie Höhe angibt.
A520  C1       5590          pop  bc          ;nächst.Bitreihe desselben
A521  10AA     5600          djnz loop2       ;Zeichens bearbeiten.
               5610 ;
A523  2A4DA5   5620          ld   hl,(ypos)  ;ursprüngliche Höhe it Ar-
A526  224FA5   5630          ld   (yposi),hl ;beitshöhe für Neubeginn.
A529  3A46A5   5640          ld   a,(lenstr) ;Stringlänge-1
A52C  3D       5650          dec  a
A52D  3246A5   5660          ld   (lenstr),a ;solange bis alle Zeichen
A530  E1       5670          pop  hl          ;aus String abgearbeitet
A531  C8       5680          ret  z           ;sind.
               5690 ;
A532  ED5B4BA5 5700          ld   de,(xposn) ;sonst Anfang X-Richtung
A536  ED5347A5 5710          ld   (xpos),de  ;für nächstes Zeichen
A53A  ED5349A5 5720          ld   (xposi),de ;umschaufeln.
A53E  C3A7A4   5730          jp   loop1
A541  00       5740 breite:  defb 0
A542  00       5750 hoehe:   defb 0
A543  00       5760 hoehe1:  defb 0
A544  0000     5770 adrstr:  defw 0
A546  00       5780 lenstr:  defb 0
A547  0000     5790 xpos:    defw 0
A549  0000     5800 xposi:   defw 0
A54B  0000     5810 xposn:   defw 0
A54D  0000     5820 ypos:    defw 0
A54F  0000     5830 yposi:   defw 0
               5840 ;auch als Zwischenspeicher
               5850 ;fuer Stackpointer bei COPY.
A551           5860 return:  equ  $
A551  0000     5870 matzei:  defw 0
A553  00000000 5880 matrix:  defb 0,0,0,0,0,0,0,0
```

Befehl TAUSCH

```
!TAUSCH,@Variable,@Variable,Art
z.B.
!TAUSCH,@a$,@b$,0       für Stringvariable
!TAUSCH,@a%,@b%,1       für Integervariable
!TAUSCH,@a ,@b ,2       für Fließkommavariable
```


Mit diesem Befehl können Sie Variable gleicher Art austauschen. Besonders bei der Stringbehandlung ein sehr nützlicher Befehl, denn bei diversen Sortierprogrammen können Sie die gefürchtete Garbage-Collektion, die die Sortierung unverhältnismäßig verlangsamt von nun an vergessen. Die Variablen werden gegeneinander ausgetauscht, ohne daß sie neu angelegt werden. Sie müssen bei den Strings nur darauf achten, daß beide die gleiche Länge haben, sonst springt das Programm zurück, ohne daß etwas geschieht.

Das Programm:

Je nach Art der zu tauschenden Variablen werden 2-(Integer), 5-(Fließpunkt) und bis zu 255 Bytes bei Strings verarbeitet.

```
               5890 *E
               5900 ;Befehl TAUSCH
               5910 ;
A55B  FE03     5920 tausch: CP   3             ;3 Parameter sonst zurück
A55D  C0       5930         RET  NZ
A55E  DD6E02   5940         LD   L,(IX+2)    ;HL=Pointer auf Var. oder
A561  DD6603   5950         LD   H,(IX+3)    ;Descriptor 1.
A564  DD5E04   5960         LD   E,(IX+4)    ;DE=Pointer auf Var. oder
A567  DD5605   5970         LD   D,(IX+5)    ;Descriptor 2.
A56A  DD7E00   5980         LD   A,(IX+0)    ;Akku = Art (Text,Int...)
               5990 ;
A56D  CB4F     6000         BIT  1,A         ;Prüfe Art und springe
A56F  201D     6010         JR   NZ,real     ;entsprechend.
               6020 ;
A571  CB47     6030         BIT  0,A
A573  2015     6040         JR   NZ,integ
               6050 ;
               6060 ;text
A575  1A       6070         LD   A,(DE)      ;zurück w. Länge String 1
A576  B7       6080         OR   A           ;ist Null.
A577  C8       6090         RET  Z
               6100 ;
A578  46       6110         LD   B,(HL)      ;zurück w. Länge String 1
A579  B8       6120         CP   B           ;und String 2 ungleich.
A57A  C0       6130         RET  NZ          ;B-reg ist Länge String
               6140 ;
A57B  13       6150         INC  DE          ;HL + DE zeigen nun auf
A57C  23       6160         INC  HL          ;Adresse von Strings.
               6170 ;
               6180 ;Simulation von LD DE,(DE)
A57D  E5       6190         PUSH HL          ;Programmiertrick um
A57E  EB       6200         EX   DE,HL       ;das Register DE mit dem
A57F  4E       6210         LD   C,(HL)      ;Inhalt der Adresse zu
A580  23       6220         INC  HL          ;laden, auf die DE vorher
A581  56       6230         LD   D,(HL)      ;zeigte (16-Bit-Wort).
A582  59       6240         LD   E,C
A583  E1       6250         POP  HL
               6260 ;
               6270 ;Simulation von LD HL,(HL)
A584  4E       6280         LD   C,(HL)      ;Das gleiche mit HL-Reg.
A585  23       6290         INC  HL
A586  66       6300         LD   H,(HL)
A587  69       6310         LD   L,C
A588  1806     6320         JR   exanch
```

```
                    6330 ;
A58A  0602          6340 integ:  LD   B,2          ;Integer hat zwei Byte,
A58C  1802          6350         JR   exanch      ;diese tauschen.
                    6360 ;
A58E  0605          6370 real:   LD   B,5          ;Realzahl hat 5 Byte
                    6380 ;
A590  4E            6390 exanch: LD   C,(HL)       ;tauschen (DE)<->(HL)
A591  1A            6400         LD   A,(DE)      ;B-Reg ist Zähler.
A592  77            6410         LD   (HL),A
A593  79            6420         LD   A,C
A594  12            6430         LD   (DE),A
A595  13            6440         INC  DE
A596  23            6450         INC  HL
A597  10F7          6460         DJNZ exanch
A599  C9            6470         RET
```

Befehl PAUSE

```
!PAUSE,wert * 1/10 sec.
```

Es lässt sich damit ein Päuschen bis zu 6556 Sekunden oder ca. 110 Minuten einlegen, ohne eine lästige leere FOR-NEXT- Schleife in Basic schreiben zu müssen.

Das Programm besteht nur aus zwei ineinander geschachtelten Schleifen mit einer Break-Tasten Abfrage, falls es Ihnen zu lange dauert.

```
                    6480 #E
                    6490 !Befehl PAUSE
                    6500 ;
A59A  DD6601        6510 pause:  ld   h,(ix+1)
A59D  DD6E00        6520         ld   l,(ix+0)
A5A0  01F301        6530 aloop:  ld   bc,#01f3
A5A3  CD09BB        6540 bloop:  call readch
A5A6  FEFC          6550         cp   #fc
A5A8  C8            6560         ret  z
A5A9  0B            6570         dec  bc
A5AA  78            6580         ld   a,b
A5AB  B1            6590         or   c
A5AC  20F5          6600         jr   nz,bloop
A5AE  2B            6610         dec  hl
A5AF  7C            6620         ld   a,h
A5B0  B5            6630         or   l
A5B1  20ED          6640         jr   nz,aloop
A5B3  C9            6650         ret
```

Befehl DPOKE

```
!DPOKE,adresse,16-Bit-Wert
```

Damit lässt sich ein 16-Bit-Wert in Zilog Schreibweise (lower-Byte zuerst) in den Speicher poken.

Befehl DPEEK

```
!DPEEK,adresse,@wert
```


Ein 16-Bit-Wort wird wieder in einer Basicvariablen abgelegt, deshalb der "Klammeraffe" vor der Variablen "wert".

```
                    6660 #E
                    6670 !Befehl DPOKE
                    6680 ;
A5B4  CDC6A5        6690 dpoke:  call var
A5B7  73            6700         ld   (hl),e     ;16-Bit Wert DE bei
A5B8  23            6710         inc  hl         ;Adressse (HL) ablegen.
A5B9  72            6720         ld   (hl),d
A5BA  C9            6730         ret
                    6740 ;
                    6750 !Befehl DPEEK
                    6760 ;
A5BB  CDC6A5        6770 dpeek:  call var
A5BE  4E            6780         ld   c,(hl)     ;16-Bit-Wert (HL)
A5BF  23            6790         inc  hl         ;nach BC-Reg.
A5C0  46            6800         ld   b,(hl)
A5C1  EB            6810         ex   de,hl      ;DE nach HL: HL wieder
A5C2  71            6820         ld   (hl),c     ;als Zeiger verwenden
A5C3  23            6830         inc  hl         ;diesmal auf Integer-
A5C4  70            6840         ld   (hl),b     ;Basicvariable und Wert
A5C5  C9            6850         ret             ;ablegen.
                    6860 ;
A5C6  FE02          6870 var:    cp   2
A5C8  C1            6880         pop  bc
A5C9  C0            6890         ret  nz
A5CA  C5            6900         push bc
A5CB  DD6E02        6910         ld   l,(ix+2)
A5CE  DD6603        6920         ld   h,(ix+3)
A5D1  DD5E00        6930         ld   e,(ix+0)
A5D4  DD5601        6940         ld   d,(ix+1)
A5D7  C9            6950         ret
```

Befehl RESTORE

```
!RESTORE,Variable als Zeilennummer

z.B.
10 x=10
20 DATA 1,2,3
30 DATA 4,5,6
40 READ a:PRINT a
50 IF a=3 THEN !RESTORE x
60 GOTO 40
```

Was in diesem Beispiel nicht so recht zum Ausdruck kommt, ist, daß Sie eine berechnete Zeilennummer einsetzen können und somit bei einem Programm mit einem großen DATA-Anteil keine umständliche Programmierung mehr in Kauf nehmen müssen. Allerdings wird bei einer Neunummerierung des Programmes eine Überprüfung der Berechnungsformel notwendig sein.

Das Programm:

Diese Routine enthält einige Besonderheiten. Zum ersten wird das Basic-ROM permanent zugeschaltet, und zum anderen in diesem ROM eine

Interpreter-Routine aufgerufen, die aus einer angegebenen Zeilennummer den physikalischen Anfang dieser Zeile berechnet. Wenn die Zeile gefunden wurde, wird sie an einer Systemadresse mit Namen DATAZEIGER abgelegt, anderfalls gibt's eine normale Fehlermeldung.

Die Basic-Interpreter Adressen sind beim CPC-664 / 6128:

```
      für "suchzl" &E861 (664)¦ &E85C (6128)  statt  &E79A
      für "datazg" &AE17 (664/6128)           statt  &AE30.
```

```
                6960 *E
                6970 ¦Befehl RESTORE
                6980 ¦
A5D8  CD00B9    6990 restor: call basrom      ¦oberes ROM zuschalten.
A5DB  DD5601    7000         ld   d,(ix+1)   ¦Zeilennummer nach DE
A5DE  DD5E00    7010         ld   e,(ix+0)   ¦Basic-Zeile DE suchen
A5E1  CD5CE8    7020         call suchzl     ¦Adresse! Basiczeilen-
A5E4  2B        7030         dec  hl         ¦beginn berichtigen und
A5E5  2217AE    7040         ld   (datazg),hl ¦als neuen DATA-Zeiger
A5E8  C9        7050         ret             ¦ablegen.
```

Befehl OVER1/0

```
¦OVER1      schaltet das Überschreiben ein.
¦OVER0      schaltet das Überschreiben aus.
```

Mit dem ersten Befehl können Sie Zeichen übereinander legen, ohne das untere zu löschen,der zweite Befehl stellt den Normalzustand wieder her.

Befehl TASTE

```
¦TASTE
```

Warten auf einen Tastendruck. Ersetzt die Basic-Befehlsfolge WHILE INKEYS.....u.s.w..
Es wird nur eine ROM-Routine verwendet und deshalb zweigt das Kommando direkt von der Sprungtabelle ins ROM ab.

Befehl INVERSE

```
¦INVERSE
```

Vertauscht bis zum erneutem Aufruf die Vorder- und die Hintergrundfarbe, hier wird ebenfalls nur eine ROM-Routine verwendet.

```
                7060 *E
                7070 ¦Befehle OVER0/1
A5E9  AF        7080 over0:  xor  a
A5EA  C39FBB    7090         jp   setgnd
A5ED  3D        7100 over1:  dec  a
A5EE  C39FBB    7110         jp   setgnd
                7120 ¦
A5F1            7130         end
                7140 ¦---------------------

Pass 2 errors: 00
```


Hardware-Erweiterungen

Dieser Teil wendet sich an die Bastler unter den CPC'lern. Es werden nacheinander folgende Schaltungsbeschreibungen und wenn nötig die Treibersoftware dazu vorgestellt:

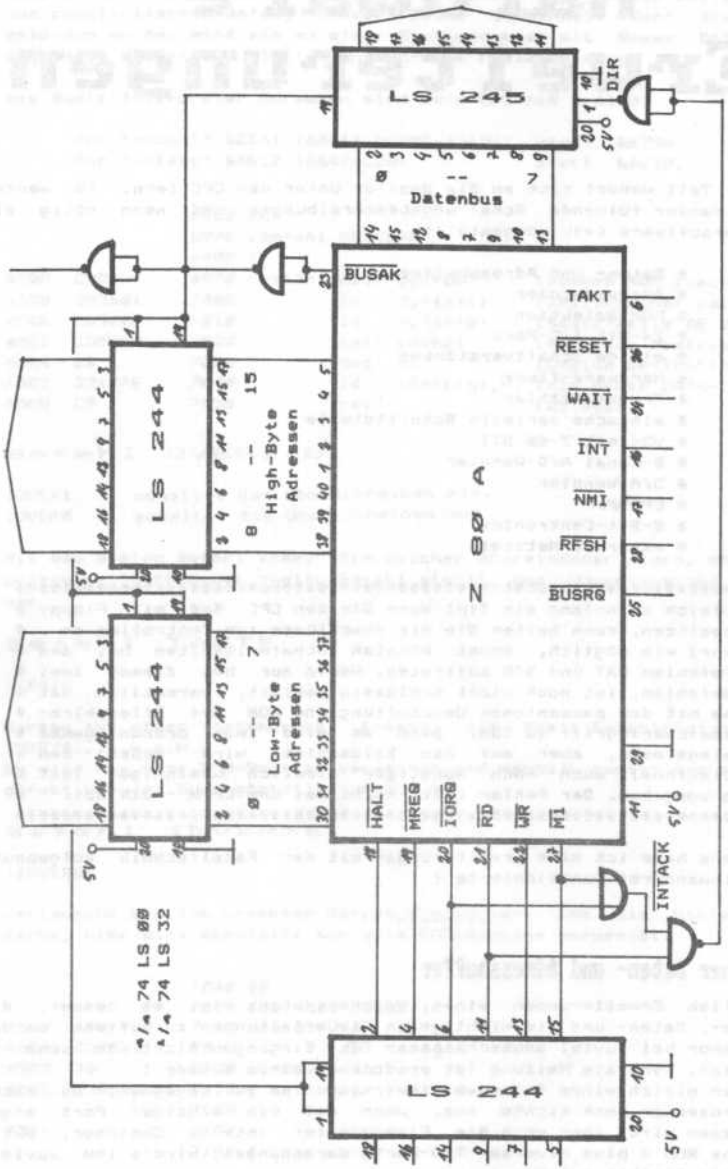
* Daten- und Adressbuffer
* Adressdecoder
* ROM-Selektion
* 24-Bit I/O-Port
* einige Schaltverstärker
* Hardware-Timer
* Frequenzzähler
* einfache serielle Schnittstelle
* V24 mit Z-80 STI
* 8-Kanal A/D-Wandler
* D/A-Wandler
* EPROMer
* 8-Bit-Centronics
* externes Netzteil

```
*****************************************************************
* Gleich am Anfang ein Tip: Wenn Sie den CPC 464 mit Floppy *
* besitzen, dann halten Sie die Anschlüsse zum Controller so  *
* kurz wie möglich, sonst könnten Schwierigkeiten bei den *
* Befehlen CAT und DIR auftreten. Warum nur bei diesen zwei *
* Befehlen, ist noch nicht schlüssig geklärt, vermutlich hat *
* es mit der pausenlosen Umschaltung vom ROM auf Bildschirm- *
* speicherzugriff zu tun. Denn es wird zwar ordnungsgemäß *
* eingelesen, aber auf dem Bildschirm wird außer dem *
* Diskinhalt auch noch sonstiger ziemlich unsinniger Text *
* ausgegeben. Der Fehler tritt nicht bei dem CP/M - DIR auf.  *
*****************************************************************
```

Übrigens habe ich alle Erweiterungen mit der Fädeltechnik aufgebaut, was einwandfrei funktionierte !

Interner Daten- und Adressbuffer

Bei allen Erweiterungen eines Rechnersystems ist es besser, die Adress-, Daten- und die wichtigsten Steuerleitungen zu buffern, um den Prozessor bei zuviel angeschalteten TTL- Eingängen nicht zu sehr zu belasten. Denn als Heizung ist er doch etwas zu schade !
Um aber gleich einem falschem Mißverständniss zu begegnen, es macht dem Prozessor noch nichts aus, wenn mal ein einziger Port angeschlossen wird, aber wenn die Floppy, der interne Speicher, evtl. externe ROM's plus diverser I/O-Ports daranhängen, wird's ihm zuviel. Also ein Treiber muß her !



Da gibt's zunächst einmal eine einfache Schaltung, die aber einen Eingriff in den Rechner -also Vorsicht wegen der Garantie !!- erfordert, und eine etwas kompliziertere, die außen angeschaltet wird.

Die erste Möglichkeit, die ich Ihnen nicht vorenthalten will, ist die: Sie nehmen den Prozessor aus der Fassung, geben ihm und den zusätzlichen Pufferbausteinen eine kleine Extraplatine und verbinden das ganze mit 4Ø-poligen IC - Steckern zum Anquetschen mit der eigentlichen Prozessorfassung. Nicht in jedem CPC-Gehäuse ist aber dafür noch Platz, besonders wenn Sie einen mit Abschirmblech um die Platine besitzen.
Der Vorteil eines solchen Eingriffs ist die einfachere Beschaltung. Außer den vier unerlässlichen Treibern vom Typ 74 LS 245 und 74 LS 244 sind nur noch zwei IC's erforderlich.

Gegenüberliegend finden Sie den Schaltplan dazu:

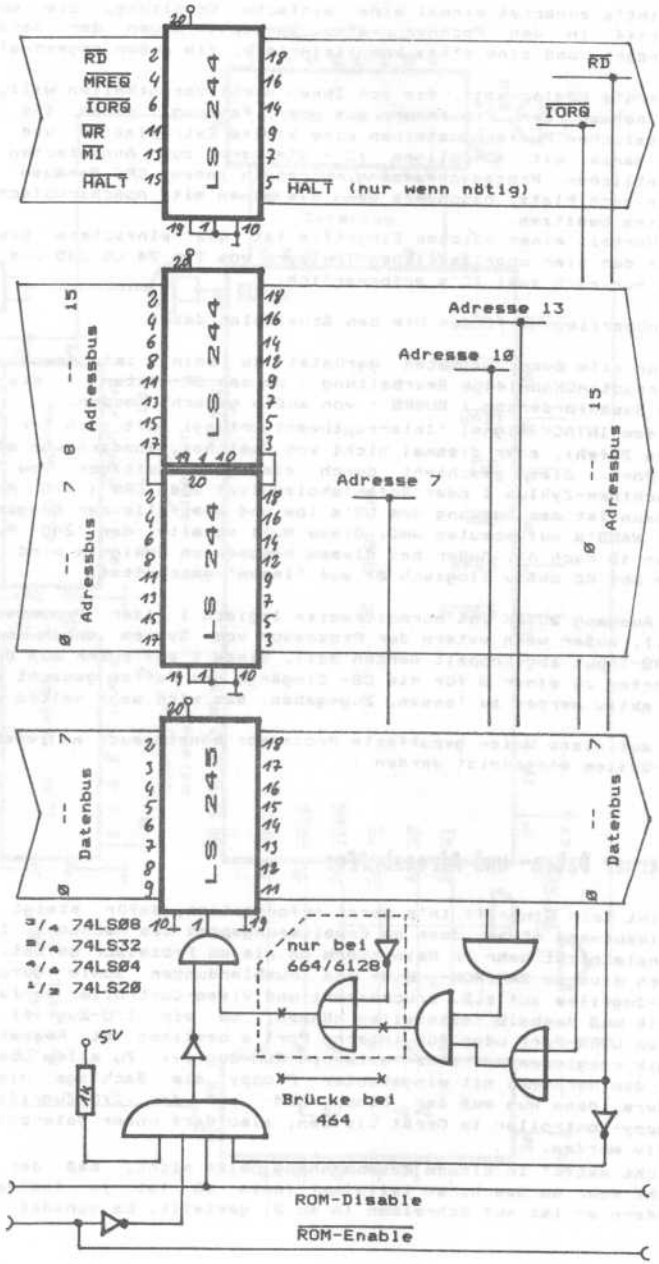
Um für alle Eventualitäten gerüstet zu sein, ist sowohl an eine INTerrupt-ACKnowledge Bearbeitung (um das OR-Gatter) als auch an eine Busanforderung (BUSRQ) von außen gedacht worden.
Bei dem INTACK-Signal (Interruptbeantwortung) holt sich der Prozessor einen Befehl, aber diesmal nicht vom Speicher, sondern von einem Z-8Ø I/O-Port. Dies geschieht durch ein gleichzeitiges low auf M1 (Maschinen-Zyklus 1 oder Befehlsholzyklus) und IORQ (I/O Anforderung). Dann ist der Ausgang des OR's low und ebenfalls der Ausgang des mit zwei NAND's aufgebauten AND. Diese Null schaltet den '245- Puffer auf lesen (B nach A). Außer bei diesem besonderen Ereignis wird der '245 noch bei RD aktiv (logisch Ø) auf "lesen" geschaltet.

Der Ausgang BUSAK ist normalerweise logisch 1 (der Prozessor arbeitet.), außer wenn extern der Prozessor vom System durch eine Ø am BUSRQ-Input abgekoppelt werden soll. Diese 1 auf BUSAK muß durch einen Inverter zu einer Ø für die CS- Eingänge der Puffer gemacht werden, um sie aktiv werden zu lassen. Zugegeben, das wird sehr selten vorkommen.

Der auf diese Weise gepufferte Prozessor könnte auch an jedem anderen Z8Ø-System eingesetzt werden !

Externer Daten- und Adressbuffer

Es ist kein Eingriff in's Gerät erforderlich, dafür steigt aber der Logikaufwand etwas, denn am Erweiterungsport des Rechners liegen die Signale nicht mehr in Reinstform an wie am Prozessor selbst. Sie sind durch diverse RAM/ROM-, Ein- und Ausblendungen sowie durch interne I/O-Zugriffe auf z.B. Druckerport und Video-Controller verfälscht. Die Logik muß deshalb feststellen können, ob ein I/O-Zugriff an einen neuen USER-Port oder für interne Port's bestimmt ist. Ferner muß diese Logik reagieren auf einen externen ROM-Zugriff. Zu allem Überfluß ist bei den Rechnern mit eingebauter Floppy die Sachlage wieder etwas anders, denn nun muß der Floppy-ROM- und der I/O-Zugriff auf den Floppy-Controller im Gerät bleiben, also darf unser Datenpuffer nicht aktiv werden.
"Nicht aktiv" in diesem Zusammenhang heißt nicht, daß der gute 245 nicht mehr am Geschehen teilnimmt (sein CS ist ja festverdrahtet), sondern er ist auf Schreiben (A zu B) gestellt. Es schadet ja nicht,



wenn er alle Daten des CPU's nach "außen" bringt, die externen Bausteine beachten diese Daten erst, wenn ihre Chipselects ebenfalls aktiv sind.

Diese Beschaltungsart des CS-Eingangs geschah aus einem guten, nämlich aus einem "zeitlichen" Grund. Die Besitzer des CPC 464 hätten sich sonst gewundert, wenn sie den Floppy-Motor nicht mehr ausschalten könnten !

Warum, will ich kurz erklären.
Dazu wäre es am besten, wenn Sie den Vorgang anhand des Laufwerk-Schaltplanes verfolgen könnten.

In dieser Beschaltungsart wird der CS-Anschluß des '245 erst nach diversen Gatterlaufzeiten aktiv, was zwar allen sonstigen Schaltungseinheiten nicht schadet (Die Signallaufzeiten sind bei typischen Prozessor-Ports schon eingeplant.), aber eben einem kleinen vorwitzigen TTL-Flip-Flop im Controller des Laufwerks das den Motor ein-/ausschaltet.
Dieses Flip-Flop vom Typ 74 LS 74 ist positiv Flankengetriggert und reagiert deshalb sauer, nämlich gar nicht, auf eine zu frühe Anstiegsflanke (von low nach high). Denn das Signal IOWR, welches am Clock-Eingang des FF's liegt, kommt früher als das Datenbit Ø (durch den '245 verzögert), das am D-Input eingelesen wird. So wird bei der Schaltflanke immer noch der Tri-Stat-Pegel eingelesen, was einem High gleichkommt !

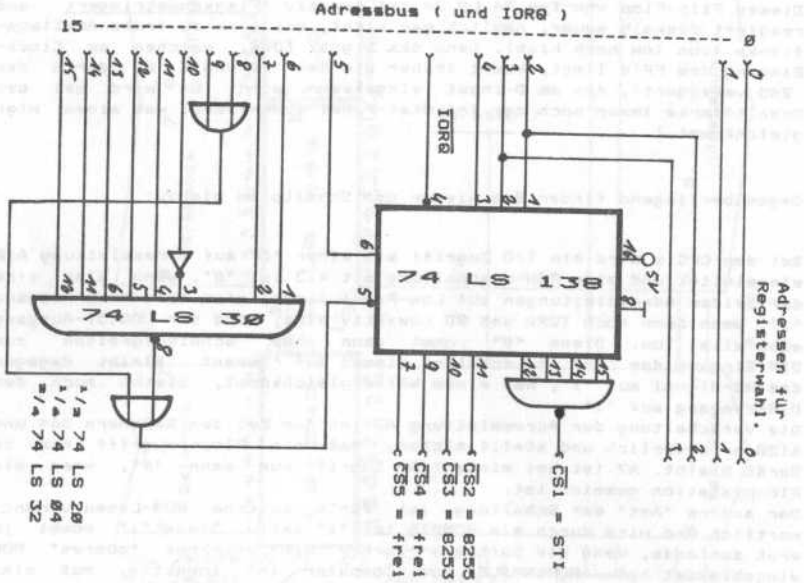
Gegenüberliegend finden Sie wieder den Schaltplan hierzu:

Bei den CPC's wird ein I/O Zugriff mit einer "Ø" auf Adressleitung A1Ø eingeleitet und eine ROM-Umschaltung mit A13 ist "Ø". Wenn also eine der beiden Adressleitungen auf Low-Pegel liegt, wird der NAND-Ausgang "1". Wenn dann noch IORQ und RD Lowaktiv sind, wird der OR(3)-Ausgang ebenfalls low. Diese "Ø" kommt dann ohne Schwierigkeiten zum DIR-Eingang des '245 und schaltet diesen auf "lesen". Bleibt dagegen das RD-Signal auf "1", was einem WRite gleichkommt, bleibt auch der DIR-Eingang auf "1"
Die Verschaltung der Adressleitung A7 ist nur bei den Rechnern 664 und 6128 erforderlich und stellt sicher, daß ein Floppyzugriff nun im Gerät bleibt. A7 ist bei einem I/O- Zugriff nur dann "Ø", wenn die Floppystation gemeint ist.
Der andere "Ast" der Schaltung ist für's externe ROM-Lesen verantwortlich und wird durch ein ROMDIS ist "1" aktiv. Diese "1" kommt ja erst zustande, wenn die Software vorher ein externes "oberes" ROM eingeblendet hat. ROMENABLE vom Computer ist lowaktiv, muß also invertiert werden, wenn die AND- Bedingung des Vierfach-AND's erfüllt sein soll. Wenn dann noch RD aktiv ("Ø") wird, ist der ROM-Zugriff perfekt und die Bytes rollen.
Der andere Teil der Schaltung besteht nur noch aus den Adress- und Steuerleitungstreibern, was keiner besonderen Erklärung mehr bedarf.

Adressdecoder

Bei den CPC's geschieht jeder I/O-Zugriff mit einer echten! 16-Bit Adresse. Dabei wird das B-Register als High-Byte und das C-Register als Low-Byte interpretiert. Bei dem Maschinen-Befehl "OUT (C),A" ist also unsichtbar auch das B-Register beteiligt und darf deshalb nicht anderweitig benutzt werden. Aus diesem Grund reduziert sich auch der Z80-Befehlsatz etwas. Die Schleifen-I/O-Kommandos (z.B. INIR, OTIR, IND,OTD, etc.) fallen deshalb aus. Die einzigen I/O-Befehle, die Sie noch verwenden können, sind OUT (C),r und IN r,(C), wobei r irgendein Register bedeutet, bei IN r,(C) darf r aber nicht das B-Register sein!

Dies gehörte zwar nur in zweiter Linie zum Thema Adressdecoder, sollte aber an dieser Stelle erwähnt werden.



Die für Erweiterungen freigegebenen Adressen lauten:

F8xx F9xx FAxx FBxx

wobei bis jetzt nur das High-Byte berücksichtigt wurde. Das Low-Byte kann die Werte E0 - FF annehmen, wobei FF nicht verwendet werden sollte, es dient für einen Peripherie-Reset.

Unser Adressdecoder wertet die Adressen F8E0 - F8FF aus und stellt folgende Adressen für die in diesem Buch gezeigten Erweiterungen zur Verfügung:

F8E0	-	F8EF	für die 16 Register des STI
		F8F0	8255 A-Port
		F8F1	8255 B-Port
		F8F2	8255 C-Port
		F8F3	8255 Statusregister
		F8F4	8253 Zähler 0
		F8F5	8253 Zähler 1
		F8F6	8253 Zähler 2
		F8F7	8253 Steuerwort
F8F8	-	F8FE	frei für weitere Zusätze.

** Schaltplan 3 **

Wie schon vorher einmal angedeutet, ist bei den für uns freien I/O-Zugriffen im High-Byte die Adressleitung A10 immer low. Die beiden Lows auf A8 und A9 ergeben sich durch den von mir gewählten Adressbereich (F8xx). Die auf diese Weise aufbereitete Adresse gibt mit A5 und IORQ den Decoder frei.
A0 und A1 sind nur durchgeschleift, da die meisten Port's intern schon vier Register haben, die durch diese zwei Bits angesteuert werden. Die Ausnahme von der Regel bildet der verwendete Serielle Schnittstellen-Baustein Z80-STI, der 16 interne Register sein eigen nennt. Deshalb mußten auch die Decoderausgänge Y0 bis Y3 wieder mit einem AND verknüpft werden, um das CS-Signal für diesen Baustein über den gesamten Bereich anstehen zu lassen.

Erweiterungs-ROM selektieren

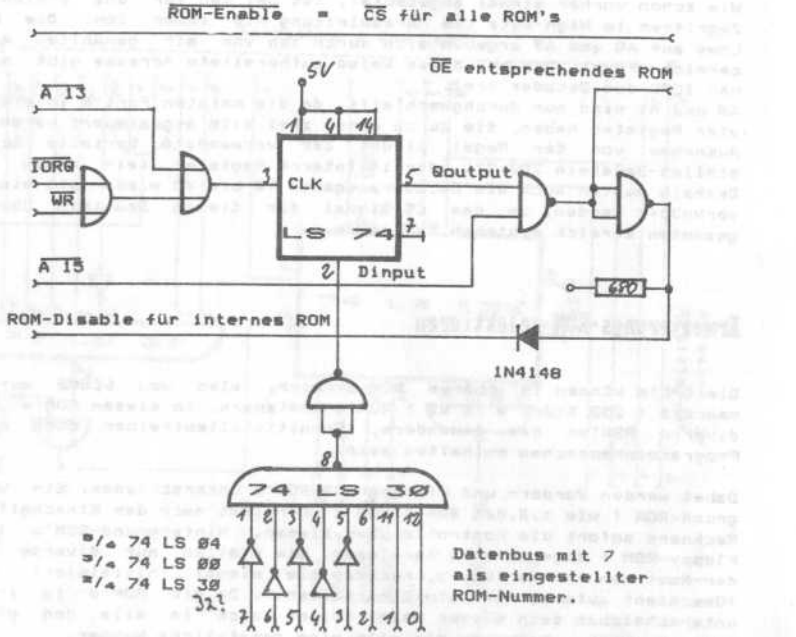
Die CPC's können im oberen ROM-Bereich, also von &C000 aufwärts, mehrere (252 Stück a 16 kB) ROM's ansteuern. In diesen ROM's können diverse RSX'en des Anwenders, Schnittstellentreiber oder weitere Programmiersprachen enthalten sein.

Dabei werden Vorder- und Hintergrund-ROM's unterschieden. Ein Vordergrund-ROM (wie z.B.das Basic-ROM) übernimmt nach dem Einschalten des Rechners sofort die Kontrolle über diesen. Hintergrund-ROM's (z.B. Floppy-ROM) dagegen sind harmloser, sie stellen nur diverse Anwender-Routinen zur Verfügung, nachdem sie einmal initialsiert wurden (Geschieht automatisch beim Einschalten.). Da die ROM's ja irgenwie unterscheidbar sein müssen -sie teilen sich ja alle den gleichen Adressbereich-, bekommen sie alle eine zusätzliche Nummer.

Für Hintergrund-ROM's können die Nummern 0-7 verwendet werden, es können damit also bis zu acht Hintergrund-ROM's gleichzeitig am System "hängen". Die Vordergrund-ROM's können von 0-252 durchnumeriert sein, also wesentlich mehr ! Aber seien wir doch ehrlich, wer will oder muß schon soviele ROM's gleichzeitig im Zugriff haben. Die Hintergrund-ROM's sind in den meisten Fällen sowieso die interessanteren und flexibelsten, was die Verwendung betrifft.
Bevor wir nun zur hardwareseitigen Lösung des Umschaltens kommen, möchte ich noch kurz den "Kopf" eines Hintergrund-ROM's besprechen. Er kann wie folgt aussehen :

```
        ORG #C000
        DEFB 1          ;Hintergrund-ROM Kennzeichen
        DEFB 0,0,0      ;für Versionsnummern etc.
        Defw Namen              ;Ähnlicher Aufbau wie
        JP Routine1             ;RAM-RSX. Siehe Basicerweiterung.
        JP Routine2
Namen:  DEFM "NAME",'1'+#80
        DEFM "NAME",'2'+#80
        DEFB 0          ; Ende-Kennzeichen Tabelle
```

Noch nicht berücksichtigt ist bis jetzt die eigentliche ROM-Nummer, welche immer das von Ihnen gewählte ROM aktiviert. Dies ist aber die Aufgabe der Hardware, welche ich nun anhand eines Schaltplanes vorstellen will. Die eingestellte ROM-Nummer, gleichbedeutend mit ROM-Select-Byte, ist in diesem Beispiel festverdrahtet auf 7.



Bei einem I/O-Schreibzugriff mit A13 = "0" wird der Clockinput des Flip-Flops getriggert. Zu diesem Zeitpunkt wird dann das auf "wahr=1" und "unwahr=0" mit dem 8-fach-(N)AND reduzierte Datenwort eingelesen. Sollte das entsprechende ROM eingeschaltet werden, erscheint nun am Q-Ausgang des FF's eine "1". Diese "1" wird mit A15 verknüpft -es soll ja nur bei einem Zugriff auf das obere ROM das Basic-ROM ausgeblendet werden !- und gibt dann den OE-Eingang (Output-Enable) des externen ROM's frei. Gleichzeitig wird durch eine weitere Invertierung eine "1" als ROMDIS-Signal an's interne Basic-ROM gesendet und dieses ausgeblendet.

Mit einer entsprechenden Inverterkonfiguration an den Eingängen des 8-fach-NAND's kann ein ROM-Select-Byte von 0 bis 7 (für Hintergrund-ROM's) eingestellt werden, wobei ein eventuelles Floppy-ROM die Nummer 7 hat.

24-Bit-I/O-Port

Die Voraussetzung, um den CPC auch als Steuergerät für diverse Erweiterungen einsetzen zu können, ist ein User-Port.
Der hier verwendete 8255 stammt zwar noch aus der 8080-Ära, aber er hat so manche Vorzüge gegenüber der Z80-PIO, wenn der vektorisierte Interrupt nicht gebraucht wird. Sein Merkmal ist vor allem der 24 Bit breite Ein-/Ausgabeport und die leichtere Programmierung.

Das IC kann in drei verschiedenen Modi betrieben werden.

Mode 2:

Port A arbeitet dabei birektional und 5 Bits des C-Ports dienen jetzt als Handshakesignale.

Mode 1:

Den Ports A und B werden jeweils 4 Bit des C-Ports als Handshakesignale zur Verfügung gestellt. Die beiden Ports A und B können jeweils als Ein- und Ausgänge betrieben werden.

Die bei einfachen Anwendungen am meisten verwendete Betriebsart ist Mode 0. Da er ausschließlich bei den in diesem Buch gezeigten Anwendungen verwendet wird, gehe ich näher darauf ein. Wenn Sie sich jedoch eingehender mit den anderen Moden auseinandersetzen wollen, empfehle ich Ihnen entweder das entsprechende Datenblatt oder die im Anhang vorgestellte Literatur.

Mode 0:

Dabei stellt der A- und B-Port je einen 8-Bit breiten Port dar, der C-Port kann in zwei Gruppen zu je 4 Bit's aufgeteilt werden. Die Datenrichtung des A- und B-Ports und der zwei Gruppen des C-Ports kann dabei jeweils getrennt festgelegt werden.

Der dabei als Statuswort auszugebende Wert ist wie folgt:

Statuswort dez:	hex:	bin:	<>Port: Bit:	A 0-7	B 0-7	C 4-7	C 0-3
128	80	10000000		out	out	out	out
129	81	10000001		out	out	out	in
130	82	10000010		out	in	out	out
131	83	10000011		out	in	out	in
136	88	10001000		out	out	in	out
137	89	10001001		out	out	in	in
138	8A	10001010		out	in	in	out
139	8B	10001011		out	in	in	in
144	90	10010000		in	out	out	out
145	91	10010001		in	out	out	in
146	92	10010010		in	in	out	out
147	93	10010011		in	in	out	in
152	98	10011000		in	out	in	out
153	99	10011001		in	out	in	in
154	9A	10011010		in	in	in	out
155	9B	10011011		in	in	in	in

Wie aus der Tabelle zu ersehen ist, schaltet z.B.der Befehl:

```
OUT &F8F3,128
```

alle Ports auf Ausgabe, und genau so einfach lassen sich auch alle anderen Kombinationen einstellen.
Automatisch wird dabei auch der Mode Ø programmiert. Die Datenbits D2, D5 und D6, die dabei alle auf "Ø" gesetzt sind, sind dafür verantwortlich.
Eine andere Variante, Einfluß auf die Ausgabebits des C-Ports zu nehmen, ist das Einzelbit zu setzen oder zu löschen. Aber wie gesagt, diese Möglichkeit besteht nur für Port C.

Das als Statuswort auszugebende Byte kann dann folgende Formen annehmen:

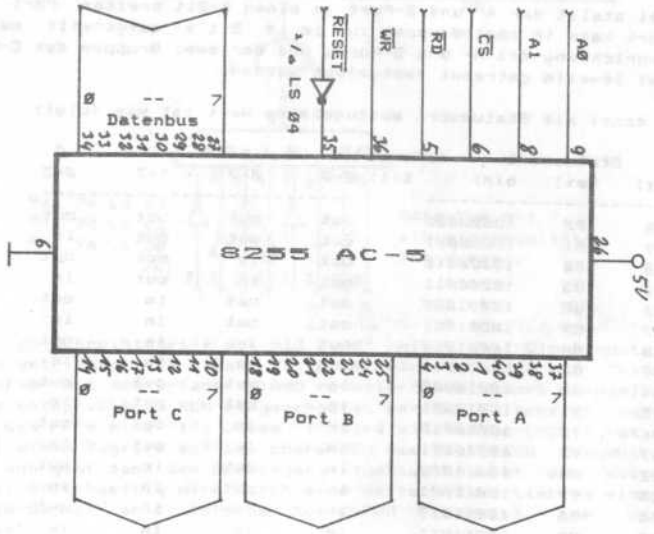
Bitnr.: Ø Ø = Bit löschen
1 = Bit setzen

		7 6 5 4 3 2 1 Ø

"	1	1 Ø 1 Ø 1 Ø 1 Ø
"	2	1 1 Ø Ø 1 1 Ø Ø
"	3	1 1 1 1 Ø Ø Ø Ø
"	4	x
"	5	x
"	6	x
"	7	Ø = Marke für Bit Set/Reset Mode.

-- Tabelle zur Bitauswahl --
(Es kann also immer nur ein Bit gleichzeitig beeinflußt werden.)

Der Schaltplan der Parallel-Schnittstelle:



Schaltverstärker

Die Belastungsfähigkeit der Ausgänge reicht für einen! TTL-Eingang aus, was ca. 1,6 mA im Lowzustand entspricht. Wird mehr Strom gefordert, so müssen Ausgangsverstärker verwendet werden.

Bis ca. 66 mA (low) und bei 5 V reicht ein TTL- Bustreiber der Sorte LS 24Ø (Inverter) oder LS 241 aus.
Bis ca. 4Ø mA und bis zu einer maximalen Lastkreisspannung von 3Ø V ist das TTL-IC 74Ø6 mit offenem Kollektor verwendbar.

Bis ca. 5ØØ mA und 5Ø V! (Gesamtverlustleistung des IC's beachten !) im Lastkreis kann man den IC ULN 2ØØ4 (=L2Ø4) verwenden. Dieses IC verfügt an jedem Ausgang über eine Freilaufdiode und ist somit sehr geeignet zur Ansteuerung von Relais.

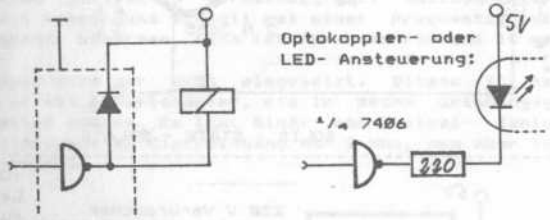
Bei Lasten darüber hinaus und induktiven Verbrauchern (Motore und starke Relais !) sollte man -um unerwünschte Masseverkopplungen (Schaltspitzen !) zu vermeiden- zu einem Optokoppler mit Schaltverstärker greifen.

****
Zur Ansteuerung eines 22Ø V Kreises ist ein Optokoppler unerläßlich !! Da Isolierung das halbe Leben ist, sollte man im Umgang mit der Netzspannung große Vorsicht walten lassen, lieber etwas zuviel als gar keine !!
Weshalb ich auch ein fertiges Opto-Relais im anschließenden Vorschlag verwende.
****

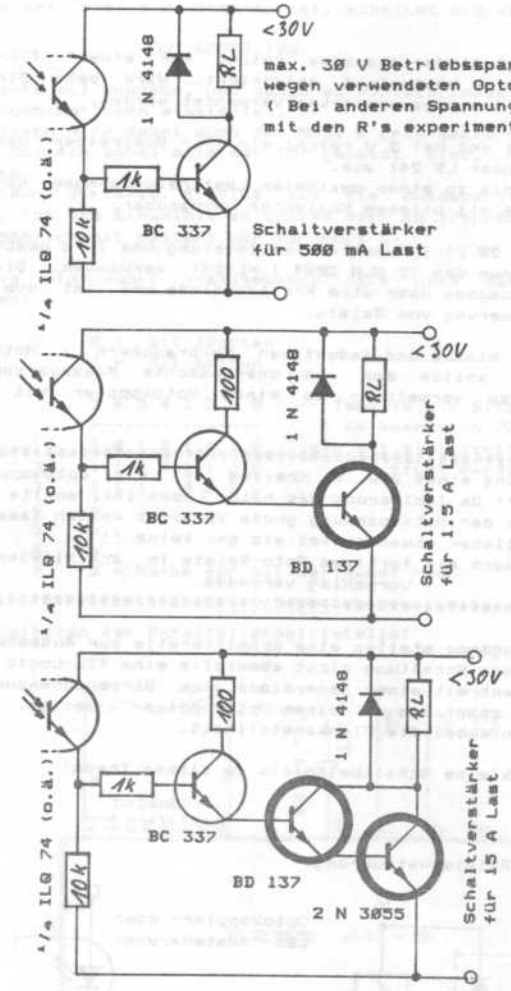
Aber auch Port-Eingänge stellen eine Schnittstelle zur Aussenwelt dar. Wenn die ansteuernde Schaltung nicht ebenfalls eine TTL-Logik besitzt, sollte man zumindest mit einer Zenerdiode plus Strombegrenzungswiderstand oder gleich ebenfalls mit einem Optokoppler arbeiten. Schmitttrigger verbessern dabei die Flankensteilheit.

Es folgen einige kleine Schaltbeispiele zu diesen Thema:

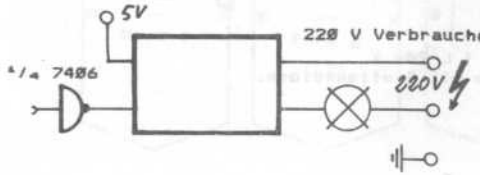
Typische Relaisansteuerung:



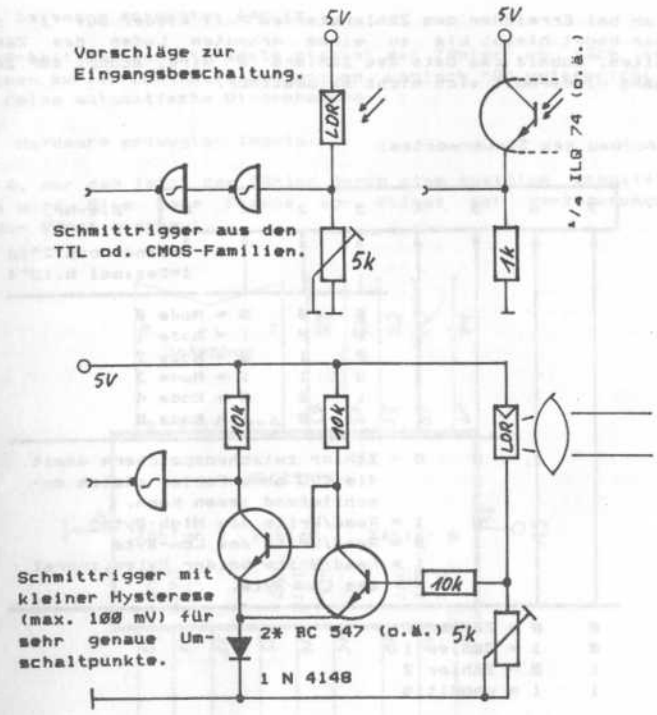
1/6 ULN 2ØØ4 (L 2Ø4)
Alle Ausgänge mit Freilaufdiode.



SOLID - STATE - RELAIS



(Optoelektronisches
Lastrelais m. Null-
durchgangsschalter)
Input:3-30V; 30mA
Strombegrenzt !
Outp.:24-280V;2.5A
Rim-Best.-Nr.:32-36-380



Hardware-Timer

Wenn man mit einem Computer Zeiten in den Griff bekommen will, wird's kritisch. Solche kitzligen Angelegenheiten sollt man lieber einem eigens dafür geschaffenen Baustein übertragen, der unabhängig von der CPU und ihrem zeitlichen Verhalten ist. Sollten Sie mal in die Verlegenheit kommen und schnell mal einen Frequenzzähler oder einen Impulsgenerator brauchen, dann ist der beschriebene IC genau richtig.

Als Zeitgeber wird der 8253 eingesetzt. Dieses IC hat drei verschiedene 16-Bit Abwärtszähler, die in sechs unabhängigen Betriebsarten arbeiten können. Es läßt Binär- und Dezimal- Zählung zu, verkraftet allerdings am Clockeingang nur 2 Mhz, was aber in den meisten Fällen ausreicht.

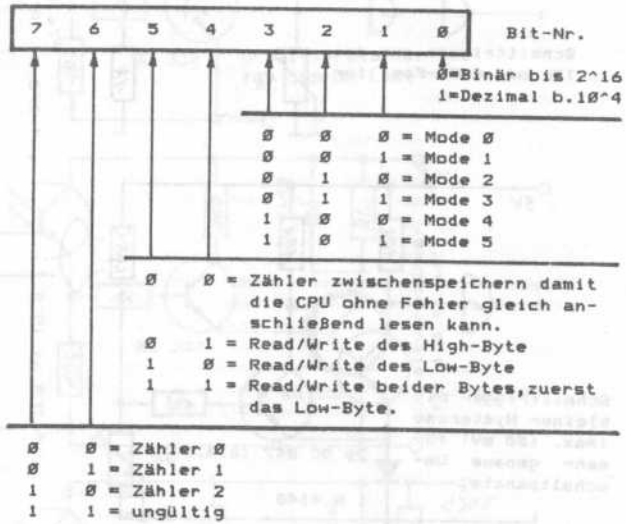
Die verschiedenen Betriebsarten:

Mode 0: Ausgang = "1" nach letztem Taktimpuls.

Der Zähler zählt sofort nach Erhalt eines Startwertes abwärts, mit der am Eingang anliegenden Taktrate. Während des Zählens ist der Ausgang

"Ø" um bei Erreichen des Zählerstandes Null wieder auf "1" zu gehen. Dieser Pegel bleibt bis zu einem erneuten Laden des Zählregister erhalten. Sobald das Gate des Zählers "Ø" wird, stoppt der Zähler. Der Vorgang wiederholt sich nicht automatisch.

Der Aufbau des Steuerwortes:



Mode 1: Monoflop

Bei einer ansteigenden Taktflanke am Gate eines Zählers wird der Zählvorgang ausgelöst, der Ausgang wird solange "Ø", bis der Zähler Null erreicht hat, um dann wieder in den Ruhezustand (= "1") überzugehen. Eine weitere Schaltflanke am Gate triggert den Zähler erneut. Der Vorgang wiederholt sich nicht automatisch.

Mode 2: Asymetrischer Teiler durch N (Generator).

Der Zähler teilt das angelegte Taktsignal fortlaufend durch seinen programmierten Wert. Eine negative Flanke am Gate stoppt sein Arbeiten, und eine positive Flanke startet den Vorgang erneut. Der übermittelte Startwert wird ERST NACH DER ERSTEN POSITIVEN TAKTFLANKE am Clock-Eigang wirklich geladen.

Mode 3: Symetrischer Teiler durch N (Generator).

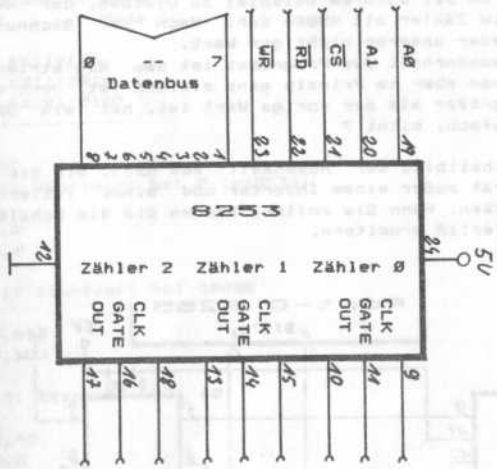
Wie Mode 2 nur daß der Ausgangsimpuls genausolange low wie high ist. Die Zählkonstante muß dabei durch zwei teilbar sein.

Mode 4: Software erzeugter Impuls.

Nach dem Erhalt der Registerwerte startet der Zähler, um nach seinem Ablauf einen kurzen Lowimpuls auszugeben. Logisch "Ø" unterbricht den Vorgang. Keine automatische Wiederholung.

Mode 5: Hardware erzeugter Impuls.

Wie Mode 4, nur daß jetzt der Zähler durch eine positive Schaltflanke ausgelöst wird. Eine neue Flanke vor Ablauf der Verzögerungszeit startet den Vorgang erneut.



Frequenzzähler

Als Schmankerl und als Beispiel, wie die verschiedenen Betriebsarten des 8253 verknüpft werden können, gleich eine praktische Anwendung. Außerdem können Sie sich noch einige Scheinchen für einen "echten" Zähler sparen !

Errinnern wir uns, wie ein Frequenzzähler arbeitet: Da ist zuerst einmal die allgegenwärtige Zeitbasis für die internen Abläufe und vor allem für die Torzeit. Diese Torzeit läßt nur z.B. eine Sekunde lang Impulse an einen Zähler durch. Der Inhalt des Zählers ist dann nach Ablauf der Torzeit direkt ein Maß für die angelegte Frequenz, nach der höchst-mathematischen Gleichung 2ØØØ Impulse ist 2ØØØ Hertz.

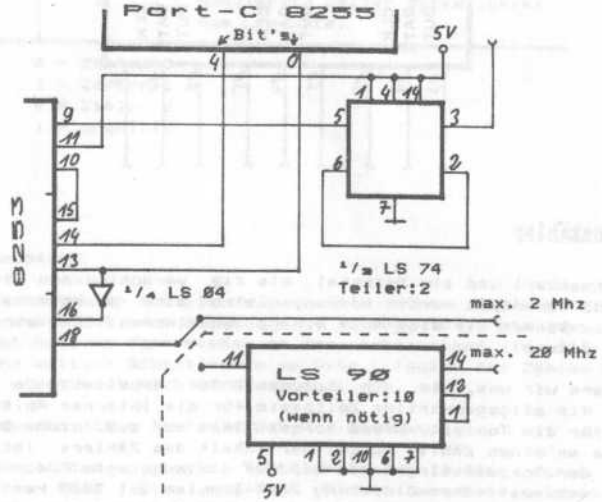
Da unser Zähler nur bis maximal 65535 zählen kann, ist dafür Sorge zu tragen, daß kein Überlauf entsteht oder -was noch viel besser ist- die einzelnen Überläufe zu sammeln.
Einen Überlauf zu verhindern ist leicht, es muß nur die Torzeit entsprechend verkürzt werden, was aber die Auflösung stark beeinträchtigt. Die Meßgenauigkeit unserer zweiten Methode bringt dagegen eine Auflösung von einem Hertz bei 2 Mhz !

Das "Sammeln" von Null-Durchgängen wird durch einen angehängten Softwarezähler erledigt. Ein kleines Rechenexempel:

Werden eine Million Impulse in der Sekunde gezählt, dann "passieren" 15 Nulldurchgänge und es wird ein Rest von 16960 gebildet, da sich ein Megahertz nicht durch 65536 ohne Rest teilen läßt. Das Programm muß nun die Nulldurchgänge wieder mit 65536 multiplizieren und den Rest dazuzählen, um auf unsere gemessene Frequenz von 1 Mhz zu kommen.

So weit, so gut - ein Häkchen ist noch bei der Geschichte. Die Zähler des 8253 arbeiten decrementierend, also anders herum. Das läßt sich aber leicht ausgleichen durch eine Subtraktion des Restes von dem Maximalwert. Um bei unserem Beispiel zu bleiben, der Rest von 19650 stellt sich im Zähler mit 45886 dar. Nach der Rechnung 65536-45886 haben wir wieder unseren richtigen Wert.
Noch eine Besonderheit des Programms ist das Registrieren der Nulldurchgänge, was aber im Prinzip ganz einfach ist. Wenn der eingelesene Zählerstand größer als der vorige Wert ist, hat ein Überlauf stattgefunden. Einfach, nicht ?

Zuerst das Schaltbild der "Außenwelt" des 8253. Wie Sie sehen, besteht das ganze Gerät außer einem Inverter und einem Teiler-Flip-Flop nur aus Drahtbrücken. Wenn Sie wollen, können Sie die Schaltung noch mit einen Vorteiler:10 erweitern.



Dann das Programm: (Es muß ja nicht immer Maschinensprache sein !)


```
100 '************************
110 '*    Frequenzzaehler    *
120 '************************
130 CLS
140 '
150 'PIO 8255 initialisieren
160 OUT &F8F3,147
170 '-------------------------
180 '
190 'Die drei Zaehler des 8253
200 'mit dem Mode versorgen.
210 '(Die Daten duerfen spaeter folgen!)
220 'Zaehler 0 und 2: Mode 2
230 'Zaehler 1      : Mode 1
240 '
250 OUT &F8F7,&X11010100
260 OUT &F8F7,&X11100010
270 OUT &F8F7,&X10110100
280 '--------------------
290 '
300 'Die Daten:
310 'Zaehler 2: Startwert bei 0 (=65536)
320 '
330 OUT &F8F6,0
340 OUT &F8F6,0
350 '
360 'Zaehler 1: Startwert bei 50000
370 '
380 OUT &F8F5,&50
390 OUT &F8F5,&C3
400 '
410 'Zaehler 0: Startwert bei 40
420 '
430 OUT &F8F4,40
440 OUT &F8F4,0
450 '--------------------------------
460 '
470 'Monoflop triggern mit pos.Flanke
480 '
490 OUT &F8F2,&X10000
500 OUT &F8F2,0
510 '--------------------
520 '
530 'Kurze Verzoegerung bis 1.Taktimpuls
540 'eintraf.(Dann wird Zaehler erst geladen)
550 '
560 FOR n=1 TO 90:NEXT
570 '
580 '-----------------------------------------
590 'Zaehlschleife: 1 Sekunde Highpegel am Gate
600 '                von Zaehler 2.
610 '
620 WHILE INP (&F8F2)=0
630 OUT &F8F7,&X10000000
640 lsb=INP (&F8F6):hsb=INP (&F8F6)
650 '
660 'Nulldurchgang ?
670 '
```

```
680 IF lsb+256*hsb>wert THEN nulldg=nulldg+1
690 wert=lsb+256*hsb
700 WEND
710 '-----------------------------------------
720 '
730 'Nulldurchgaenge und Restinhalt von Zaehler
740 'auswerten.
750 LOCATE 1,1
760 PRINT 65536*(nulldg-1)+(65536-wert)
770 wert=0:nulldg=0
780 GOTO 160
```

Eine zeitkritische Anmerkung zur Software:

Zuerst ein paar Feststellungen:

Durchlaufzeit der FOR-NEXT Schleife: 100 ms.
Durchlaufzeit der WHILE-WEND Schleife: 16 ms.
Schaltflanke bei 10 Hz-Signal spätestens nach 100 ms.
Bei 2 Mhz-Signal findet alle 32 ms ein Überlauf statt und bei 500 khz alle 130 ms.

In diesem Zusammenhang will ich auch das Vorhandensein der FOR-NEXT-Schleife erklären.
Da bei einer kleinen Eingangsfrequenz von z.B. 10 Hz, wie gesagt, die positive Flanke erst nach ca. 100ms auftreten kann, und damit erst der Startwert des Zählers geladen wird (siehe Erklärung von Mode 2), wird dieses Laden, durch das selbst in Basic wesentlich schnellere Abfragen als Nulldurchgang interpretiert, was aber absolut nicht stimmt.

Deshalb die Verzögerung von 100 ms.

Diese Schleife muß aber bei Frequenzen größer als 500 khz inaktiviert werden, um den ersten Nulldurchgang dieser höheren Meßfrequenz nicht zu verpassen (bei 2 Mhz immerhin alle 32 ms !).
Frequenzen unter 10 Hz sollten ohnehin über den Umweg der Periodendauer gemessen werden, um eine höhere Auflösung zu erreichen.

Und zum Abschluß deshalb gleich eine Anregung:

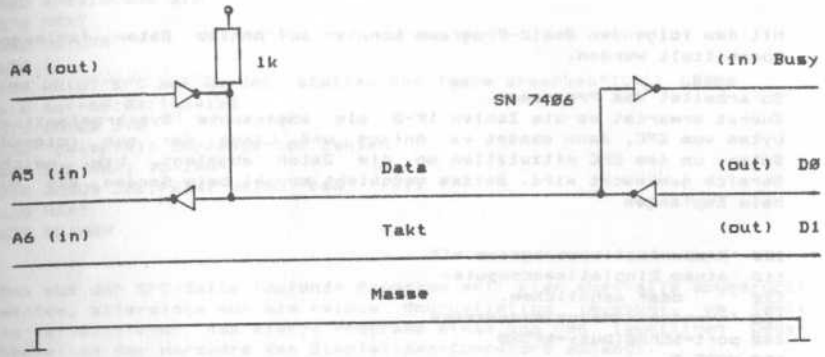
Einem Ausbau der Hard- und Software des Frequenzzählers zu einem Impuls/Pausen-, Perioden- und Drehzahlmessers steht eigentlich nichts im Wege, finden Sie nicht auch ?

Einfache synchrone serielle Schnittstelle

Wenn Sie z.B., wie ich, ein Ein-Platinen-Computerchen zusammengezimmert und ihm ein mehr oder weniger komfortables Monitorprogramm gespendet haben, dann kommt bestimmt einmal der Zeitpunkt, wo Sie Daten von und zu diesem Computer-Ableger übertragen wollen.
Sei es nun weil Sie Ihren Schneider CPC als komfortables Entwicklungssystem einsetzen, oder weil Ihr Zweiter? als Meßwerterfassungssystem arbeitet.
Eine V24 (oder RS 232c) ist hier aber fehl am Platz, erstens weil diese Hardware zu umfangreich und zweitens die Programmierung der "großen Schwester" ein Kapitel für sich ist, nicht nur in diesem Buch.

Ohne großen Aufwand an Hard- und Software läßt sich aber eine serielle synchrone (weil vom selben Takt abhängig.) Schnittstelle am Centronics-Port des Schneiders simulieren.

Die dabei nötigen zusätzlichen Teile zeigt folgendes Schaltbild der Schnittstelle. EPC soll im Folgenden die Abkürzung für Ein-Platinen-Computer sein.

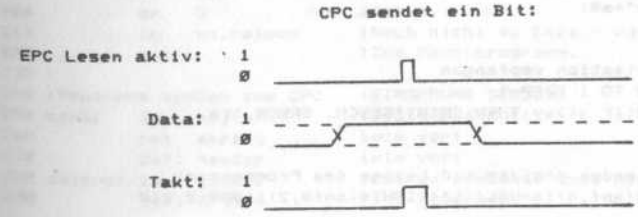


Das Taktsignal teilt dem EPC mit, wann er ein Datenbit zu senden hat, bzw. wann das Datenbit stabil anliegt, wenn der CPC sendet.

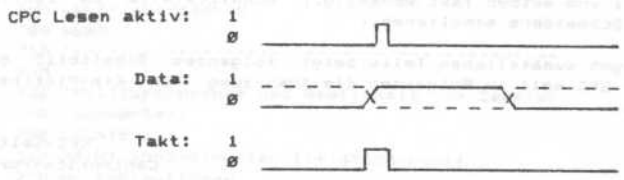
Der Schneider stellt bei dieser Anordnung und durch die anschließende Software, den höherwertigen Rechner, also einen Hostrechner dar. Der diesem Rechner folgende Computer ist damit zu einem Download-Rechner degradiert. Diese Rangfolge muß sein, um eindeutige Zustände auf der Schnittstelle zu schaffen.

Selbstverständlich müssen vorher beide Computer in den entsprechenden Modus gebracht werden, also EPC empfängt und CPC sendet oder umgekehrt, dazu ist es am Besten wenn der EPC, bevor der Schneider mit seinem Taktsignal die Schnittstelle aktiviert, in den entsprechenden Modus gebracht wird.
Nur so ist es möglich das keine Daten verloren gehen.

Folgendes Bild soll das Zeitverhalten verdeutlichen:



CPC wartet auf ein Bit:



Mit dem folgenden Basic-Programm konnten auf Anhieb Daten fehlerfrei übermittelt werden.

So arbeitet das Programm:
Zuerst erwartet es die Zahlen 10-0 als sogenannte Synchronisationsbytes vom EPC, dann sendet es Anfang und Länge der nun folgenden Daten, um dem EPC mitzuteilen wo die Daten abgelegt, bzw. welcher Bereich gewünscht wird. Beides geschieht sowohl beim Senden als auch beim Empfangen

```
100 'Kommunikationsprogramm mit
110 'einem Einplatinencomputer
120 '    oder aehnlichem.
130 '==========================
140 port=&EF00:busy=&F500
150 MODE 2
160 INPUT "S)enden oder E)mpfangen";i$
170 IF i$<>"s" AND i$<>"e" THEN 160
180 IF i$="s" THEN GOSUB 510 ELSE GOSUB 600
190 PRINT:GOTO 160
200 '
210 'Byte -a- ausgeben
220 a$=BIN$(a,8)
230 FOR n=1 TO 8
240 obyte=0
250 IF MID$(a$,n,1)="1" THEN obyte=1
260 OUT (port),obyte:OUT (port),obyte OR 2:OUT (port),obyte
270 NEXT n
280 RETURN
290 '
300 'Byte -a- empfangen
310 a$="00000000"
320 FOR n=1 TO 8
330 OUT (port),2:a=INP (busy) AND &X10000000:OUT (port),0
340 IF a THEN MID$(a$,n,1)="1"
350 NEXT n
360 a=VAL("&x"+a$)
370 RETURN
380 '
390 'Synchronisation empfangen
400 FOR sy=10 TO 1 STEP-1
410 GOSUB 300:IF a<>sy THEN PRINT"SYNCH. ERROR ";a
420 NEXT
430 '
440 'Header senden (Anfang und Laenge des Programmes)
450 anf$=HEX$(anf,4):a=VAL("&"+RIGHT$(anf$,2)):GOSUB 210
```



```
460 a=VAL("&"+LEFT$(anf$,2)):GOSUB 210
470 lge$=HEX$(lge,4):a=VAL("&"+RIGHT$(lge$,2)):GOSUB 210
480 a=VAL("&"+LEFT$(lge$,2)):GOSUB 210
490 RETURN
500 '
510 PRINT"EPC auf Empfang stellen und Taste druecken":CALL &BB06
520 anf=&8100:lge=100
530 GOSUB 390
540 'Beispiel: Sende Zahlen 1-100
550 FOR se=1 TO 100
560 a=se:GOSUB 210
570 NEXT
580 RETURN
590 '
600 PRINT"EPC auf Senden  stellen und Taste druecken":CALL &BB06
610 anf=&8100:lge=100
620 GOSUB 390
630 'Beispiel: Empfange 100 Zahlen
640 FOR em=1 TO 100
650 GOSUB 300:PRINT USING "### ";a;
660 NEXT
670 RETURN
```

Das auf der EPC-Seite laufende Programm soll hier ebenfalls abgedruckt werden, allerdings nur als reines Sourcelisting (Auszug), um damit zu verdeutlichen, daß dieses Programm etwas von den jeweiligen Gegebenheiten der Hardware des Einplatinen-Computers abhängt.

```
 10 ;ld hl,textxx laedt den
 20 ;Anfang eines Textes und
 30 ;rst string druckt ihn aus.
 40 ;Beide Routinen sind System-
 50 ;spezifisch.
 60 ;-------------------------
 70 ;Programm laden vom CPC     ;EINSPRUNG EMPFANGEN:
 80 load:   ld   hl,text10     ;Systembedingt:
 90         rst  string        ;Text laden und auf Display bringen
100         call header        ;hole Anfang u. Länge der Daten
110         ld   a,h
120         cp   #81           ;Unter 8100hex darf nichts geladen wer-
130         jr   nc,reloop     ;den, wegen Stack u. Systemdaten !!
140         ld   hl,#8100
150 reloop: call reada         ;Hole Byte nach Akku.
160         ld   (hl),a        ;Lege es am entspr. Speicherplatz ab.
170         inc  hl            ;Zeiger +1
180         dec  bc            ;Länge -1
190         ld   a,c           ;Prüfe auf Ende der zu übermittelnden
200         or   b             ;Daten.
210         jr   nz,reloop     ;Noch nicht zu Ende - Weiter.
220         ret                ;Zum Hauptprogramm.
230 ;
240 ;Programm senden zum CPC    ;EINSPRUNG SENDEN:
250 send:   ld   hl,text9      ;Systembedingt:
260         rst  string        ;wie vor:
270         call header        ;wie vor:
280 seloop: ld   a,(hl)        ;Das zu sendende Byte nach Akku,
290         call writea        ;und ausgeben.
```

```
300         inc  hl             ;Zeiger +1
310         dec  bc             ;Länge -1
320         ld   a,c
330         or   b              ;wie vor:
340         jr   nz,seloop
350         ret
360 ;
370 ;Header empfangen vom CPC
380 ;Anfang und Laenge des Programms
390 header: call sync           ;Synchronisationsbytes senden.
400         call reada
410         ld   l,a            ;Hole Lowerbyte Anfang nach L-Reg.
420         call reada
430         ld   h,a            ;Hole Higherbyte Anfang nach H-Reg.
440         call reada
450         ld   c,a            ;Hole Lowerbyte Länge nach C-Reg.
460         call reada
470         ld   b,a            ;Hole Higherbyte Länge nach B-Reg.
480         ret
490 ;
500 ;CPC wartet auf diese Daten um in
510 ;Aktion zu treten.(Zahlen 10-0)
520 sync:   ld   b,10
530 syloop: ld   a,b
540         call writea
550         djnz syloop
560         ret
570 ;
580 ;EPC empfaengt Byte im Akku
590 reada:  push bc             ;BC retten.
600         ld   b,8            ;Ein Byte hat 8 Bit.
610 inbyte: call clkon          ;Warte auf aktiven Takt von CPC.
620         rla                 ;Bit5 von Input in Carry schieben,
630         rla
640         rla
650         rl   c              ;und Carry's in C-Reg sammeln.
660         call clkoff         ;Warte bis Takt inaktiv.
670         djnz inbyte         ;Wiederhole bis alle Bits in C-Reg.
680         ld   a,c            ;Byte soll ja im Akku sein.
690         pop  bc             ;BC zurück und fertig.
700         ret
710 ;
720 ;EPC sendet Byte im Akku
730 writea: push bc             ;BC retten
740         ld   c,a            ;Sendebyte nach C weil Akku gebr.wird.
750         ld   b,8            ;Ein Byte hat immer noch 8 Bit.
760 outbyt: call clkon          ;Warte auf aktiven Takt von CPC.
770         res  4,a            ;Akku ist Portbyte,Bit4 löschen.
780         rl   c              ;Das aktuelle Sendebit ins Carry-Flag.
790         jr   nc,lowbit      ;Wenn 0,dann sende das gelöschte Bit4.
800         set  4,a            ;Sonst sende High-Pegel.
810 lowbit: out  (datia),a      ;Gib diese Bit im Akku aus.
820         call clkoff         ;Warte bis Takt inaktiv.
830         djnz outbyt         ;Mache weiter bis alle Bits gesendet.
840         res  4,a            ;Gib am Schluß Ruhepegel aus.
850         out  (datia),a
860         pop  bc             ;BC zurück und fertig.
870         ret
```



```
880 ;
890 clkon:  in   a,(datia)      ;Warte auf aktives Taktsignal vom CPC.
900         bit  6,a
910         jr   z,clkon
920         ret
930 ;
940 clkoff: in   a,(datia)      ;Warte  bis Taktsignal inaktiv,also 0.
950         bit  6,a
960         jr   nz,clkoff
970         ret
```

V24 (RS 232) - Schnittstelle

Um mit anderen Rechnern, Modems, Akustikkopplern oder Druckern zu kommunizieren, ist oft eine serielle Schnittstelle erforderlich, mit der Norm V-24 oder RS 232 c. Bei dieser Art von Schnittstelle folgen die Bits im Gänsemarsch anstatt -wie beim Centronicsport- auf einmal.

Ein weiterer wichtiger Punkt ist, daß Daten in beide Richtungen übertragen werden können (Vollduplex) bei Modems etc. oder auch nur in einer Richtung z.B. bei Druckern (Halbduplex).

Da dabei nur wenige Leitungen (Leitungen für Senden, Empfangen plus einiger Handshakeverbindungen, zur Absprache der beiden beteiligten Systeme untereinander.) erforderlich sind, eignet sich diese Schnittstelle besser als eine parallele zur Überbrückung weiterer Entfernungen. Weiterhin erfolgt die Übertragung, aus Gründen der Datensicherheit (Leitungswiderstand, Einstreuungen), mit einen höheren Spannungspegel als 5 Volt, in der Regel +-9 bis +-12 Volt.

Man könnte nun die einzelnen zu sendenden Bits über einen normalen Port, wie vorher, seriell ausgeben, da es aber speziell für diese Normschnittstelle eigene IC's gibt, wollen wir mit so einem arbeiten.

Deshalb wird auch die eigentliche Arbeit, das Senden und Empfangen von Datenwörtern, das Reagieren auf Handshakesignale etc, von einem extra dafür geschaffenen Z-80 Baustein erledigt. Dieser Baustein enthält sowohl die Taktgeber für die Baudrate (Bit/Sekunde) der Datenübermittlung, die Sende- und Empfangssteuerungen als auch einen acht Bit breiten frei programmierbaren Port zum Bereitstellen der Handshakesignale.

Die Wahl fiel dabei auf ein nicht ganz billiges IC, welches aber durch seine komfortable Programmierung und seine Kompaktheit besticht. Die Rede ist von dem Baustein Z80 STI, der zu Unrecht etwas unbekannt ist.

Die Kurzdaten des IC's:

Baudraten einstellbar von 50 bis 19200 Baud,
5, 6, 7 oder 8 Bit breites Datenwort,
1, 1½, oder 2 Stoppbits,
gerader, ungerader oder keine Parität (Prüfbit).

Weitere Daten für den Programmierer:

* 24 Register stehen dem Programmierer zur Verfügung,
* 16 Register sind davon direkt über die Adressen F8E0 - F8EF ansprechbar,
* die restlichen acht Register sind indirekt ansteuerbar über ein spezielles Register der ersten 16 Register.

Jetzt zuerst die Bedeutung und die Adressen der einzelnen Register:

Register 0, Portadresse F8E0:
Dieses Register überträgt die Daten von und zu einem angewählten indirekten Register.

Register 1, Portadresse F8E1:
Dieses Register überträgt die Daten von und zu dem I/O-Port des Chips. Bei der V 24 stellt dieser Port die Handshakesignale zur Verfügung. In unserer Schnittstelle haben die einzelnen Bits folgende Bedeutung: (Bit 0 und 1 sind dabei auf Ausgabe programmiert)

Bit 0:	DTR (Data Terminal Ready)
Bit 1:	RTS (Ready to Send)
Bit 2:	CTS (Clear to Send)
Bit 3:	DSR (Data Set Ready)
Bit 4-7:	nicht verwendet

Register 2-7, Portadresse F8E2 - F8E7:
Die Register stehen dem Interrupthandler zur Verfügung. Da diese Eigenschaften bei unserer Anwendung nicht genützt werden, möchte ich nicht näher darauf eingehen. Wer sich dennoch für diese Eigenschaften interessiert, empfehle ich die Datenblätter der Firma Mostek.

Register 8, Portadresse F8E8:
Die Bits 3 - 7 stehen wieder der Interruptverwaltung zur Verfügung. Die Bits 0 - 2 dagegen sind wichtig, sie dienen zur Auswahl der indirekten Register. Der mögliche Inhalt der drei Bits (0-7) gibt dann die Nummer der indirekten Register an.

Register 9, Portadresse F8E9:
Kontrollregister für Timer A und B, sie stellen den jeweiligen Vorteiler sowie den Zählermodus ein. Bei der V-24 sind diese beiden Timer zwar nicht verwendet, sie lassen sich aber sehr wohl für andere externe Zwecke verwenden (Vorteiler, Generatoren etc.).

Bitnr.:	7	6	5	4	3	2	1	0
Wertigkeit für			Zähler A	und			Zähler B:	
	3	2	1	0	3	2	1	0

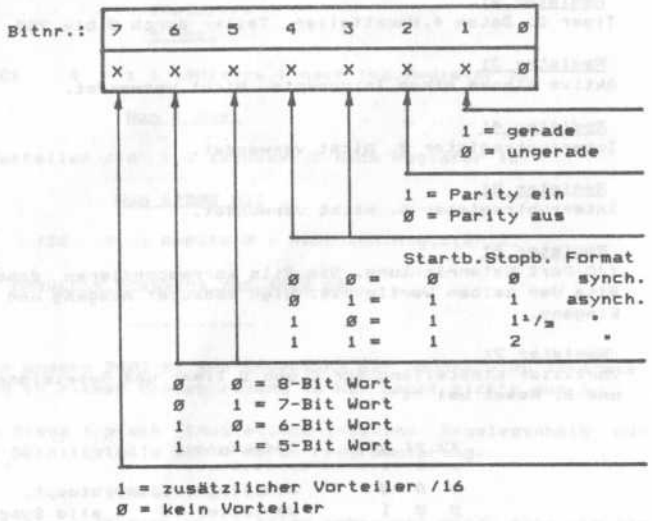
Die möglichen Kommandos für die Zähler:

3 2 1 0 (Wertigkeit wie vor.)		
0 0 0 0	Zähler stoppt !	
0 0 0 1	Vorteiler /4	symetrischer Ausgangsimpuls
0 0 1 0	" /10	" "
0 0 1 1	" /16	" "
0 1 0 0	" /50	" "
0 1 0 1	" /64	" "
0 1 1 0	" /100	" "
0 1 1 1	" /200	" "
1 0 0 0	Ereigniszähler	
1 0 0 1	Vorteiler /4	asymetrischer Ausgangsimpuls
1 0 1 0	" /10	" "
1 0 1 1	" /16	" "
1 1 0 0	" /50	" "
1 1 0 1	" /64	" "
1 1 1 0	" /100	" "
1 1 1 1	" /200	" "

Register 10, Portadresse F8EA:
Zähler B Wert für Hauptteiler, Teiler von /0 - /255.

Register 11, Portadresse F8EB:
Zähler A Wert für Hauptteiler, Teiler von /0 - /255.

Register 12, Portadresse F8EC:
USART-Register. Stellt die Übertragungsparmeter ein wie Wortbreite, Synchron/Asynchron, Stoppbits, Parity ein/aus und - gerade/ungerade.



Register 13, Portadresse F8ED:
Empfänger Status. Von den 8 Bits des Datenwortes sind nur die folgenden wichtig.

Bit Ø: Empfänger-Enable, muß auf "1" gesetzt sein, wenn Empfang erlaubt sein soll.
Bit 1: Empfangsbuffer voll "1" oder leer "Ø". Der Empfang der eingestellten Wortbreite gilt als "Buffer voll".

Register 14, Portadresse F8EE:
Sende Status. Wieder sind davon nur 2 Bits für uns ausschlaggebend.
Bit Ø: Sende-Enable, muß auf "1" gesetzt sein, wenn gesendet werden darf.
Bit 7: Sendebuffer voll "1" oder leer "Ø". Dieses Bit kann dazu benutzt werden, um festzustellen, ob ein Datenwort schon vollständig abgeschickt wurde.

Register 15, Portadresse F8EF:
Das tatsächliche Sende- und Empfangsregister. Ein zu sendendes Wort wird hier abgelegt oder ein empfangenes abgeholt.

Die indirekten Register (Auszuwählen über Register 8.):

Register Ø:
Synchron-Charakter-Register. Nicht verwendet.

Register 1:
Timer D: Daten f.Hauptteiler, Teiler durch Ø bis 255.

Register 2:
Timer C: Daten f.Hauptteiler, Teiler durch Ø bis 255.

Register 3:
Aktive Flanke eines Interrupts. Nicht verwendet.

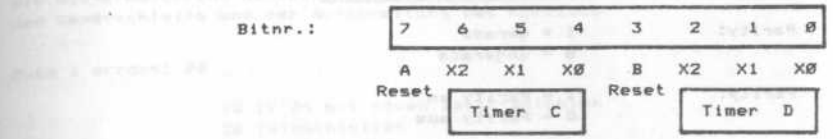
Register 4:
Interruptregister B. Nicht verwendet.

Register 5:
Interruptregister A. nicht verwendet.

Register 6:
I/O-Port Datenrichtung. Die Bits korrespondieren dabei mit den I/O Bits der selben Wertigkeit. High bedeutet Ausgang und Low steht für Eingang.

Register 7:
Vorteiler Einstellung für C und D Timer und Resetsignale der Timer A und B. Reset bei "1".

X2	X1	XØ	Bedeutung:
Ø	Ø	Ø	Jeweiliger Timer stoppt.
Ø	Ø	1	Vorteiler /4 (alle Synchron !)
Ø	1	Ø	" /10
Ø	1	1	" /16
1	Ø	Ø	" /50
1	Ø	1	" /64
1	1	Ø	" /100
1	1	1	" /200



Zum Abschluß noch ein Beispiel zur Programmierung der richtigen Teilerverhältnisse für eine bestimmte Baudrate.

```
*****************************
* Gewünschte Baudrate: 12ØØ *
* Prozessortakt      : 4 Mhz *
*****************************
```

Es soll Timer C verwendet werden, also hardwaremäßig den Ausgang des Timers C mit den Eingängen TC oder RC verbinden. Es können auch beide Eingänge (TC,RC) am gleichen Timerausgang hängen, dann sind halt Sende- und Empfangstakt gleich, was ja ohnehin in den meisten Fällen zutrifft.

(Es wird immer die doppelte Frequenz gebraucht, da jedes Bit zweimal gescannt wird. In unserem Fall also 24ØØ Hz.)

---

4 Mhz:

Vorteiler von C: /4 : (xØØ1xxxx) nach Ind.Register 7.

Nun 1 Mhz:

Zusätzlicher Vorteiler /16 : (1xxxxxxx) nach Register 12.

Nun 625ØØ Hz:

Hauptteiler C: /26 : (ØØØ11Ø1Ø) nach Ind.Register 2.

Gesuchte Frequenz von 24Ø3 Hz.

Jetzt haben wir unsere 24Ø3 Hz zur Erzeugung der gewünschten Baudrate. Eine Abweichung in dieser Größenordnung (3 Hz) macht nichts aus !

Im Anschluß an diese typisch theoretisch-trockene Angelegenheit nun der Aufbau der Schnittstelle sowie deren Programmierung.

****

Da diese Erweiterung mit Sicherheit öfter gebraucht wird, habe ich die Software mittels RSX-Technik in das Basic des Rechners eingebunden.

Es stehen deshalb drei neue Basic-Befehle zur Verfügung:

!FORMAT, Nr.d. Baudrate, Anz.Stoppbits, Wortbreite, Parity ein/aus, Parity gerade/ungerade

Data Bits

Möglichkeiten der Einstellungen:

Parity:	1 = gerade 0 = ungerade
Parity:	1 = Parity an 0 = Parity aus
Stoppbits:	1 = 1 Stoppbit 0 = 2 Stoppbit
Wortbreite:	0 = 7 Datenbit 1 = 8 Datenbits
Baudratennummer:	0 = 50 Baud 1 = 75 Baud 2 = 110 Baud 3 = 150 Baud 4 = 300 Baud 5 = 600 Baud 6 = 1200 Baud 7 = 2400 baud

```
|RECORD, @a$
```

Die ankommenden Zeichen werden in einen von Basic bereitgestellten String eingelesen. Es wird solange eingelesen, bis der String voll ist, BREAK gedrückt wird oder ein CR (Chr$ (13)) erkannt wird. Diese CR's und alle sonstigen (Steuer-)Zeichen werden ebenfalls in den String gepackt, um die Übermittlung von Binärfiles zuzulassen.

```
|SEND, @a$
```

Der String wird bis zum Ende gesendet, außer es wurde BREAK gedrückt; es wird am Schluß nicht automatisch ein CR oder LF angehängt, um auch hier wieder die Sendedaten nicht zu verfälschen.

Das Basicprogramm verdeutlicht die Zusammenhänge und stellt gleichzeitig eine einfache Bedienung der V-24 dar.

```
100 'V-24 Treiberprogramm
110 'Initialisieren
120 '-------------------
130 MEMORY 40999
140 LOAD"v24.bin",41000
150 CALL 41000
160 '-------------------
170 ' 2400 Baud
180 ' 1 Stoppbit
190 ' 8-Bit-Wortbreite
200 ' kein Parity-Check
210 '
220 |FORMAT,7,1,8,0,0
230 '
240 '-----------------
250 'Empfangsschleife
260 '-----------------
270 a$=" "
280 |RECORD,@a$
290 PRINT a$;:GOTO 280
300 '
310 '---------
320 'Sendeteil
330 '---------
340 a$="Sendetext"
350 |SEND,@a$
```


Die RSX-Erweiterung besteht nun in der Hauptsache aus einer Empfangs- und Sendeschleife und der Aufbereitung des Formates.

```
Pass 1 errors: 00

                        10 ;V-24 mit neuen Basic-Befehlen
                        20 ;einschleifen
                        30 ;----------------------------
                        40 ;init
A028                    50          org   41000
A028  0131A0            60          ld    bc,jump
A02B  214CA0            70          ld    hl,memory
A02E  C3D1BC            80          jp    #bcd1
                        90 ;
A031  3CA0              100 jump:   defw  namen
A033  C350A0            110         jp    format
A036  C3D7A0            120         jp    senden
A039  C3EAA0            130         jp    empfgn
                        140 ;
A03C  464F524D          150 namen:  defm  "FORMA"
A041  D4                160         defb  "T"+80
A042  53454E            170         defm  "SEN"
A045  C4                180         defb  "D"+80
A046  5245434F          190         defm  "RECOR"
A04B  C4                200         defb  "D"+80
A04C                    210 memory: defs  4
                        220 ;----------------------------
                        230 ;baud,stopbits,wortlaenge,etc.
                        240 ;einstellen.
A050  FE05              250 format: cp    5           ;5 Argumente ?
A052  C0                260         ret   nz
A053  DD7E00            270         ld    a,(ix+0)  ;Parameter gerade/un-
A056  E601              280         and   %00000001 ;gerade nach Akku holen
A058  CB27              290         sla   a         ;Bit-0 ausblenden und
A05A  CBFF              300         set   7,a       ;linksschieben
A05C  47                310         ld    b,a       ;Bit 7 immer 1
                        320 ;                       ;Bit 0 immer 0
A05D  DD7E02            330         ld    a,(ix+2)  ;Parameter Parity ein/
A060  E601              340         and   %00000001 ;aus holen, Bit-0 ausbld.
A062  CB27              350         sla   a         ;2mal linksschieben und
A064  CB27              360         sla   a         ;mit B oderieren
A066  B0                370         or    b         ;Im Akku wird Formatbyte
                        380 ;                       ;zusammengestellt !
A067  DD4604            390         ld    b,(ix+4)  ;Wortbreite 7/8 Bit nach
A06A  CB40              400         bit   0,b       ;B holen, nur Bit-0 davon
A06C  2002              410         jr    z,noset   ;testen und Bit 5 im Akku
A06E  CBEF              420         set   5,a       ;in Abhängigkeit davon
                        430 ;                       ;setzen oder nicht.
A070  DD4606            440 noset:  ld    b,(ix+6)  ;Stoppbit-Byte holen,auf
A073  CB40              450         bit   0,b       ;1 oder 2 testen und in
A075  2002              460         jr    nz,stop1  ;Abhängigkeit davon
A077  CBE7              470         set   4,a       ;Bit 4 und/oder Bit 3 im
A079  CBDF              480 stop1:  set   3,a       ;Akku setzen.
A07B  01ECF8            490         ld    bc,#f8ec
A07E  ED79              500         out   (c),a     ;fertiges Format-Byte nach
                        510 ;                       ;USART-Contr.Reg.ausgeben.
A080  0EE8              520         ld    c,#e8
A082  3E06              530         ld    a,6       ;Indirekt-Reg 6 wählen.
```

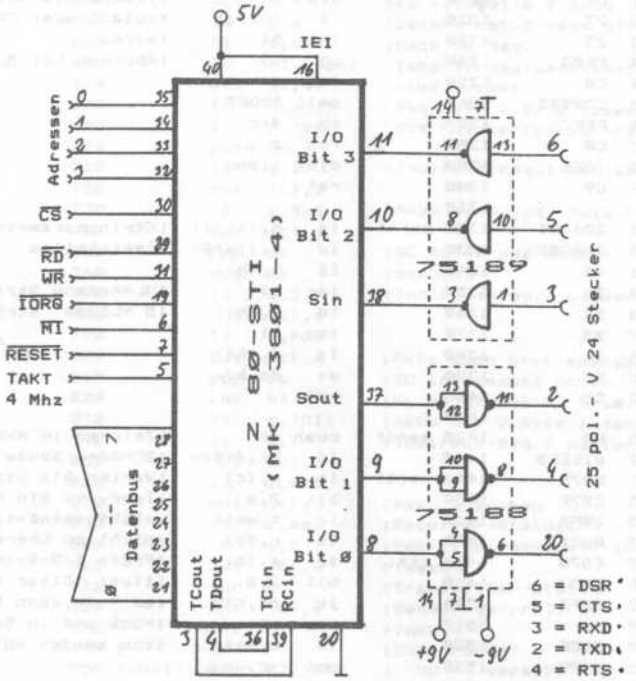
```
A084  ED79      540          out  (c),a
A086  0EE0      550          ld   c,#e0      ;I/O-Port Bit 0 und 1
A088  3E03      560          ld   a,3        ;sind Ausgabeleitungen.
A08A  ED79      570          out  (c),a
A08C  0C        580          inc  c          ;über General-Purp.-Reg.
A08D  3E01      590          ld   a,1        ;DTR sperren.
A08F  ED79      600          out  (c),a
                610 ;
A091  DD7E08    620          ld   a,(ix+8)   ;Nummer d. Baudrate holen.
A094  87        630          add  a,a        ;*2 w.Tabelle 2-Byte Werte
A095  5F        640          ld   e,a        ;beinhaltet.Diesen Offset
A096  1600      650          ld   d,0        ;nach DE-Reg.
A098  21C7A0    660          ld   hl,tabell  ;Tabellenbasisadr.bestim.
A09B  19        670          add  hl,de      ;plus Offset
A09C  7E        680          ld   a,(hl)     ;Akku mit 1.Byte laden
A09D  0EE8      690          ld   c,#e8      ;BC= Pointer/Vektor-Reg.
A09F  C5        700          push bc
A0A0  1E01      710          ld   e,1        ;Indirekt-Reg.1 wählen.
A0A2  ED59      720          out  (c),e
A0A4  0EE0      730          ld   c,#e0      ;Akku nach Ind.-Data-Reg.
A0A6  ED79      740          out  (c),a
A0A8  C1        750          pop  bc         ;BC holen und wieder
A0A9  C5        760          push bc         ;zurück.
A0AA  1E02      770          ld   e,2        ;Indirekt-Reg.2 wählen.
A0AC  ED59      780          out  (c),e
A0AE  0EE0      790          ld   c,#e0
A0B0  ED79      800          out  (c),a      ;Akku auch hier ausgeben.
A0B2  C1        810          pop  bc         ;BC letztesmal holen.
A0B3  23        820          inc  hl         ;HL zeigt auf n. Byte.
A0B4  7E        830          ld   a,(hl)     ;Akku mit 2.Byte laden
A0B5  1E07      840          ld   e,7        ;Indirekt-Reg.7 wählen.
A0B7  ED59      850          out  (c),e
A0B9  0EE0      860          ld   c,#e0
A0BB  ED79      870          out  (c),a      ;Akku ausgeben
A0BD  0EED      880          ld   c,#ed      ;Receiver-Status:
A0BF  1E01      890          ld   e,1        ;Empfangen generell zu-
A0C1  ED59      900          out  (c),e      ;lassen !
A0C3  0C        910          inc  c          ;Transmitter-Status;
A0C4  ED59      920          out  (c),e      ;Senden generell zu-
A0C6  C9        930          ret             ;lassen !
                940 ;                        ;ENDE FORMAT
A0C7  2755      950 tabell:  defb #27,#55    ;  50    versch.
A0C9  1A55      960          defb #1a,#55    ;  75    Baud-
A0CB  4733      970          defb #47,#33    ; 110    raten.
A0CD  3433      980          defb #34,#33    ; 150
A0CF  1A33      990          defb #1a,#33    ; 300
A0D1  3411     1000          defb #34,#11    ; 600
A0D3  1A11     1010          defb #1a,#11    ;1200
A0D5  0E11     1020          defb #0e,#11    ;2400
               1030 ;---------------------------
               1040 ;Sende- und Empfangsschleife
               1050 ;
A0D7  CD00A1   1060 senden: call para       ;String aufbereiten.
A0DA  7E       1070 sloop:  ld   a,(hl)     ;1.Zeichen nach Akku
A0DB  23       1080         inc  hl         ;B-Reg ist Zähler.
A0DC  C5       1090         push bc
A0DD  CD0DA1   1100         call send       ;alle Zeichen aus String
A0E0  C1       1110         pop  bc         ;ausgeben oder nach
```



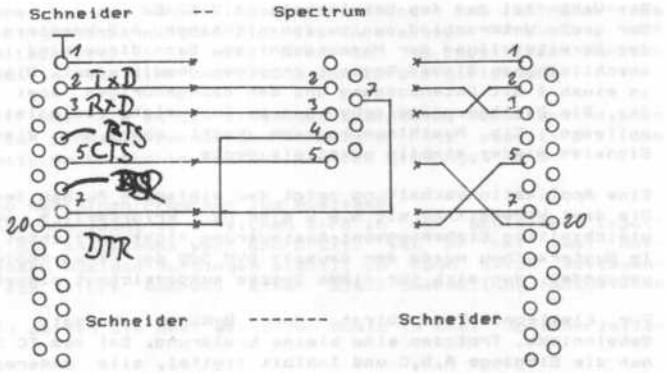
```
A0E1  CD09BB   1120         call #bb09      ;Break zurück.
A0E4  FEFC     1130         cp   #fc
A0E6  C8       1140         ret  z
A0E7  10F1     1150         djnz sloop
A0E9  C9       1160         ret
               1170 ;
A0EA  CD00A1   1180 empfgn: call para       ;Stringdesc.aufbereiten.
A0ED  C5       1190 rloop:  push bc         ;Von nun an so viele
A0EE  CD25A1   1200         call record     ;Zeichen in String ein-
A0F1  C1       1210         pop  bc         ;lesen, wie dessen Länge
A0F2  77       1220         ld   (hl),a     ;zuläßt oder CR empfangen
A0F3  23       1230         inc  hl         ;wird.
A0F4  FE0D     1240         cp   13         ;Abbruch bei Break.
A0F6  C8       1250         ret  z
A0F7  CD09BB   1260         call #bb09
A0FA  FEFC     1270         cp   #fc
A0FC  C8       1280         ret  z
A0FD  10EE     1290         djnz rloop
A0FF  C9       1300         ret
               1310 ;
A100  DD6601   1320 para:   ld   h,(ix+1)   ;Stringparameter
A103  DD6E00   1330         ld   l,(ix+0)   ;feststellen
A106  46       1340         ld   b,(hl)
A107  23       1350         inc  hl         ;HL=Anfang String
A108  5E       1360         ld   e,(hl)     ;B =Länge  String
A109  23       1370         inc  hl
A10A  56       1380         ld   d,(hl)
A10B  EB       1390         ex   de,hl
A10C  C9       1400         ret
               1410 ;
A10D  F5       1420 send:   push af         ;Zeichen im Akku retten.
A10E  01EEF8   1430         ld   bc,#f8ee   ;BC=Adr. Sende-Status.
A111  ED78     1440 wait:   in   a,(c)      ;Warten bis Sendebuffer
A113  CB7F     1450         bit  7,a        ;leer, um ein noch
A115  28FA     1460         jr   z,wait     ;nichtgesendete Zeichen
A117  0EE1     1470         ld   c,#e1      ;nicht zu überschreiben.
A119  ED78     1480 cts:    in   a,(c)      ;Prüfe I/O-Byte
A11B  CB57     1490         bit  2,a        ;liegt "Clear to Send"
A11D  20FA     1500         jr   nz,cts     ;an ? Ja,dann Akku zu-
A11F  F1       1510         pop  af         ;rück und in Data-Reg.
A120  0EEF     1520         ld   c,#ef      ;zum senden ablegen.
A122  ED79     1530         out  (c),a
A124  C9       1540         ret
               1550 ;
A125  01E1F8   1560 record: ld   bc,#f8e1   ;BC=Adr.I/O-Byte (Hand-
A128  AF       1570         xor  a          ;shake).--Akku löschen
A129  ED79     1580         out  (c),a      ;und ausgeben entspricht
A12B  0EED     1590         ld   c,#ed      ;"Data Terminal Ready"
A12D  ED78     1600 nowort: in   a,(c)      ;Empfänger Status prüfen.
A12F  CB7F     1610         bit  7,a        ;und warten bis 7 oder 8
A131  28FA     1620         jr   z,nowort   ;Bits ankamen.
A133  0EEF     1630         ld   c,#ef      ;Jetzt enthält Data-Reg.
A135  ED78     1640         in   a,(c)      ;das empf.Byte.Nach A.
A137  0EE1     1650         ld   c,#e1      ;Auf I/O-Byte "Data Ter-
A139  1E01     1660         ld   e,1        ;minal NOT Ready" ausge-
A13B  ED59     1670         out  (c),e      ;ben.
A13D  C9       1680         ret

Pass 2 errors: 00
```

Die Hardware selbst ist sehr einfach ins System zu integrieren, da kein externer Taktgenerator erforderlich ist und deshalb alles von einem IC erledigt wird.

Die V-24 Treiber vom Typ SN 75189 und SN 75188 sorgen für die Entkopplung der verschiedenen Spannungen.



Um auch gleich einen praktische Problemlösung zu bieten, zeige ich das erforderliche Kabel um den SINCLAIR-SPECTRUM + Interface 1 mit dem SCHNEIDER CPC xxxx zu verbinden. Die Pinbezeichnungen beziehen sich auf der CPC-Seite auf einen 25-pol. V-24 Stecker und auf der Seite des SPECTRUMs auf einen 9-pol.-DIN-D-Stecker.
Die Datenübertragung gelang zwischen beiden Geräten auf Anhieb. Wenn's mal nicht funktioniert mit anderen Rechnern und Druckern, liegt's entweder am Kabel (z.B. TXD und RXD vertauscht!) oder am Übertragungsformat selber. Also experimentieren, denn die Norm V24 ist sehr,sehr dehnbar !
Böse Zungen behaupten, es gäbe so viele verschiedene Anschlußmöglichkeiten, wie der Stecker Pin's hat.



Ein Gutes hat aber diese Norm auf alle Fälle, die Ausgänge sind kurzschlußfest, so daß auch mal zwei Ausgänge gegeneinander arbeiten dürfen !

Einem Plauderstündchen diverser Computer steht nun nichts mehr im Wege.

8-Kanal Analog / Digital Wandler

Lassen Sie doch mal Ihren Computer messen !
Gerade wenn Messergebnisse über einen längeren Zeitraum kontinuierlich erfasst und vielleicht noch grafisch ausgegeben werden sollen, ist ein Rechner mit Analog-Eingang sehr hilfreich.
Da nun die digitale Welt eines Computers mit analogen Werten nichts anfangen kann, muß ein Umwandler die Aufbereitung solcher Signale übernehmen.
Der Markt bietet nun eine ganze Reihe solcher Wandler-IC's, die speziell für solche Aufgaben geschaffen wurden. Die meisten setzen einen analogen Eingangswert direkt auf einen 8 Bit (oder mehr) breiten Datenbus um. Es gibt aber auch einige IC's die für andere Zwecke, nämlich der direkten Ansteuerung von Siebensegmentanzeigen ausgelegt sind. Der Vorteil ist, daß diese IC's in den meisten Fällen wesentlich billiger sind als ihre Kollegen.

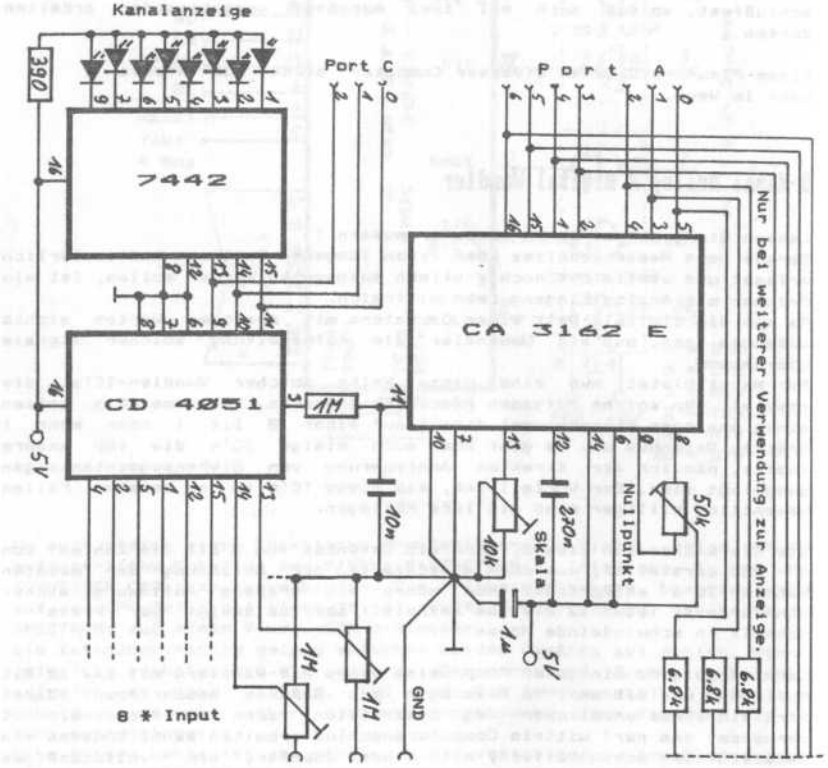
Wie Sie sicherlich wissen, kann ein Datenbus von 8 Bit die Zahlen von 0 - 255 darstellen, was auch gleichzeitig der Auflösung der meisten Wandler-IC's entspricht. Nun wäre ein größere Auflösung sicher von Vorteil, (etwa 12 Bit zum Beispiel) aber da steigt der Preis schnell in schwindelnde Höhen.

Es wird folgend ein guter Kompromiss eines A/D-Wandlers mit ca. 10 Bit Auflösung und mit max. 96 Messungen pro Sekunde beschrieben. Dabei wird ein etwas unüblicher Weg beschritten, denn es wird kein IC verwendet das nur! mittels Computeranschluß arbeiten kann. Sondern ein Baustein der schon allein, mit etwas Zubehör, ein vollständiges 3-stelliges Digitalvoltmeter bildet.

Die Wahl fiel auf den Schaltkreis CA 3162 E.
Der große Unterschied nun zu den richtigen A/D-Wandlern, besteht in der Bereitstellung der Messergebnisse. Denn diese sind ja -wegen der anschließbaren Sieben-Segment-Anzeige- gemultiplext. Dies stellt sich in einen 4 Bit Datenausgang und den dazugehörigen drei Digittreibern dar. Die Digitausgänge geben nun an für welche Dezimalstelle die Daten anliegen. Ein Maschinenprogramm macht aber aus diesen "wilden" Signalen wieder stabile parallele Werte.

Eine Applikationsschaltung zeigt den einfachen Aufbau des Instruments. Die drei Widerstände mit 6,8 k sind nur erforderlich wenn auf eine gleichzeitige Siebensegment-Ansteuerung nicht verzichtet werden soll. Im Musteraufbau wurde der Bausatz DVM 300 der Firma RADIO-RIM, München verwendet, der sich für diese Zwecke ausgezeichnet eignet.

Für Elektronik-Fans birgt der 8-Kanal Vorsatz sicher keine Geheimnisse. Trotzdem eine kleine Erklärung. Bei dem IC CD 4051 sind nur die Eingänge A,B,C und Inhibit digital, alle anderen gehören zu Analog-Schaltern mit, in dieser Beschaltung, einem Eingangsspannungsbereich von 0-5 Volt, der Durchlaßwiderstand ist dabei



vernachlässigbar bei dem hohen Eingangswiderstand (ca. 100 MOhm) des CA 3162. Der Eingangsspannungsbereich des CA 3162 von -99 mV bis 999 mV (Millivoltschritte !) wird hier nur im positiven Bereich ausgenützt, was in etwa einer Auflösung von 10 Bit gleichkommt.

Die (Wendel-)Trimmer 1 M an den Eingängen des Analog-Schalters dienen der Erhöhung des Eingangsspannungsbereichs. Der 7442 stellt mit den LED's eine optische Kontrolle des aktivierten Einganges dar.

Das zugehörige Maschinen-Programm zum Auslesen:
Um eine sichere Auslesung zu erreichen wird in der Schleife "Input" immer zweimal der anstehende Wert überprüft. Ist er bei zwei aufeinanderfolgenden Auslese-Vorgängen stabil, so kann davon ausgegangen werden, daß nicht während einer Digit-Umschaltung ausgelesen wurde.
Ansonsten wird werden die drei Werte für Basic in drei Speicherzellen abgelegt.

Pass 1 errors: 00

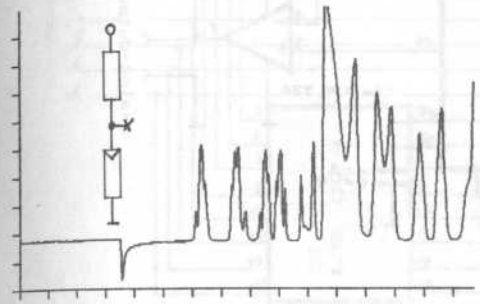
```
                    10 ;   DVM-Auslesung
                    20 ;in drei Speicherzellen
                    30 ;----------------------
                    40 ;
9C40                50          org  40000
9C40  06F8          60          ld   b,#f8
9C42  0EF0          70          ld   c,#f0
                    80 ;
9C44  CD729C        90 hundt:   call input
9C47  CB57         100          bit  2,a
9C49  20F9         110          jr   nz,hundt
9C4B  CD6C9C       120          call schieb
9C4E  32819C       130          ld   (byte),a
                   140 ;
9C51  CD729C       150 einer:   call input
9C54  CB47         160          bit  0,a
9C56  20F9         170          jr   nz,einer
9C58  CD6C9C       180          call schieb
9C5B  32839C       190          ld   (byte+2),a
                   200 ;
9C5E  CD729C       210 zehnr:   call input
9C61  CB4F         220          bit  1,a
9C63  20F9         230          jr   nz,zehnr
9C65  CD6C9C       240          call schieb
9C68  32829C       250          ld   (byte+1),a
9C6B  C9           260          ret
                   270 ;
9C6C  1F           280 schieb:  rra
9C6D  1F           290          rra
9C6E  1F           300          rra
9C6F  E60F         310          and  #0f
9C71  C9           320          ret
                   330 ;
9C72  ED78         340 input:   in   a,(c)
9C74  57           350          ld   d,a
9C75  C5           360          push bc
9C76  060A         370          ld   b,10
```

```
9C78  10FE      380 loop:   djnz loop
9C7A  C1        390         pop  bc
9C7B  ED78      400         in   a,(c)
9C7D  BA        410         cp   d
9C7E  20F2      420         jr   nz,input
9C80  C9        430         ret
9C81  000000    440 byte:   defb 0,0,0

Pass 2 errors: 00
```

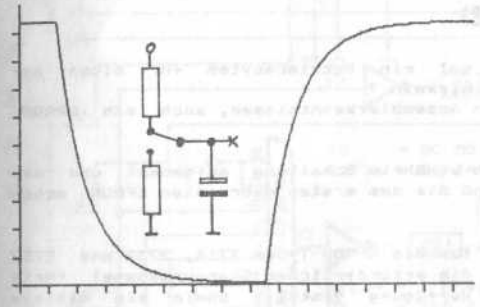
Um Sie aber nicht mit dem Schaltplan und dem Maschinen-Programm allein zu lassen, gleich eine Anwendung die Sie auf den Geschmack bringen soll.
Ein kleines (ausbaufähiges) Basic-Programm stellt eine Art Speicheroszilloskop dar, freilich nur für nicht allzu schnelle Vorgänge, aber immerhin.

```
10 ' kleines Speicheroszilloskop
20 ' mit dem DVM-300 + Grafikaufbereitung
30 '    "Zeitbasis" = ca. 15 sec
40 ' ------------------------------------
50 '
60 OUT (&F8F3),152:' 8255-Datenrichtung
70 OUT (&F8F2),1:'   Kanal 1 waehlen
80 CLS
90 '
100 '-------------------------------------
110 'linke und untere Bildschirmbegrenzung
120 'plus Zeit- und Voltteilung
130 '
140 MOVE 10,399:DRAW 10,10:DRAW 639,10
150 FOR n=10 TO 650 STEP 40
160   PLOT n,10:DRAW n,-10
170 NEXT
180 FOR n=10 TO 390 STEP 39
190   PLOT 0,n:DRAW 10,n
200 NEXT
210 '
220 ' ----------------------
230 ' 628# DVM-Wert auslesen
240 '
250 GOSUB 350:PLOT 10,wert%/2.6+11
260 FOR n%=11 TO 639
270   GOSUB 350
280   DRAW n%,wert%/2.6+11
290 NEXT
300 GOTO 300
310 '
320 '------------
330 'DVM einlesen
340 '
350 CALL 40000
360 wert%=100*PEEK(&9C81)+10*PEEK(&9C82)+PEEK(&9C83)
370 RETURN
```

Damit lassen sich schon ganz schön aussagekräftige Schaubilder erzeugen, was die zwei Hardcopy's verdeutlichen sollen.



Die erste kleine negative Spitze war die Entladung eines Fotoblitzers, also eines relativen schnellen Vorganges ! Die anderen Zacken entstanden nur durch die mutwillige Abdeckung des LDR's.

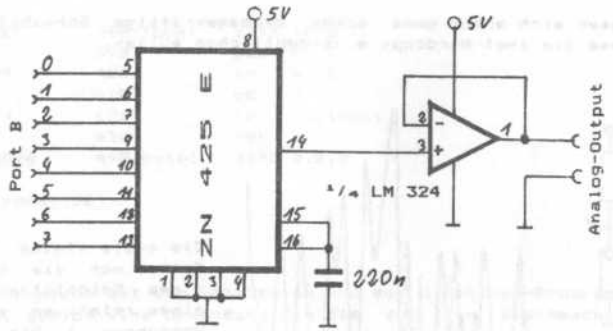


Die Entlade- und Ladekurve eines Elko's.

Digital / Analog Wandler

Auch der umgekehrte Weg ist möglich, nämlich der, aus einem digitalen ein analoges Signal zu machen.
Das IC ZN-425-E wandelt den Datenwert 0 - 255 in den analogen Wert 0 - 2,55 V um. Da aber der Ausgang des Wandlers ziemlich hochohmig ist und somit die Ausgangsspannung bei Belastung zusammenbricht, wurde am Ausgang ein Operationsverstärker LM 324 als Spannungsfolger angeschaltet. Der Ausgangsstrom kann nun bis zu 20 mA betragen. Nicht bei allen Operationsverstärkern, mit asymetrischer Stromversorgung, kann der Ausgang dem Eingang bis 0 V runter folgen, bei dem LM 324 schon, weshalb er auch verwendet wurde.

Die Programmierung des D/A-Wandlers ist so einfach, daß ich hier auf ein Programmbeispiel verzichten möchte, denn jede Ausgabe auf Port-B des 8255 wird direkt in einen Spannungswert gewandelt.



EPROM Programmierer (2716,2732)

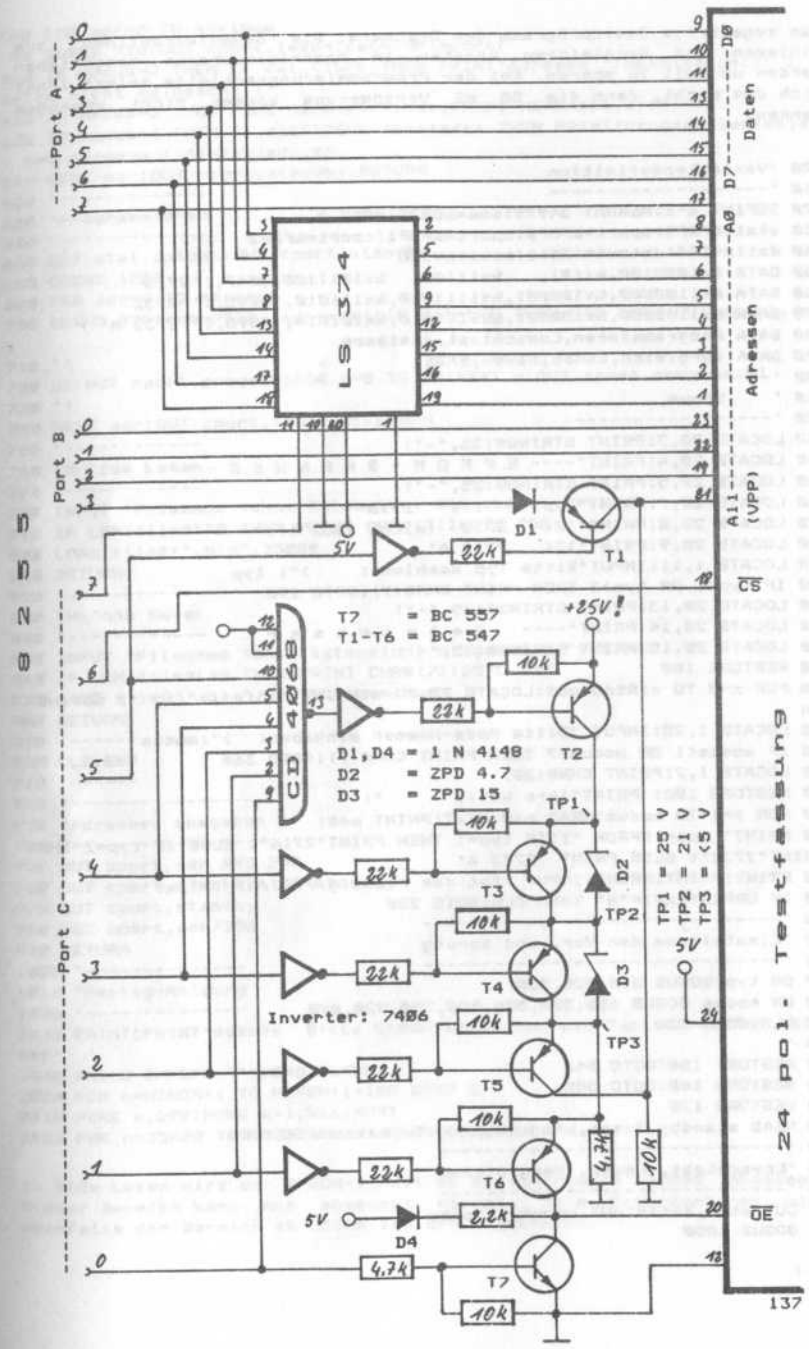
Wollten Sie nicht schon immer mal ein Betriebssytem für einen der vielen Einplatinencomputer entwickeln ?
Dann brauchen Sie, außer guten Assemblerkenntnissen, auch ein EPROM-Programmiergerät.

Wenn Sie die folgende mehrfach bewährte Schaltung aufgebaut und die Software eingetippt haben, sind Sie dem ersten gebrannten EPROM schon sehr nahe gekommen.

Die vorliegende Schaltung ist für die EPROM-Typen 2716, 2732 und 2732 A geeignet. Es werden von ihr die erforderlichen Spannungspegel für's Lesen und Programmieren zur Verfügung gestellt sowie ein mittels Software erzeugter Programmierimpuls von genau 50 mS.
Da Pin 21 am EPROM bei dem Typ 2716 der Programmierspannungsanschluss und bei den 2732er Typen das Adressbit 11 ist, muß die Umschaltung durch T1 vorgesehen werden.
Die Transistoren T2 - T7 dienen zur Erzeugung der diversen Spannungspegel an den Eingängen des EPROM's.

Durch das NAND-Gatter wird ein verhängnisvoller "Fehler" der Hardware vermieden, denn wenn Sie den Computer einschalten sind alle Ports des 8255 auf Input geschaltet. Dies wird von der angeschlossenen Schaltung als "high" interpretiert und da alle Transistoren leiten würden, -ohne diese Sicherungsmaßnahme- einen Kurzschluss der Programmierspannung bedeuten.
Ein weiteres Detail der Schaltung ist die Verwendung des 74 LS 374 als Zwischenspeicher des LSB der Adressen. Somit kann jede beliebige Adresse sofort angelegt werden. Die Software unterscheidet dabei zwischen Daten und Adressen und gibt bei Letzteren einen kurzen Latch-Impuls zum Speichern aus. Da noch weitere Bits des B-Ports für das MSB der Adresse zur Verfügung stehen kann die Schaltung auch noch für höher integrierte EPROMS -nach Veränderung- verwendet werden.

Nun die Schaltung selbst:



Das zugehörige Basicprogramm des Brenners. Die "Schaufelroutinen" zum Einlesen und Vergleichen könnten in Maschinensprache geschrieben werden um Zeit zu sparen. Bei der Programmierungsschleife selbst lohnt sich das nicht, denn die 50 mS Verzögerung können nicht umgangen werden.

```
100 'Variablendefinition
110 '-------------------
120 DEFINT a-z:MEMORY 34999:mem=&88B8:MODE 2
130 stat=&F8F3:aport=&F8F0:bport=&F8F1:cport=&F8F2
140 datin=144:datout=128:clockin=128
150 DATA &x1000100,&x101,   &x11100,  &x1011100,2047,:'2716
160 DATA &x1100000,&x100001,&x1111010,&x111010, 4095,:' 2732
170 DATA &x1100000,&x100001,&x1101010,&x101010, 4095,:' 2732 A
180 DATA Programmieren,Loeschtest,Auslesen
190 DATA Vergleich,Laden,Saven,ENDE
200 '-------------------
210 '    Menue
220 '-------------------
230 LOCATE 20,3:PRINT STRING$(35,"-")
240 LOCATE 20,4:PRINT"---- E P R O M - B R E N N E R ----"
250 LOCATE 20,5:PRINT STRING$(35,"-")
260 LOCATE 20,7:PRINT"Typ:     )1(   2716"
270 LOCATE 28,8:PRINT")2(   2732  (nicht 2532 !!)"
280 LOCATE 28,9:PRINT")3(   2732 A"
290 LOCATE 1,11:INPUT"Bitte Typ waehlen :    )"; typ
300 IF typ<1 OR typ>3 THEN PRINT CHR$(7):GOTO 290
310 LOCATE 20,13:PRINT STRING$(35,"-")
320 LOCATE 20,14:PRINT"----   B e t r i e b s a r t   ----"
330 LOCATE 20,15:PRINT STRING$(35,"-")
340 RESTORE 180
350 FOR n=0 TO 6:READ md$:LOCATE 25,17+n:PRINT ")";n+1;"(   ";md$:NEX
T n
360 LOCATE 1,25:INPUT "Bitte Mode-Nummer eingeben:  )";modus
370 IF modus<1 OR modus>7 THEN PRINT CHR$(7):GOTO 360
380 LOCATE 1,7:PRINT CHR$(20)
390 RESTORE 180: PRINT"Ihre Wahl:      ";
400 FOR n=1 TO modus:READ md$:NEXT:PRINT md$;
410 PRINT" fuer EPROM ";:IF typ=1 THEN PRINT"2716"; ELSE IF typ=2 THEN
 PRINT"2732"; ELSE PRINT "2732 A"
420 PRINT:PRINT:PRINT:INPUT "Ist das richtig ? (J/N)  )";x$
430 IF UPPER$(x$)="N" THEN CLS:GOTO 230
440 '------------------------------
450 'Einstellung der Var. und Sprung
460 '------------------------------
470 ON typ GOSUB 510,520,530
480 ON modus GOSUB 650,550,550,550,750,820,890
490 CLS:GOTO 230
500 '
510 RESTORE 150:GOTO 540
520 RESTORE 160:GOTO 540
530 RESTORE 170
540 READ standby,lesen,progaus,progein,maximum:RETURN
550 '------------------------------
560 'Loeschtest, Lesen, Vergleichen
570 '------------------------------
580 OUT stat,datin:OUT cport,standby
585 GOSUB 1000
```



```
590 FOR adr=0 TO maximum
600 GOSUB 920:OUT cport,lesen:dat=INP(aport)
610 IF modus=2 THEN IF dat <>&FF THEN PRINT"Adresse ";HEX$(adr,4);" :
Zelle nicht geloescht !"
620 IF modus=3 THEN POKE (mem+adr),dat:PRINT HEX$(dat,2);"  ";
630 IF modus=4 THEN IF dat<>PEEK (mem+adr) THEN PRINT"Programmierfehle
r bei Adresse : ";HEX$(adr,4)
640 NEXT adr:OUT cport,standby:RETURN
650 '-------------
660 'Programmieren
670 '-------------
680 OUT stat,datout:OUT cport,standby
685 GOSUB 1000
690 FOR adr=0 TO maximum
700 GOSUB 920:dat=PEEK (adr+mem):IF dat=&FF THEN 740 ELSE OUT aport,da
t
710 '!
720 DI:OUT cport,progein:FOR n=0 TO 85:NEXT n:OUT cport,progaus:EI
730 '!
740 NEXT adr:OUT cport,standby:RETURN
750 '------------
760 'M-Code Laden
770 '------------
780 INPUT "Filename (ohne Extension) eingeben : ";file$
790 IF LEN(file$)>8 THEN PRINT CHR$(7):GOTO 780
800 LOAD file$+".bin",35000
810 RETURN
820 '------------
830 'M-Code Saven
840 '------------
850 INPUT "Filename (ohne Extension) eingeben : ";file$
860 IF LEN(file$)>8 THEN PRINT CHR$(7):GOTO 850
870 SAVE file$+".bin",b,35000,maximum
880 RETURN
890 '------
900 CLS:END
910 '------
920 '-----------------
930 'Adressen ausgeben
940 '-----------------
950 OUT bport,adr AND 255
960 OUT cport,clockin OR standby
970 OUT cport,standby
980 OUT bport,adr/256
990 RETURN
1000 '--------------
1010 'Fertig-Meldung
1020 '--------------
1030 PRINT:PRINT"******  Bitte EPROM einsetzen und Taste druecken. ***
***"
1040 WHILE INKEY$="":WEND:RETURN
2000 FOR n=HIMEM+1 TO HIMEM+1+100 STEP 2
2010 POKE n,&99:POKE n+1,&66:NEXT
3000 FOR n=35000 TO 35100:PRINT PEEK(n),:NEXT
```

Im Mode Lesen wird der EPROM-Inhalt ab Speicherzelle 35000 abgelegt. Dieser Bereich kann nun abgesavt werden. Beim Programmieren wird ebenfalls der Bereich ab 35000 ins EPROM gebracht.

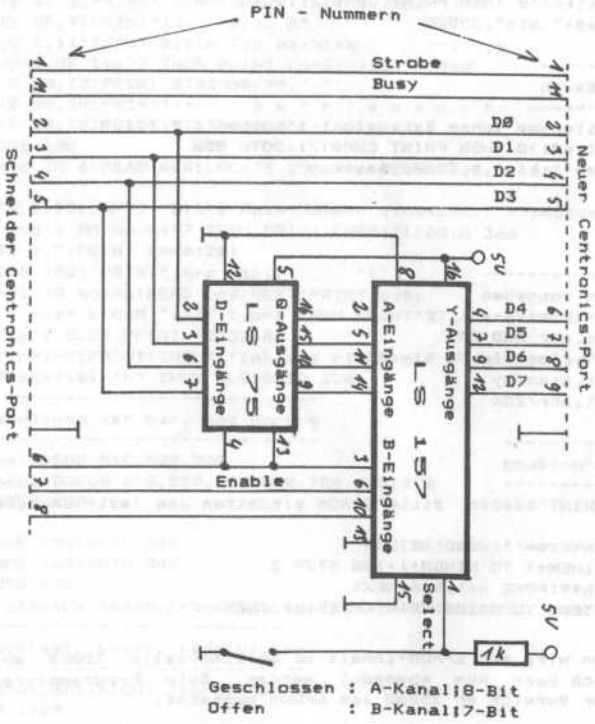
8-Bit Centronicsport

oder die Überwindung eines Nadelöhrs bei den CPC's.

Vor allem bei der Grafikausgabe und bei Ansteuerung eines Druckers der mehr als 128 Zeichen bereitstellt, fehlt dieses Bit sehr.

Eine Lösung, die mit dem bereits bestehenden Druckerport auskommt -also ohne Eingriff in den Rechner selbst wegen der Garantie- und alle Zeichen von Ø-255 senden kann, stelle ich im folgenden vor. Eine kleiner Hardwarezusatz plus ein paar Bytes für das Treiberprogramm sind dabei unumgänglich.

```
**********************************
* Alle in diesem Buch  gezeigten *
* Hardcopy - Programme  benutzen *
*       diese Erweiterung !      *
**********************************
```



Geschlossen : A-Kanal;8-Bit
Offen : B-Kanal;7-Bit

Zu der Schaltung:

Wie sie leicht feststellen können, werden die unteren vier Bits (Bit Ø-3) aufgesplittet. Diese Bits werden zum einen durchgeschleift und zum anderen über einen Zwischenspeicher (D-FF,74LS75) auch für die neuen -software-erzeugten- Bits 4-7 bereitgestellt. Das "alte" Bit 4 des Computers ist nun das notwendige Taktsignal für das Latch. Das Signal STROBE wird erst dann ausgelöst, wenn sich alle acht Bit auf diese Weise am Port "breitgemacht" haben. Wir haben jetzt eine vollständig andere Ansteuerung, um trotzdem noch fertige Software benützen zu können, habe ich noch eine Handumschaltung (74LS157) von 7 auf 8 Bit vorgesehen, um nicht jedesmal umstecken zu müssen.

Das Treiberprogramm:

Das Programm wird in die Sprungtabelle des CPC eingepatcht, so daß nun jede Druckerausgabe über die neue Routine läuft !

Es teilt das zu druckende Zeichen im Akku in zwei Halbbytes auf, sendet zuerst das obere Nibble, und einen kurzen Impuls auf Bit 4 zum Zwischenspeichern im Latch. Das untere Nibble wird dann im Akku bereitgestellt und von der normalen ROM- Routine ab Adresse Ø7F8 (beim 464) mit dem STROBE-Impuls gesendet, einschließlich dem Warten auf BUSY.

Um den Treiber wieder auszuklinken, rufen Sie bitte den CALL BD28 auf, der den indirekten Sprung wieder auf die Original-Routine zeigen läßt.

Pass 1 errors: ØØ

```
                    1Ø ;Patch fuer 8-Bit-Centronics
                    2Ø ;direkt unter HIMEM (+Floppy).
                    3Ø ;
BDF1                4Ø indjmp: equ  #bdf1
                    5Ø ;
A653                6Ø        org  42619-4Ø
                    7Ø ;Initialisieren
A653  2AF2BD        8Ø        ld   hl,(indjmp+1)
A656  2279A6        9Ø        ld   (retour+1),hl
A659  2160A6       1ØØ        ld   hl,bit8
A65C  22F2BD       11Ø        ld   (indjmp+1),hl
A65F  C9           12Ø        ret
                   13Ø ;
                   14Ø ;neue Ausgaberoutine
                   15Ø ;ist ueber Druckerindirektion
                   16Ø ;#BDF1 eingeschleift !
                   17Ø ;
A660  Ø6EF         18Ø bit8:  ld   b,#ef
A662  F5           19Ø        push af
A663  CB3F         2ØØ        srl  a
A665  CB3F         21Ø        srl  a
A667  CB3F         22Ø        srl  a
A669  CB3F         23Ø        srl  a
A66B  ED79         24Ø        out  (c),a
A66D  CBE7         25Ø        set  4,a
A66F  ED79         26Ø        out  (c),a
A671  CBA7         27Ø        res  4,a
A673  ED79         28Ø        out  (c),a
```

```
A675  F1        290         pop  af
A676  E60F      300         and  #0f
                310 ;Jump wird berichtigt
                320 ;je nach System !
A678  C3F807    330 retour: jp   #07f8

Pass 2 errors: 00
```

Netzteil

Um bei umfangreicheren Schaltungen die rechnereigene 5 Volt Versorgung nicht zu überlasten und um zusätzliche Spannungen zur Verfügung zu haben, ist ein externes Netzteil vorzusehen.

Das vorgeschlagene Netzteil liefert folgende Spannungen:

+5 Volt	1 A	
+9 Volt	20 mA	(für V24)
-9 Volt	20 mA	" "
+25 Volt	70 mA	(für EPROM'er)

Sowie die gleichgerichtete und geglättete Spannung von ca. 12 Volt vor dem Spannungsregler 7805 für Relais etc..

Aus Sicherheitsgründen sollten die 220 Volt führenden Teile berührungssicher gemacht und das ganze Netzteil in ein Gehäuse gebaut werden (Lüftungsschlitze nicht vergessen !).

Der Regler 7805 muß mit einem nicht zu klein dimensionier- ten Kühlblech gekühlt werden. Die Verbindungsleitungen der 5-Volt Spannung sowie von Masse sollten einen Querschnitt von 1 qmm haben.

Die Reihenfolge beim Einschalten der verschiedenen Spannungsquellen sollte so sein:

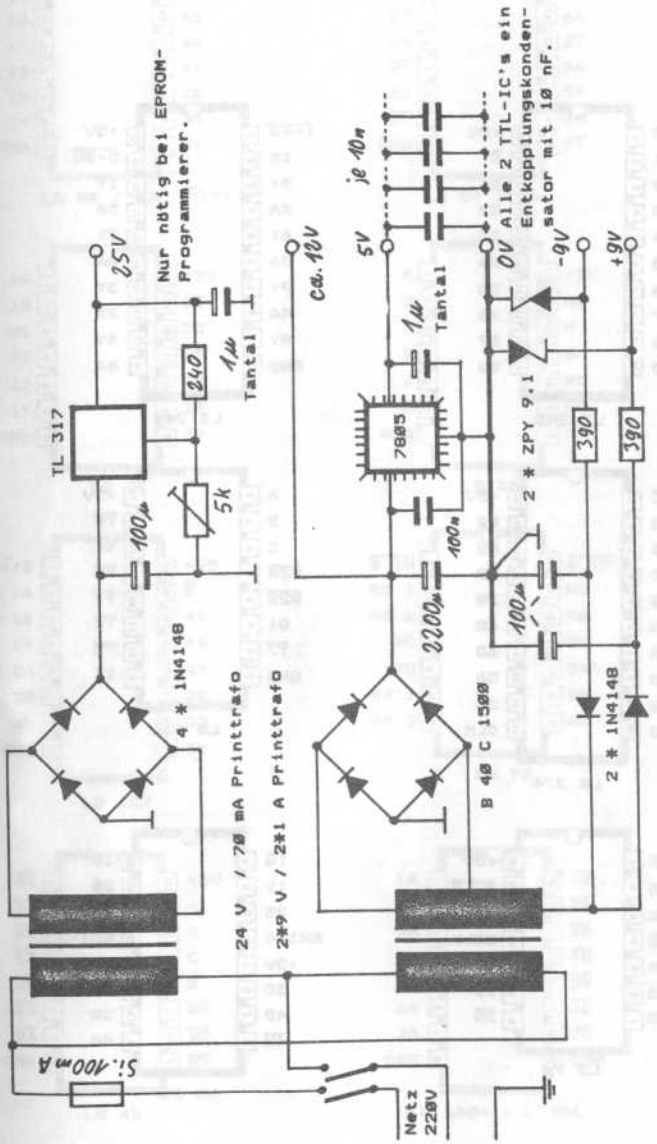
1. Floppystation(en)
2. Erweiterung
3. Bildschirm
4. Rechner

Das Ausschalten sollte in umgekehrter Richtung erfolgen.

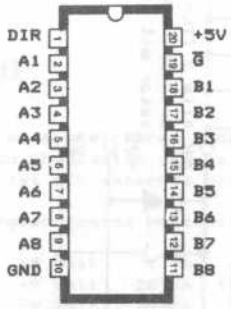
Den Schaltplan des Netzteiles sehen Sie auf der nächsten Seite.

Damit sind wir beim Schluss des Buches, es folgen noch auf weiteren Seiten die Anschlußbelegungen der verwendeten IC's.

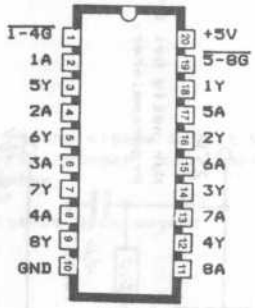
****
****
****



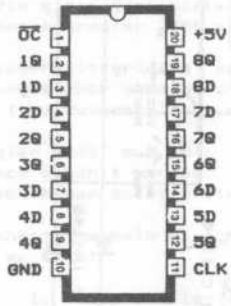
IC Ansichten



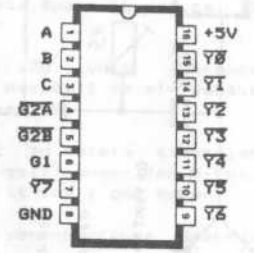
LS 245



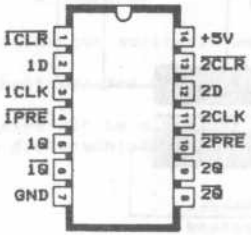
LS 244



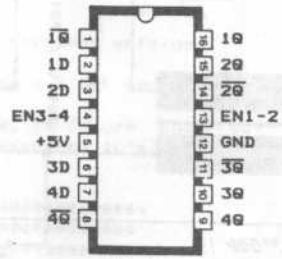
LS 374



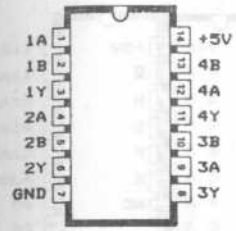
LS 138



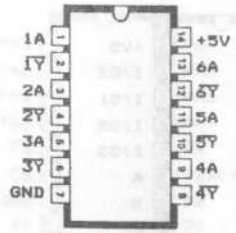
LS 74



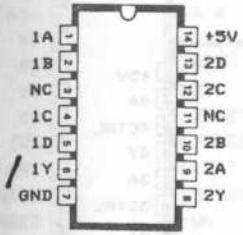
LS 75



LS 00 / 32 (Y/Y)

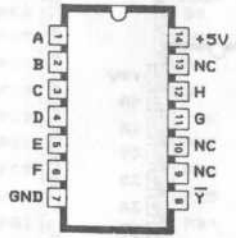


LS 04 / 06

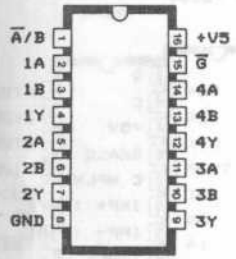


LS 21

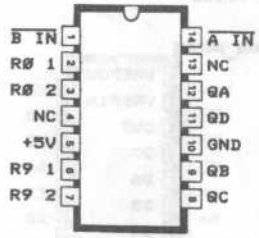
LS20 / 4Y



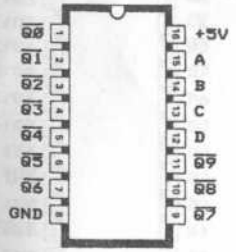
LS 30



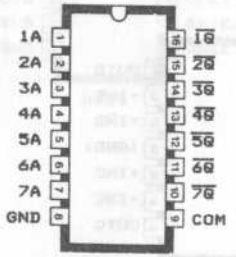
LS 157



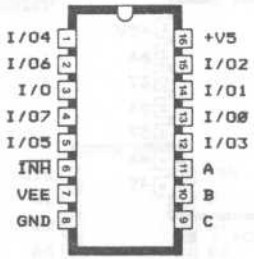
LS 90



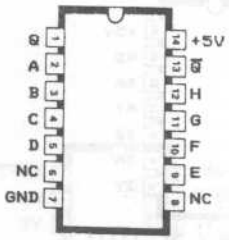
LS 42



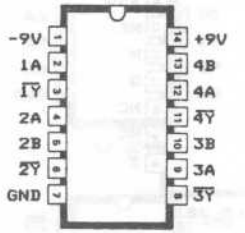
ULN 2004 / L 204



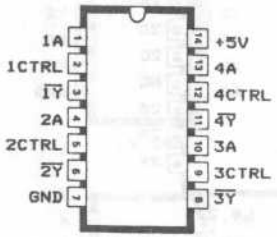
CD 4051



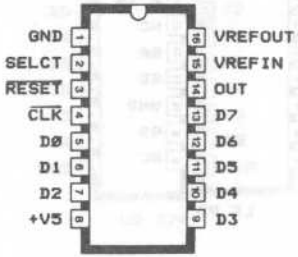
CD 4068



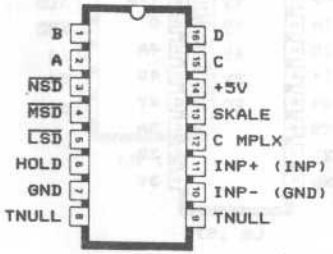
SN 75188



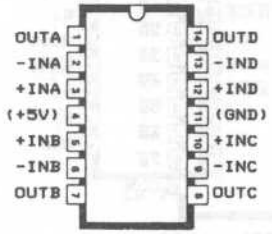
SN 75189



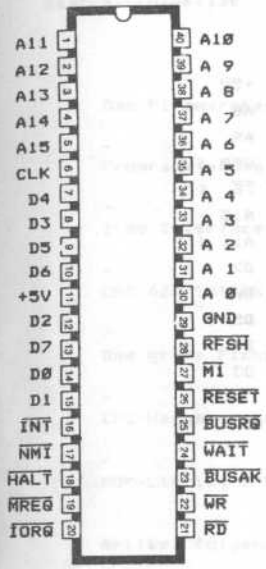
ZN 425 E



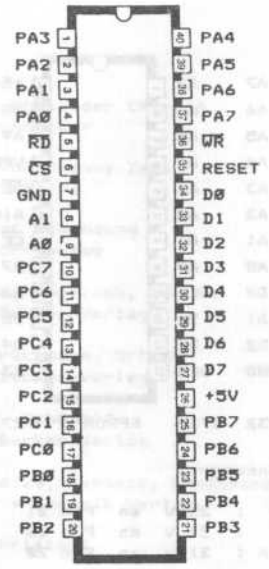
CA 3162



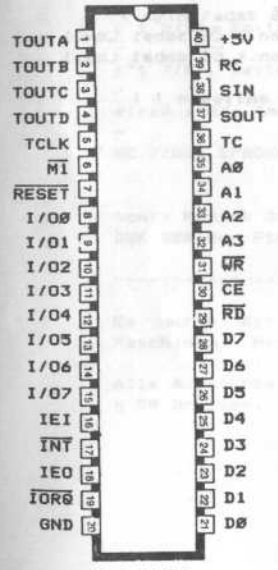
LM 324



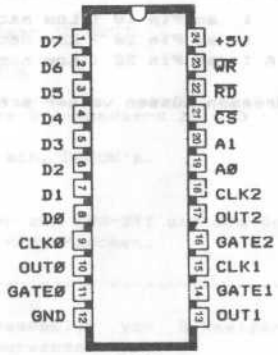
Z-8Ø



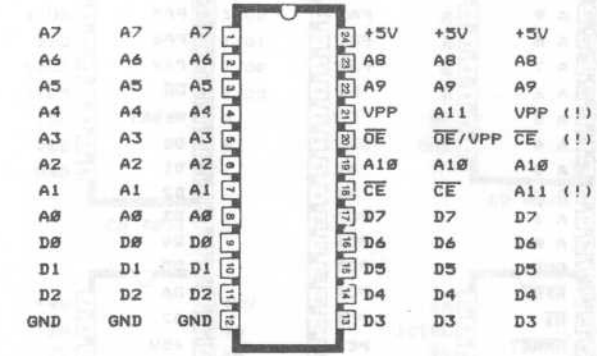
8255



Z-8Ø-STI



8253



Programmierspannungen:

2716	:	25 V	an	Pin 21
2732	:	21 V	an	Pin 20
2732 A	:	21 V	an	Pin 20

Programmierimpuls: (50 mS)

2716	:	an Pin 18 (Low nach High; $\overline{OE}$ dabei High.)
2732	:	an Pin 20 (Low nach Prg.Spann.; $\overline{CE}$ dabei Low.)
2732 A	:	an Pin 20 (Low nach Prg.Spann.; $\overline{CE}$ dabei Low.)

(Daten und Adressen müssen vorher schon stabil anliegen !)

Literaturhinweise

Das Firmwarehandbuch des Schneider CPC-464
von Schneider
-
Programmierung des Z 80 von Rodnay Zaks
Sybex-Verlag
-
Z-80 Interface-Technik und Anwendung
Elektor-Verlag
-
CPC 464 INTERN, Brückmann, Englisch, Gerits
Data-Becker-Verlag
-
Das große Floppy-Buch, Brückmann, Schieb
Data-Becker-Verlag
-
CPC-Harwareerweiterungen, Schüssler
Data-Becker-Verlag
-
ROM-Listing CPC 464,664,6128, Janneck, Mossakowski
Markt + Technik Verlag

Artikel folgender Zeitschriften:

c't 4/84 und 8/84, Beiträge zu dem A/D-Wandler mit CA 3162
-
c't 8/84, Beitrag zum Timer IC 8253
-
c't 7/83, Beitrag zum 8255
-
elrad 1/83, Anwendung des D/A-Wandlers ZN 425
-
MC 7/83, EPROMer für versch. EPROM's.

sowie Mostek Datenblätter zum Z80-STI und dem Schaltplan zum DVM 300 der Firma RADIO-RIM, München.

Es wurde der HiSoft-Assembler von Schneider bei allen Maschinen - Programmen verwendet.

Alle Ausdrucke erfolgten mit einem EPSON-kompatiblen SHINWA M-80 Drucker.