Computer Shop

Alfred Görgens
Karl-Heinz Koch

ATARI BASIC-Trickkiste

Springer Basel AG

B

Computer Shop
Band 22

Alfred Görgens / Karl-Heinz Koch

ATARI BASIC-Trickkiste

Springer Basel AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Görgens, Alfred:
ATARI BASIC-Trickkiste / Alfred Görgens ; Karl-Heinz Koch. – Basel ; Boston ; Stuttgart : Birkhäuser, 1985.
(Computer-Shop ; Bd. 22)
ISBN 978-3-7643-1663-1
NE: Koch, Karl-Heinz:; GT

Ursprünglich erschienen bei Birkhäuser Verlag, Basel 1985
Umschlaggestaltung: Bruckmann & Partner, Basel

ISBN 978-3-7643-1663-1 ISBN 978-3-0348-6758-0 (eBook)
DOI 10.1007/978-3-0348-6758-0

Inhalt

Vorwort

Dieses Buch richtet sich an alle, die bereits gewisse Erfahrungen mit ihrem ATARI und BASIC gesammelt haben und nun nach Informationen suchen, um selbstgeschriebene Programme noch effektiver und vielseitiger zu machen, ohne gleich eine neue Programmiersprache zu erlernen.

Es war an vielen Stellen schwierig, die Grenze zu finden zwischen dem, was an Wissen vorausgesetzt werden soll und den Erläuterungen, denn je tiefer man in die Materie eindringt, um so mehr hängen die verschiedenen Kenntnisse zusammen. Man kann auch ein Stück Tuch nicht beschreiben, indem man einen Faden erklärt und doch muß man irgendwo anfangen.

Wenn Sie irgendetwas nicht verstehen, dann sollten Sie unbedingt die Beispielprogramme eingeben und die Erläuterungen zu den einzelnen Zeilen gründlich lesen. Manches Mal kann es auch hilfreich sein, einen Abschnitt zweimal zu lesen. Das eine oder andere sehen Sie vielleicht erst im richtigen Zusammenhang, wenn Sie das ganze Buch durchgearbeitet haben.

Besonders wenn im Kapitel Grafik die Display-Liste behandelt wird, stößt dieses Buch an seine Grenzen. Nicht nur, daß hier die Schwelle zur Maschinensprache betreten wird, in diesen Aspekt spielt auch noch die gesamte Video- und Bildschirmtechnik hinein.

Das Buch gliedert sich in zwei Teile. Zuerst werden die Bearbeitung von Text, Sound und Grafik behandelt, dann die Peripherie-Geräte Drucker und Diskettenstation. Im Anhang finden Sie viele nützliche Tabellen für die Programmierarbeit.

I Text

Böse Zungen behaupten, daß ATARI-Computer kaum Möglichkeiten für String-Manipulationen bieten und damit zur Text-Verarbeitung ungeeinget seien.

Das Gegenteil ist der Fall. Als einzige Heim-Computer ihrer Preisklasse bieten die ATARI-Modelle einen internationalen Zeichensatz, der durch POKE 756,204 eingeschaltet werden kann. Mit der Taste CONTROL + der gewünschten Zeichentaste können Sie dann Ä,Ö,Ü, á,é,ú usw. aufrufen. Mit POKE 756, 224 kehren Sie zum normalen Zeichensatz zurück.

In diesem Kapitel erfahren Sie, wie man mit dem ATARI-Computer die vielen verschiedenen String-Manipulationen durchführt, die für die Text-Verarbeitung, aber auch zur Lösung anderer Programm-Aufgaben wichtig sind.

Dazu zählen das Verketten von String-Variablen bzw. das Verketten von Teil-Strings, das Selektieren von einzelnen Buchstaben oder Wörtern aus Texten (z.B. zum Erstellen eines Stichwort-Verzeichnisses) und die künstliche STRING$-Funktion, mit der man durch nur eine Eingabe einen String erzeugen kann, der aus n-mal dem eingegebenen Zeichen besteht.

Besonders unterhaltsam ist ein Abschnitt über das Generieren von Zufallsbuchstaben und Zufallswörtern. Diese Prinzipien können Sie z.B. in fortgeschrittenen Spielen einsetzen, bei denen der Spieler einen "Geheim-Code" knacken muß, um an Informationen heranzukommen, um Zutritt zu einem Geheimraum zu bekommen usw.

Neben diesen Tips und Tricks im Umgang mit String-Variablen finden Sie in diesem Kapitel noch Erläuterungen zu speziellen String-Funktionen wie ASC, STR$ und VAL, die zur Bewältigung fortgeschrittener Programm-Aufgaben eingesetzt werden können.

Zum Abschluß bieten wir Ihnen noch eine 3-Zeilen-Utility, mit der Sie bequem die Variablen-Namen aus Ihren BASIC-Programmen verändern können, ohne das ganze Listing durchstöbern zu müssen. Durch eine einzige Anweisung wird ein geänderter Variablen-Name automatisch an alle Stellen Ihres BASIC-Programms eingefügt, an denen er verarbeitet werden soll.

1.1 String-Manipulationen

Mit *normalen* String-Variablen haben Sie sicher schon gearbeitet (z.B. DIM A$(4):A$="TEXT":PRINT A$). In diesem Abschnitt lesen Sie, wie man Strings verkettet, eine künstliche STRING$-Funktion erzeugt (also die beliebige Wiederholung eines Zeichen durch nur eine Angabe) und wie man Zufallswörter generiert. Außerdem wird auf die BASIC-Funktionen VAL, STR$ und ASC eingegangen.

1.1.1 Strings verketten und selektieren

Wenn Sie schon einmal eine String-Variable definiert haben, die über drei Bildschirm-Zeilen hinausgehen sollte, werden Sie sechs Zeichen vor Ende der dritten Zeile einen Kontrollton gehört haben. Das bedeutet gewissermaßen "Bis hierher und nicht weiter". Selbst wenn Sie den Text länger schreiben, werden beim PRINTen nur die Zeichen bis zum Punkt des Kontrolltons ausgegeben.

Falls Sie also längere Strings haben wollen, müssen Sie sie miteinander verketten. Dazu folgendes Beispiel:

```
10 DIM A$(300),B$(100),C$(100)
20 A$="Dies ist ein Test, mit dem festgestellt wird,
wieviele Strings man miteinander verbinden kann. "
30 B$="Dies ist ein Test, mit dem festgestellt wird,
wieviele Strings man miteinander verbinden kann. "
40 C$="Dies ist ein Test, mit dem festgestellt wird,
wieviele Strings man miteinander verbinden kann. "
50 ? A$:?
60 A$(LEN(A$)+1)=B$
70 ? A$:?
80 A$(LEN(A$)+1)=C$
90 ? A$:?
```

LEN (A$) berechnet die Anzahl der Zeichen von A$. Mit +1 wird erreicht, daß B$ auf der ersten Position hinter A$ beginnt. Wenn Sie z.B. +0 angeben, wird das letzte Zeichen von A$ gelöscht und vom ersten Zeichen des B$ besetzt. Nach RUN sehen Sie, wie die Strings nacheinander verkettet wurden und nun fast den gesamten Bildschirm ausfüllen.

Neben dem einfachen Verketten von Strings ist es auch möglich, nur String-Teile zu verbinden. Dazu müssen Sie genau angeben, welche Zeichen der einzelnen String-Variablen verkettet werden sollen; z.B. A$(1,3) bedeutet "vom ersten bis dritten Zeichen". Versuchen Sie folgendes Beispiel:

```
10 DIM A$(30),B$(17)
20 A$="Dies ist der Original Text"
30 B$="umgestaltete Text"
40 ? A$:?
50 A$(LEN(A$(1,13))+1)=B$
60 ? A$
```

Hier wird nicht die Gesamtlänge von A$ mit B$ verkettet, sondern nur das erste bis dreizehnte Zeichen von A$. Nach RUN sehen Sie folgendes auf dem Bildschirm:

```
Dies ist der Original Text
Dies ist der umgestaltete Text
```

Wenn Sie Strings verketten, müssen Sie darauf achten, daß Sie für den Haupt-String genügend Zeichen reservieren. Beim obenstehenden Beispiel wurden für A$ 30 Zeichen DIMensioniert, obwohl A$ selbst nur 26 Zeichen beansprucht. Durch die Verkettung mit B$ nimmt jedoch die Anzahl der Zeichen zu. Folgende Skizze verdeutlicht dies:

A$ =	Dies ist der Original Text
Zeichen:	1234567890... ...26

B$ =	umgestaltete Text
Zeichen:	1234567890... ...17

A$(1,13)+B$ =	Dies ist der umgestaltete Text
Zeichen:	1234567890... ...30

Falls Sie nicht zwei Strings verketten, sondern aus einem String bestimmte Teile herausfiltern wollen, so gibt es selbstverständlich auch hierfür eine Lösung. Versuchen Sie folgendes Programm:

```
10 DIM A$(26)
20 A$="Dies ist der Original Text"
30 ? A$:?
40 A$(LEN(A$(1,13))+1)=A$(23,26)
50 ? A$
```

Nach RUN sehen Sie auf dem Bildschirm:

```
Dies ist der Original Text
Dies ist der Text
```

Hier wurden also nicht zwei String-Variablen miteinander verbunden, sondern zwei Teile eines einzelnen Strings zusammengefügt und das Wort "Original" herausgefiltert. Es ist nun auch möglich, solche herausgefilterten Teilstrings mit

anderen Teilstrings zu verketten. Im folgenden Beispiel werden verschiedene Manipulationen mit Strings, Teilstrings und herausgefilterten Teilstrings durchgeführt. Sie können nach diesem Muster eigene Text-Spielchen nach Lust und Laune programmieren.

```
10 DIM A$(53),B$(23),C$(28)
20 A$="Dies ist der Original Text"
30 B$="neue, umgestaltete Text"
40 C$=", herausgefilterte Textsalat"
50 ? A$:?
60 A$(LEN(A$(1,13))+1)=B$
70 ? A$:?
80 A$(LEN(A$(1,13))+1)=A$(20,36)
90 ? A$:?
100 A$(LEN(A$(1,25))+1)=C$
110 ? A$:?
120 A$(LEN(A$(1,9))+1)=A$(45,53)
130 ? A$
```

Als Ergebnis sehen Sie folgendes auf dem Bildschirm:

```
Dies ist der Original Text
Dies ist der neue, umgestaltete Text
Dies ist der umgestaltete Text
Dies ist der umgestaltete, herausgefilterte Textsalat
Dies ist Textsalat
```

Nach diesen etwas banalen Vorstudien folgt nun der "Ernst des Programmierens". Es gibt nämlich auch einen sehr nützlichen Einsatz des Verkettens, Vergleichens oder Ausfilterns von Strings und Teilstrings. Zum Beispiel können Sie aus einer wahllos zusammengestellten Wörtermenge alle Wörter heraussuchen lassen, die mit einem bestimmten Buchstaben beginnen. Dazu das folgende Selektier-Programm:

```
10 DIM A$(1),B$(20),C$(1)
15 ? CHR$(125)
20 ? "Geben Sie einen Suchbuchstaben ein";
30 INPUT A$
40 ? :? "Folgendes ist unter diesem Buchstaben"
50 ? "in der Datei gespeichert:":?
60 READ B$:IF B$="ENDE" THEN GOTO 90
70 IF B$(1,1)=A$ THEN ? B$:C$=A$
80 GOTO 60
90 IF C$<>A$ THEN ? "NICHTS"
100 RESTORE
110 ? :? :? "Noch einmal (J/N)? ":INPUT A$
120 IF A$<>"J" THEN END
```

```
130 GOTO 15
1000 DATA Afghanistan,Belgien,Albanien
1010 DATA Chile,Indonesien,Jamaika
1020 DATA Korea,Libyen,USA,Deutschland
1030 DATA Frankreich,Schweiz,Zypern
1040 DATA Griechenland,Tunesien,Taiwan
1050 DATA China,Ungarn,Mexiko,Panama
1060 DATA ENDE
```

15: CHR$(125) löscht den Bildschirm.

60: Nachdem Sie Ihre Eingabe gemacht haben (Zeile 30), liest der Rechner die einzelnen Wörter der DATA-Zeilen. Wenn das gelesene Wort = ENDE ist, soll das Programm in Zeile 90 fortgesetzt werden.

70: Wenn das erste Zeichen des aktuell eingelesenen Wortes mit dem eingegebenen Suchbuchstaben übereinstimmt, soll das Wort auf dem Bildschirm ausgePRINTet werden. In diesem Fall nimmt C$ das Zeichen A$ an (vergl. Zeile 90).

80: Solange das Wort ENDE nicht erreicht ist, springt der Rechner zur Zeile 60 zurück und liest das nächste Wort aus den DATA-Zeilen.

90: Wenn keines der Wörter aus den DATA-Zeilen mit dem eingegebenen Suchbuchstaben beginnt, bleibt die Bedingung aus Zeile 70 unerfüllt. Dadurch hat auch C$ nicht das Zeichen von A$ angenommen. In diesem Fall wird das Wort "NICHTS" auf dem Bildschirm ausgegeben.

110 bis 130: Der Rechner fragt, ob das Programm noch einmal starten soll.

Wenn Sie selbst eine ähnliche Datei in DATA-Zeilen ablegen wollen, müssen Sie darauf achten, daß zwischen den einzelnen Wörtern kein Leerzeichen steht. Beim READ-Befehl wird das Leerzeichen als erstes Zeichen von B$ erkannt, so daß das Programm nicht richtig arbeitet.

Eine weitere nützliche Anwendung der oben gezeigten String-Manipulation zeigt das folgende Programm. Hier können aus beliebigen Texten bestimmte Wörter selektiert werden. Dies ist z.B. notwendig, wenn ein Stichwortverzeichnis für ein Buch, eine Diplomarbeit etc. angefertigt werden muß.

Die nachfolgend aufgeführten DATA-Zeilen sind natürlich nur als Prinzip gedacht; "Seite 1", "Seite 2" usw. sollen für tatsächliche Textseiten stehen.

Sie müssen das zu selektierende Wort selbstverständlich in der gleichen Weise eingeben, wie es in den Texten vorkommt

(also in Groß- und Kleinbuchstaben). Geben Sie z.B. nach RUN (RETURN) das Wort "Computer" ein. Der Rechner PRINTet dann "Seite 1", "Seite 3", "Seite 5". Auch nach allen anderen Wörtern können Sie suchen lassen (das Programm macht nämlich Spaß).

```
10 DIM A$(20),B$(100),C$(20)
20 ? CHR$(125)
30 ? "Welches Wort soll selektiert werden"
40 INPUT A$
50 ? :? "Das eingegebene Wort kommt auf folgen-den Se
iten vor:":?
60 TRAP 100:READ B$:IF B$="ENDE" THEN 110
70 FOR X=1 TO LEN(B$)
80 IF B$(X,X+LEN(A$)-1)=A$ THEN ? B$(1,7):C$=A$
90 NEXT X
100 GOTO 60
110 IF C$<>A$ THEN ? "Auf keiner Seite!"
120 RESTORE
130 ? :? :? "Noch einmal (J/N) ";:INPUT A$
140 IF A$<>"J" THEN END
150 GOTO 20
1000 DATA Seite 1: In diesem Text kommt das Wort Comp
uter vor
1010 DATA Seite 2: Dies ist ein Probetext
1020 DATA Seite 3: Ein Computer ist ein elektronische
r Rechner
1030 DATA Seite 4: Anstelle dieser Zeile steht eine B
uchseite
1040 DATA Seite 5: Auch in diesem Text steht das Wort
 Computer
1050 DATA ENDE
```

70: Da jeder String, der auf das eingegebene Wort hin untersucht werden soll, eine unterschiedliche Anzahl von Zeichen hat (vergl. DATA-Zeilen), durchläuft die FOR-NEXT-Schleife jeweils die Anzahl der Zeichen des aktuellen Strings; also FOR X=1 TO LEN(B$).

80: Bei jedem Durchlauf der FOR-NEXT-Schleife erhöht sich X um 1; d.h. der gesamte String wird Zeichen für Zeichen nach dem eingegebenen Wort abgesucht; nämlich von X bis X + der Länge von A$. Die Angabe -1 ist notwendig, da X bereits das erste Zeichen von A$ ist. Wenn bei dieser Suche der Teilstring von B$ mit dem eingegebenen Suchwort (A$) übereinstimmt, soll der Rechner vom aktuellen B$ die Zeichen 1 bis 7 PRINTen; das entspricht jeweils dem Wort "Seite" und der Seitenzahl. Auch hier soll, wie im vorhergehenden Pro-

gramm bereits beschrieben) C$ die Zeichen von A$ annehmen (vergl. Zeile 110).

110: Wenn das gesuchte Wort in keinem String enthalten ist, PRINTet der Rechner "Auf keiner Seite".

1000 bis 1050: Wie Sie sehen, müssen DATA-Zeilen nicht immer aus einem einzelnen Zeichen, Wort oder Wert bestehen. Bei String-DATAs werden Zwischenräume als Leerzeichen interpretiert und ohne Probleme verarbeitet.

1.1.2 Suchwort-Verzeichnis

Wenn Sie Ihre "richtigen" Texte auf Diskette abspeichern und anschließend feststellen wollen, auf welchen Seiten sich bestimmte Wörter befinden, müssen Sie diese Aufgabe etwas anders lösen als in dem oben gezeigten Beispiel. Mit dem folgenden Programm können Sie beliebige Texte auf Diskette schreiben und prüfen, auf welchen Seiten sich gesuchte Wörter befinden.

Das Programm ist so angelegt, daß jede einzelne Textseite mit einem eigenen Dateinamen (SEITE1, SEITE2, SEITE3 usw.) abgespeichert wird. Zum Erstellen dieser Textfiles müssen Sie nur RUN (RETURN) und die Seite eingeben, die geschrieben werden soll. Danach erscheint auf dem Bildschirm ein Fragezeichen. Zum Textschreiben tippen Sie am Anfang und Ende jeder Zeile einen An- bzw. Ausführungsstrich ("). Der Computer übernimmt eine automatische Zeilen-Nummerierung und setzt in jede Zeile die Anweisung LPRINT. Sie können, falls Sie nicht mit einem Drucker arbeiten, auch nur eine PRINT-Anweisung geben (s. Zeile 390).

Zum Abbruch des Programms drücken Sie die HELP-Taste. Das Textfile wird dann automatisch in den Rechner geladen. Das eigentliche Text- und Selektier-Programm wird dabei nicht gelöscht.

Falls sich bei den Eingaben Programmier-Fehler eingeschlichen haben sollten, so werden diese jetzt automatisch auf dem Bildschirm ausgegeben. Sie müssen sie korrigieren und das Textfile noch einmal unter seinem Namen auf Diskette LISTen (jedoch nur von Zeile 1000 bis Ende, damit das Text- und Selektier-Programm nicht mitgeLISTet wird).

Danach können Sie mit dem Selektieren beginnen. Tippen Sie dazu GOTO 20 (RETURN). Der Rechner gibt Ihnen dann alle Seiten-Angaben, in denen das gesuchte Wort auftritt. Zum Schreiben einer neuen Seite geben Sie wieder RUN ein.

```
10 GOTO 280
20 CLR :DIM A$(20),B$(150),N$(17)
30 Z=0
40 N$="D:SEITE"
50 ? CHR$(125)
60 ? "Welches Wort soll selektiert werden"
70 INPUT A$
80 ? :? "Das eingegebene Wort kommt auf folgen-den Se
iten vor:":?
90 Z=Z+1:N$(LEN(N$(1,7))+1)=STR$(Z)
100 CLOSE #1
110 TRAP 200
120 OPEN #1,4,0,N$
130 POKE 195,0
140 TRAP 200
150 INPUT #1;B$
160 FOR X=1 TO LEN(B$)
170 IF B$(X,X+LEN(A$)-1)=A$ THEN ? N$(3)
180 NEXT X
190 GOTO 150
200 IF PEEK(195)=136 THEN GOTO 90
210 IF PEEK(195)=170 THEN GOTO 240
220 GOTO 140
230 GOTO 90
240 ? :? "Noch einmal (J/N) ";
250 INPUT A$
260 IF A$<>"J" THEN END
270 GOTO 20
280 DIM T$(255),N$(17)
290 POKE 732,0
300 ZN=990
310 N$="D:"
320 ? "Programm-Name "
330 INPUT T$
340 N$(LEN(N$)+1)=T$
350 CLOSE #1
360 OPEN #1,8,0,N$
370 ZN=ZN+10
380 INPUT T$
390 PRINT #1;ZN;" LPRINT ";T$;CHR$(155)
400 IF ZN=1500 THEN CLOSE #1:ENTER N$
410 IF PEEK(732)=17 THEN CLOSE #1:ENTER N$
420 GOTO 370
```

90: Z ist zunächst eine numerische Variable, die anschließend bei der Verkettung durch STR$ in eine String-Variable umgewandelt wird (vergl. Abschnitt 1.3 "Spezielle String-Funktionen"). Mit der Umwandlung wird erreicht, daß beim ersten Programm-Durchlauf der Dateiname SEITE1, beim zweiten Durchlauf SEITE2 usw. heißt. Auf diese Weise werden alle Textfiles, die mit SEITE beginnen, in die spätere Selektierung mit einbezogen.

100: OPEN, CLOSE, INPUT und andere Befehle, die den Datenaustausch zwischem Computer und Disketten-Station regulieren, werden ausführlich im Kapitel 5 "Disketten-Operationen" erklärt.

130 und 140: In Register 195 wird die Code-Zahl abgelegt, die beim Auftritt eines Fehlers auf dem Bildschirm ausgegeben wird (z.B. ERROR 9 - AT LINE...). Da dieses Register im Laufe dieses Programms abgefragt wird und der letzte ERROR bis zum nächsten in diesem Register gespeichert bleibt, muß es durch POKE 195,0 gelöscht werden. TRAP 200 bestimmt, daß der Rechner beim Auftritt eines ERRORs zur Zeile 200 springen soll. In diesem Programm ist man also nicht den möglichen Fehlern ausgeliefert; vielmehr werden sie für die Ausführung bestimmter Operationen genutzt.

150: Jede von Ihnen geschriebene Textzeile wird auf Diskette gespeichert.

160: Da jede Textzeile unterschiedlich lang ist, durchläuft die FOR-NEXT-Schleife die Anzahl der Zeichen des aktuellen B$.

170: Dies ist gewissermaßen das Kernstück des Programms: Mit der FOR-NEXT-Schleife wird der gesamte String Zeichen für Zeichen abgetastet und vom aktuellen Zeichen X bis X + der Länge des gesuchten Wortes mit A$ verglichen. Falls diese Zeichen übereinstimmen (das gesuchte Wort also auf dieser Seite zu finden ist), soll der Rechner den Dateinamen vom dritten Zeichen ab PRINTen. Dieser beginnt in jedem Fall mit SEITE. Die beiden ersten Zeichen des Dateinamen (D:) werden hier nicht benötigt.

200: Der ERROR 136 tritt immer auf, wenn der gesamte Textfile abgefragt wurde und kein INPUT mehr möglich ist. In diesem Fall soll der Rechner zur Zeile 90 springen und das nächste Textfile laden.

210: ERROR 170 tritt auf, wenn alle Files mit dem Namen SEITEn geladen wurden; d.h. die Selektierung beendet ist. In diesem Fall soll das Programm nicht einfach abgebrochen

werden. Vielmehr springt der Rechner zur Zeile 240, um das Suchen nach weiteren Wörtern zu ermöglichen.

290 bis 420: Hier befindet sich das Text-Eingabe-Programm. Alle Prinzipien dieses Programms, das auch eine automatische Zeilen-Nummerierung beinhaltet, finden Sie im Kapitel 4 "Disketten-Operationen" ausführlich beschrieben.

Das vorliegende Programm ist darauf eingerichtet, Files mit dem Namen SEITEn zu bearbeiten. Selbstverständlich ist es auch möglich, jeden anderen Programm-Namen oder File-Gruppen auf diese Weise für Wortselektierungen in den Rechner zu laden.

Nachfolgend finden Sie ein Beispiel, wie das gleiche Programm alle Files "abklappert", die den Namen BIRn.TXT tragen. Dabei befanden sich auf unserer Diskette Files von BIR1.TXT bis BIR20.TXT (der Text zu diesem Kapitel). Von dem vorhergehenden Programm müssen nur folgende Zeilen geändert werden:

```
20 CLR :DIM A$(20),B$(150),N$(17),NN$(5)
30 Z=0:NN$=".TXT"
40 N$="D:BIR"
90 Z=Z+1:N$(LEN(N$(1,7))+1)=STR$(Z):N$(LEN(N$)+1)=NN$
```

Mit NN$ wird eine weitere Variable DIMensioniert, die den Extender der File-Namen ".TXT" enthält.

90: Zur bereits bekannten Verkettung von Programm-Namen und Umwandlung von numerischen Variablen (Z) zum String wird der hier entstandene String noch zusätzlich mit dem Extender verkettet, so daß beim Programm-Durchlauf BIR1.TXT, BIR2.TXT usw. entsteht. Mit diesen Anregungen wird es Ihnen sicher gelingen, Ihre eigenen Programm-Namen in dieser Weise bearbeiten zu lassen und das Wort-Selektier-Programm nutzbringend einzusetzen.

1.2 Künstliche STRING$-Funktion

Mit der STRING$-Funktion kann man einer String-Variable das gleiche Zeichen zuweisen, ohne das Zeichen selbst z.B. 20 mal tippen zu müssen.

ATARI verfügt nicht über einen STRING$-Befehl. Mit einem kleinen Trick kann man jedoch diese Funktion künstlich erzeugen. Hier das Prinzip:

```
10 DIM A$(10)
20 A$="X"
30 A$(10)=A$:A$(2)=A$
40 ? A$
```

Nach RUN (RETURN) sehen Sie:

```
XXXXXXXXXX
```

Nachdem in Zeile 20 A$ als "X" definiert ist, wird in Zeile 30 bestimmt, daß das letzte Zeichen mit dem ersten identisch sein soll. Danach wird festgelegt, daß das zweite Zeichen ebenfalls mit dem ersten identisch sein soll. Beim PRINTen weist der Rechner dann automatisch A$(3) das Zeichen von A$(2), A$(4) das Zeichen von A$(3) zu usw.

Das klingt zunächst etwas umständlich; solange die STRING$-Funktion in einem Programm nur einmal angewendet werden soll, ist die oben gezeigte Programmierung tatsächlich aufwendiger als das Zeichen selbst zu schreiben. Wenn jedoch ein STRING$ immer wieder verwendet wird (z.B. als Rahmen in Menü-Überschriften), kann man nach dem oben gezeigten Prinzip Zeit und vor allem Speicherplatz sparen. Versuchen Sie folgendes Programm:

```
10 DIM A$(38)
20 A$="*":A$(38)=A$:A$(2)=A$
40 ? A$
50 ? "          ANWENDUNGS-BEISPIEL"
60 ? :? A$
```

Auf dem Bildschirm sehen Sie nach RUN (RETURN):

```
**************************************
          ANWENDUNGS-BEISPIEL

**************************************
```

Nach diesem Muster können Sie jedes beliebige Zeichen als STRING$ definieren und in Ihren Programmen nach Lust und Laune einsetzen.

1.3 Spezielle String-Funktionen

Außer den verschiedenen Manipulationen mit String-Variablen, die bisher gezeigt wurden, bietet ATARI noch einige andere Funktionen, die bestimmte Programm-Aufgaben erleichtern können.

Hierzu zählen die Anweisungen ASC, STR$ und VAL. ASC gibt den ASCII-Wert für das erste Zeichen eines Strings; z.B.:

```
10 DIM A$(20),B$(20),C$(20)
20 A$="Erster Text"
30 B$="Zweiter Text"
40 C$="Dritter Text"
50 X=ASC(A$)
60 Y=ASC(B$)
70 Z=ASC(C$)
80 ? X,Y,Z
```

Als Ergebnis sehen Sie:

```
69          90          68
```

Mit der STR$-Anweisung (nicht zu verwechseln mit STRING$) können Sie einen numerischen Wert in einen String umwandeln. Im Wort-Selektier-Programm Seite (Zeile 90) wurde dies bereits angewendet. Hier noch ein einfacheres Beispiel:

```
10 DIM A$(5),B$(5),C$(5),D$(5)
20 X=65:Y=8000:Z=2*15
30 A$=STR$(X)
40 B$=STR$(Y)
50 C$=STR$(Z)
60 D$=STR$(12345)
70 ? A$,B$,C$,D$
```

Mit VAL erreichen Sie das genau Gegenteil von STR$. Durch diese Anweisung wird ein String in einen numerischen Wert umgewandelt. Versuchen Sie folgendes Programm:

```
10 DIM A$(5),B$(4)
20 A$="12345"
30 B$="8000"
40 X=VAL(A$)
50 Y=VAL(B$)
60 Z=VAL("987")
70 ? X,Y,Z
```

Nach RUN (RETURN) sehen Sie folgendes Ergebnis auf dem Bildschirm:

```
12345       8000        987
```

Im folgenden Beispiel wird der numerische Wert, der sich aus ASC(A$) ergibt, durch STR$ in einen String umgewandelt und B$ zugewiesen. In Zeile 40 wird der ASCII-Wert des achten Zeichens von A$ zum String umdefiniert und C$ zugewiesen.

In den Zeilen 60 und 70 wird STR$ durch CHR$ ersetzt, so daß B$ und C$ nicht als Werte, sondern als Buchstaben erscheinen.

Umwandlungen dieser Art sind in allen Sortier-Programmen nützlich (z.B. wenn man bei einer Adressen-Verwaltung Orte nach Postleitzahlen selektieren will).

```
10 DIM A$(11),B$(2),C$(3)
20 A$="Erster Text"
30 B$=STR$(ASC(A$))
40 C$=STR$(ASC(A$(8)))
50 ? A$,B$,C$
60 B$=CHR$(ASC(A$))
70 C$=CHR$(ASC(A$(8)))
80 ? A$,B$,C$
```

Nach RUN (RETURN) sehen Sie folgendes Ergebnis auf dem Bildschirm:

```
Erster Text          69          84
Erster Text          E           T
```

1.4 Zufallsbuchstaben - Zufallswörter

Eine weitere Aufgabe im Umgang mit Strings ist das Erzeugen von Zufallsbuchstaben oder -wörtern. Dies kann z.B. in Spielen nützlich sein, wenn ein Geheimwort o.ä. eingegeben werden muß, damit sich dem Spieler eine Tür öffnet usw. Es wäre ja sinnlos, solch ein Wort vorher zu definieren, da man dann die Lösung bereits kennt.

Die einfachste Form, einen Zufallsbuchstaben zu erzeugen, zeigt folgendes Listing:

```
10 X=INT(26*RND(0))+65
20 ? CHR$(X)
```

Zunächst wird für die 26 möglichen Buchstaben des Alphabets eine Zufallszahl erzeugt. Da das Alphabet im ASCII-Code mit 65 (für A) beginnt, muß der Zufallszahl noch 65 hinzugerechnet werden.

Falls aus dem Zufallsbuchstabe ein Zufallswort werden soll, muß das Programm folgendermaßen ergänzt werden:

```
5 FOR Y=1 TO 5
10 X=INT(26*RND(0))+65
20 ? CHR$(X);
30 NEXT Y
```

Damit haben Sie nun zwar ein Zufallswort; jedoch ist dieses Wort als String-Variable nirgends registriert und damit in einem weiteren Programm-Ablauf verloren. Wenn Sie also mit einem zufällig generierten Wort mehrmals operieren wollen, müssen Sie es in einem String speichern. Im folgenden Beispiel wird diese Aufgabe gelöst:

```
10 DIM A$(5)
20 FOR X=1 TO 5
30 A$(X,X)=CHR$(INT(26*RND(0))+65)
40 NEXT X
50 ? A$
```

Mit jedem Durchlauf der FOR-NEXT-Schleife erhöht sich der Wert von X um 1. Dadurch wird A$ nacheinander mit fünf Zufallsbuchstaben "aufgefüllt". Die Funktionen aus den Zeilen 10 und 20 der beiden zuvor gezeigten Programme sind hier in Zeile 30 zusammengefaßt.

Zum Abschluß dieses Abschnitts noch ein kleines Wortspiel, bei dem das oben gezeigte Prinzip angewendet wird.

```
10 DIM A$(3),B$(3)
20 FOR X=1 TO 3
30 A$(X,X)=CHR$(INT(2*RND(0))+65)
40 NEXT X
50 ? CHR$(125)
60 ? "Geben Sie den Geheim-Code ein"
70 ? "(Drei Buchstaben A oder B)"
80 ? :INPUT B$
90 IF B$=A$ THEN ? "Zutritt gestattet!"
100 IF B$<>A$ THEN ? "Zutritt verweigert!"
110 ? :? :? "Noch einmal (START oder N)?"
120 IF PEEK(53279)=6 THEN GOTO 20
130 IF PEEK(764)=35 THEN END
140 GOTO 120
```

In Zeile 30 generiert der Rechner in zufälliger Reihenfolge drei Buchstaben, die entweder "A" oder "B" lauten. Ihre Aufgabe ist es, diesen Geheim-Code durch eine Eingabe zu knacken. Gelingt es, wird Ihnen "Zutritt gestattet". Im anderen Fall haben Sie verloren. Zeile 120 fragt die START-Taste ab (Register 53279); in Zeile 130 wird das Register für den Tastatur-Code (35 = Buchstabe N) abgefragt.

1.5 Die Variablen-Tabelle

Ihr ATARI legt bereits während des Programmierens intern eine Tabelle an, die alle von Ihnen definierten Variablen-Namen enthält. Die Anfangs-Adresse dieser Tabelle ist flexibel und hängt von der Länge Ihres Programms ab.

Zur Ermittlung dieser Anfangs-Adresse werden die Speicherstellen 130 und 131 benutzt. In diesen beiden Registern sind LO-Byte- und HI-Byte-Werte für die Anfangs-Adresse der Variablen-Tabelle enthalten. Fachmännischer ausgedrückt: Diese beiden Speicherstellen dienen als "Zeiger" auf die Variablen-Tabelle.

Durch PEEKen der Dezimalwerte aus den einzelnen Tabellen-Adressen können Sie sich die Variablen-Namen "ansehen". Das allein ist freilich langweilig. Interessant wird es, wenn Sie in Ihrem Programm einen Variablen-Namen ändern wollen, ohne dafür das ganze Listing zu durchstöbern. Es genügt, die gewünschte Änderung durch einen einzigen POKE vorzunehmen. Im BASIC-Programm wird dann der geänderte Name automatisch an allen Stellen eingefügt, an denen der ursprüngliche Name stand.

Mit der folgenden 3-Zeilen-Utility (32740 bis 32742) können Sie sich die Variablen-Namen des kurzen Demonstrations-Programms (5 bis 35) ansehen und nach Belieben verändern.

```
5 REM DEMO-PROGRAMM
10 DIM A$(5),B$(5)
20 A$="AAAAA":B$="BBBBB"
30 C=123:D=456
35 REM
32740 CLR :X=PEEK(130)+PEEK(131)*256
32741 Y=PEEK(X):? "POKE ";X;",";Y,:IF Y>128 THEN Y=Y-
128
32742 ? CHR$(Y):X=X+1:IF Y<>0 THEN GOTO 32741
```

Nach RUN (RETURN) sehen Sie:

```
POKE 7676,65        A
POKE 7677,164       $
POKE 7678,66        B
POKE 7679,164       $
POKE 7680,195       C
POKE 7681,196       D
POKE 7682,216       X
POKE 7683,217       Y
POKE 7684,0
POKE 7685,0
```

32740: Durch die Formel LO+HI*256 erhalten Sie die Anfangs-Adresse der Variablen-Tabelle. In diesem Fall ist das die Adresse 7676. In ihr ist der ASCII-Wert 65 entsprechend dem A von A$.

32741: Die Bequemlichkeit dieses kleinen Programms besteht darin, daß auf dem Bildschirm das Wort POKE, die ermittelte Adresse mit dem darin befindlichen Dezimalwert und das entsprechende CHR$-Zeichen erscheint, das den Namen oder die Art der Variable (z.B. String-Variable) kennzeichnet. Falls Sie nun einen Variablen-Namen ändern wollen, fahren Sie einfach mit dem Cursor in die betreffende Zeile, ändern den Dezimalwert nach Ihren Wünschen, löschen das CHR$-Zeichen und drücken RETURN. Der Variablen-Name ist damit geändert, wie Sie nach LIST (RETURN) feststellen werden. Die Bedingung IF Y>128 THEN Y=Y-128 hat folgende Bedeutung: Bei numerischen Variablen wird der erste Buchstabe invertiert und die folgenden in normalen Zeichen ausgegeben (sofern der Variablen-Name länger als ein Buchstabe ist). Bei String-Variablen wird der String-Name mit normalen Zeichen ausgegeben, das Zeichen $ hingegen invertiert. Damit Sie ein klares Bild der Variablen-Tabelle erhalten, werden durch die Anweisung -128 alle invertierten Zeichen in normale umgewandelt.

32742: Nachdem die Anfangs-Adresse der Variablen-Tabelle festliegt, wird sie durch X=X+1 fortlaufend erhöht, damit die gesamte Tabelle auf dem Bildschirm erscheint. Solange der in Zeile 32741 ermittelte Dezimalwert nicht Null ist (Ende der Tabelle), kehrt der Rechner zu dieser Zeile zurück und setzt seine Arbeit fort. Im anderen Fall ist das Programm beendet.

Falls Sie dieses Programm bereits abgetippt und ausprobiert haben, ist Ihnen sicher gleich ein Problem aufgefallen: Die im Demo-Programm aufgeführten Variablen-Namen bestehen alle aus nur einem Buchstaben. Diesen zu ändern ist einfach. Was aber tun, wenn man einen ursprünglich langen Namen abkürzen will? - Selbstverständlich gibt es auch hierfür eine Lösung: Bei String-Variablen müssen Sie in alle überflüssigen Register eine 0 POKEn; bei numerischen Variablen eine 128 (wenn Sie auch bei überflüssigen Zeichen von numerischen Variablen eine 0 in die betreffenden Register POKEn, bekommen Sie ein heilloses Durcheinander).

Das folgende Listing dient als Demonstrations-Beispiel zur Änderung von Variablen-Namen, die aus ursprünglich zwei Zeichen bestanden.

```
5 REM DEMO-PROGRAMM
10 DIM AA$(5),B$(5)
20 AA$="AAAAA":B$="BBBBB"
30 CC=123:D=456
35 REM
32740 CLR :X=PEEK(130)+PEEK(131)*256
32741 Y=PEEK(X):? "POKE ";X;",";Y,:IF Y>128 THEN Y=Y-
128
32742 ? CHR$(Y):X=X+1:GOTO 32741
```

Nach RUN (RETURN) erscheint auf dem Bildschirm:

```
POKE 7676,65        A
POKE 7677,65        A
POKE 7678,164       $
POKE 7679,66        B
POKE 7680,164       $
POKE 7681,67        C
POKE 7682,195       C
POKE 7683,196       D
POKE 7684,216       X
POKE 7685,217       Y
POKE 7686,0
POKE 7687,0
```

Sie müssen das Programm mit BREAK abbrechen, da in diesem Fall über das Ende der Variablen-Tabelle hinaus Register aufgelistet werden.

Wie Sie sehen, existieren in der Tabelle zwei Variablen mit je zwei Buchstaben (AA$ und CC). Fahren Sie nun mit dem Cursor hoch und ändern Sie:

```
POKE 7677,0      (RETURN)
POKE 7682,128    (RETURN)
```

Die Buchstaben, die sich in den jeweiligen Zeilen befinden, müssen Sie löschen. Wenn Sie nun ein LIST eingeben, sehen Sie, daß die geänderten Variablen-Namen in das Programm aufgenommen wurden.

Zur praktischen Anwendung dieser kurzen Utility ist es ratsam, die Zeilen 32740 bis 32742 erst zu ENTERn, wenn sich das zu bearbeitende Programm bereits im Speicher befindet. Wie Sie oben nach RUN gesehen haben, werden auch die Variablen-Namen des Utility-Programms mit ausgedruckt. Es ist daher übersichtlicher, wenn diese Namen am Schluß der Tabelle stehen.

Weiter ist zu beachten, daß in der Tabelle auch Variablen-Namen gespeichert bleiben, die aus alten Programmen stam-

men bzw. die Sie während des Programmierens einmal benutzt und später wieder gelöscht haben. Um unnötiges Durcheinander zu vermeiden, geben Sie einfach NEW (RETURN) ein, bevor Sie das zu bearbeitende Programm laden. Durch NEW wird die Variablen-Tabelle gelöscht.

Falls Sie sich durch die teilweise invertierte Zeichendarstellung nicht irritieren lassen, können Sie das Programm sogar in nur zwei Zeilen unterbringen. Das Listing sieht dann folgendermaßen aus:

```
32740 CLR :X=PEEK(130)+PEEK(131)*256
32741 Y=PEEK(X):? "POKE ";X;",";Y,CHR$(Y):X=X+1:IF Y<
>0 THEN GOTO 32741
```

1.6 Und noch ein Trick

Im Abschnitt 1.1.1 wurde das Verketten von Strings behandelt. Die prinzipielle Anweisung hierzu lautet:

```
A$(LEN(A$)+1)=B$
```

Diese Methode muß eingesetzt werden, wenn die Zeichenlänge von A$ nicht bekannt ist. Falls Sie jedoch wissen, aus wievielen Zeichen die String-Variable besteht, an die Sie einen anderen String anhängen wollen, können Sie sich das Verketten nach folgendem Prinzip erleichtern:

```
10 DIM A$(10),B$(5)
20 A$="TEXT1":B$="TEXT2"
30 ? A$,B$
40 A$(6)=B$
50 ? A$
```

Nach RUN (RETURN) sehen Sie:

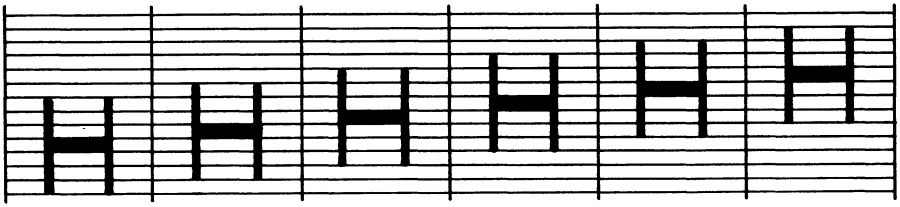
```
TEXT1      TEXT2
TEXT1TEXT2
```

A$ und B$ bestehen hier aus jeweils fünf Zeichen. Durch die Verkettung wird A$ zehn Zeichen lang und muß entsprechend DIMensioniert werden. In Zeile 40 wird dann bestimmt, daß das sechste Zeichen von A$ = B$ sein soll. Hierdurch kommt die Verkettung zustande, ohne daß die Länge von A$ mit LEN ermittelt werden muß.

1.7 Scrolling

Über das Register 622 können Sie bei Ihrem ATARI-Computer (nur die neuen Modelle) ohne große Mühe ein Fein-Scrolling erreichen; d.h. der Bildschirm-Inhalt rollt nicht Printzeile für Printzeile (Grob-Scrolling) vorüber, sondern Bildschirmzeile für Bildschirmzeile.

Dieser Effekt ist sehr angenehm. Sie können z.B. eine Spielanleitung auf diese Weise über den Bildschirm rollen lassen oder auch Programm-Menüs. Folgende Skizze verdeutlicht das Prinzip des Fein-Scrolling.



Um ein Fein-Scrolling zu programmieren, müssen Sie in Register 622 eine 255 ablegen und einen Datenkanal zum Bildschirm-Editor eröffnen. Das folgende Muster-Programm können Sie in Ihre eigenen Listings übernehmen, wobei das Betätigen der HELP-Taste zum Unterbrechen des Programms natürlich fortfallen kann.

```
10 POKE 622,255
20 POKE 732,0
30 OPEN #1,12,0,"E:"
40 ? "Dies ist ein Text, der demonstriert,"
50 ? "wie Fein-Scrolling aussehen kann."
60 ? "Druecken Sie HELP, um das Programm"
70 ? "abzubrechen."
80 ?
90 IF PEEK(732)<>17 THEN GOTO 40
100 POKE 622,0
110 CLOSE #1
```

2 Sound

Mit vier unabhängigen Tongeneratoren (Audiofrequenz-Registern) und einem Tonumfang von 3 1/2 Oktaven (BASIC) bzw. 8 Oktaven (Direktzugriff) bieten die ATARI-Computer Töne in Hülle und Fülle.

Hinzu kommen sechs verschiedene Verzerrungs-Grade, die im SOUND-Statement angewählt werden können. Insgesamt lassen sich damit rund 1,2 Millionen verschiedene Effekte wie Explosionen, Schnarren, Fauchen, Brummen usw. erzeugen.

In diesem Kapitel gehen wir nicht auf alle diese verschiedenen Sound-Möglichkeiten ein; das würde den Rahmen dieses Buches sprengen. Vielmehr zeigen wir Ihnen einige nützliche Programmier-Tricks für bestimmte Sounds, wie sie z.B. in Spielen vorkommen. Außerdem finden Sie Hinweise zur Programmierung von Rhythmus-Effekten durch Verwendung der internen ATARI-Timer.

Aus verschiedenen Noten-Tabellen können Sie ersehen, wie man Musikstücke übertragen oder die Tasten des ATATI mit Orgel-Tönen belegen kann. Wer sich darüber hinaus intensiv mit der Programmierung von Sound-Effekten und Musik beschäftigen möchte, dem sei das "ATARI Sound- und Musik-Buch" (Birkhäuser Verlag) empfohlen, das sich auf 130 Seiten ausführlich diesem Thema widmet. Sie finden darin sowohl eine leicht verständliche Einführung in die Sound- die Musik-Programmierung als auch raffinierte Programmier-Tricks für Fortgeschrittene. Außerdem bietet das Buch erstmals vollständige Tabellen aller Sound-Effekte in den verschiedenen Verzerrungs-Graden.

Sound-Effekte gehören heute zum guten Ton eines jeden Programmierers. Und dies nicht nur in Computer-Spielen. Auch Anwender-Programme und Utilities erlangen erst durch eine sachgerechte Vertonung ihre Professionalität (z.B. Signal-Sound statt ERROR-Meldung bei Fehlbedienung). In diesem Kapitel finden Sie einige Anregungen hierzu.

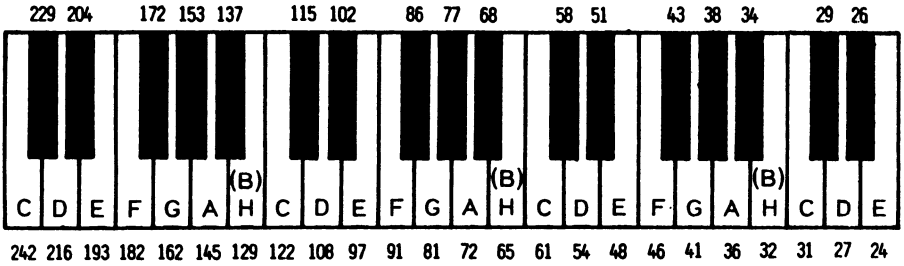
2.1 Musik und Verzerrung

Falls Sie schon einmal einfache Töne spielen oder Musiknoten übertragen wollten und dazu die Notenwerte aus der Bedienungsanleitung des ATARI zu Rate zogen, werden Sie festgestellt haben, daß die einzelnen Töne miteinander unsauber klangen. Der Grund dafür ist ganz einfach: Die Tabelle stimmt nicht.

Bei ATARI werden keine Original-Frequenzwerte, sondern in BASIC-Werte umgerechnete Frequenzen eingegeben. Die Formel dazu lautet: BASIC-Wert = 31960 : (Originalfrequenz + 1).

Das mittlere C z.B. schwingt mit einer Original-Frequenz von 261,62 Hz (Hertz). Demnach errechnet sich der BASIC-Wert hierfür: 31960 : (261,62 + 1) = 121,69 (aufgerundet 122). Im Anleitungsbuch steht jedoch für das mittlere C der Wert 126.

Auch andere Tonwerte sind falsch angegeben. Die folgende Abbildung zeigt Ihnen eine harmonisch aufeinander abgestimmte Frequenz-Tabelle.



Natürlich könnte man noch weitere Töne anhängen (Werte bis Null). Aber ab dem BASIC-Frequenzwert 30 klingen die Noten wie der Mitternachtsgesang einer Grille mit Grippe.

Außer reinen Tönen lassen sich beim ATARI noch sechs verschiedene Verzerrungen einstellen, die ganz unterschiedliche Effekte bringen und besonders für die Spiele-Programmierung interessant sind.

Bei der Programmierung von Sound-Effekten kann es hin und wieder vorkommen, daß gar nichts ertönt. Das hängt mit der Funktionsweise der Audiofrequenz-Register zusammen und hat nichts damit zu tun, daß Sie falsch programmiert haben.

Wählen Sie in solchen Fällen einfach eine andere Frequenz (z.B. 246 statt 247). Meistens wird dadurch das Problem bereits gelöst.

Die folgende Tabelle zeigt die Sound-Charaktere der einzelnen Verzerrungs-Grade.

Verzerrung 0 Anwendung:	unrhythmisches, knurrendes Rauschen Explosion, Schuß, Rauschgeräusche
Verzerrung 2 Anwendung:	verzerrtes, aber rhythmisch-periodisches Rauschen Begleitsound für Action, Bewegung
Verzerrung 4 Anwendung:	verzerrtes, knatterndes Schlagen Motor, utopische Flug- und Fahrgeräusche
Verzerrung 6 Anwendung:	rhythmisches Surren Maschinengeräusche, Roboterschritte, Rennwagen
Verzerrung 8 Anwendung:	unrhythmisches, fauchendes Rauschen Explosion, Schuß, Drachen, Fauchgeräusche
Verzerrung 12 Anwendung:	vom verzerrten Brummen bis zum melodischen Summen vielseitig; für Action, Flipper, Elektronik-Geräusche
Die Angaben 10 und 14 bringen unverzerrte Töne	

Hier einige Anwendungs-Beispiele für Sound-Effekte wie sie in vielen Spielen oder anderen Programmen eingesetzt werden.

```
0 REM FALLENDES OBJEKT
10 FOR FREQ=15 TO 255
20 SOUND 0,FREQ,10,15
30 NEXT FREQ
```

```
0 REM PUNKTGEWINN
10 FOR FREQ=255 TO 15 STEP -2
20 SOUND 0,FREQ,10,15
30 NEXT FREQ
```

```
0 REM GEFAHR/ALARM
5 FOR X=0 TO 3
10 FOR FREQ=200 TO 15 STEP -2
20 SOUND 0,FREQ,10,15
30 NEXT FREQ
40 NEXT X
```

```
0 REM ELEKTRONIK-SOUND
5 FOR X=0 TO 10
10 FOR FREQ=200 TO 15 STEP -20
20 SOUND 0,FREQ,10,15
30 NEXT FREQ
40 NEXT X
```

```
0 REM LASER-WAFFEN
5 FOR X=0 TO 10
10 FOR FREQ=15 TO 200 STEP 10
20 SOUND 0,FREQ,10,15
30 NEXT FREQ
40 NEXT X
```

```
0 REM AUFDRINGLICHES OBJEKT
5 FOR X=0 TO 10
10 FOR FREQ=200 TO 255
20 SOUND 0,FREQ,10,15
30 NEXT FREQ
40 NEXT X
```

```
0 REM SCHUSS
1 REM AUSLOESEN MIT 'RETURN'
2 REM UNTERBRECHEN MIT 'RREAK'
10 DIM A$(1)
20 LAUT=15
30 PRINT "SCHUSS ";
40 INPUT A$
50 FOR F=0 TO 29 STEP 0.5
60 SOUND 0,F,0,LAUT
70 LAUT=LAUT-0.2
80 NEXT F
90 SOUND 0,0,0,0
100 GOTO 20
```

```
0 REM TYPISCHER VIDEOSPIEL-SOUND
1 REM PUNKTVERLUST/FEHLSCHLAG
2 REM NIEDERLAGE/REGELWIDRIGKEIT
10 FOR FREQ=5 TO 14
20 SOUND 0,FREQ,2,8
30 FOR ZEIT=0 TO 50
40 NEXT ZEIT
50 NEXT FREQ
```

```
0 REM SCHUSS
10 FOR F=0 TO 59
20 SOUND 0,F,8,10
30 NEXT F


0 REM FAUCHEN
10 FOR LAUT=-10 TO 15
20 SOUND 0,149,8,15-ABS(LAUT)
30 SOUND 1,120,8,15-ABS(LAUT)
40 SOUND 2,135,8,15-ABS(LAUT)
50 FOR ZEIT=0 TO 5:NEXT ZEIT
60 NEXT LAUT


0 REM FLIPPER-SOUND
10 LAUT=15
20 FOR FREQ=0 TO 50
30 SOUND 0,FREQ,12,LAUT
40 FOR ZEIT=0 TO 20:NEXT ZEIT
50 LAUT=LAUT-0.25
60 NEXT FREQ
```

2.1.1 Timer

Die ATARI-Computer verfügen über mehrere sog. Timer, die als Taktgeber programmiert und eingesetzt werden können. Das ist besonders für Sound- und Rhythmus-Effekte interessant, da sich die Tongeneratoren relativ unabhängig vom sonstigen Programmablauf einschalten bzw. ausschalten lassen.

Jedes der fünf verfügbaren Timer-Register (es gibt darüber hinaus noch Timer mit anderer Funktionsweise, die hier nicht interessant sind) ist in LO-Byte-Timer und HI-Byte-Timer unterteilt. Sie zählen grundsätzlich von einem eingegebenen Wert bis 0; d.h. wenn im HI-Byte-Timer eine 5 steht, taktet das LO-Byte fünfmal von 255 bis 0, wobei sich mit jedem Zyklus das HI-Byte um 1 reduziert. Sie können sich diese Funktionsweise mit folgendem Programm veranschaulichen:

```
10 POKE 537,5
20 ? PEEK(536),PEEK(537)
30 GOTO 20
```

Damit kennen Sie auch schon die Adressen des ersten Timers. Folgende Tabelle zeigt die Adressen aller fünf Timer.

LO-Byte-Register		HI-Byte-Register
536	Timer 1	537
538	Timer 2	539
540	Timer 3	541
542	Timer 4	543
544	Timer 5	545

Es ist übrigens nicht unbedingt notwendig, immer LO-Byte und HI-Byte zusammen einzusetzen. Wenn Sie z.B. in Ihrem BASIC-Programm irgendeine Aufgabe schnell takten wollen, benutzen Sie am besten nur den LO-Byte-Timer; soll hingegen ein relativ langsamer Takt wirksam werden, ist der HI-Byte-Timer geeinget.

Versuchen Sie folgendes Demonstrations-Programm, in dem nur das LO-Byte programmiert wird:

```
10 POKE 536,255
20 A=PEEK(536)
30 IF A=255 THEN ? "Register 536 = 255":SOUND 0,122,1
0,8
40 IF A=200 THEN ? "Register 536 = 200":SOUND 0,97,10
,8
50 IF A=150 THEN ? "Register 536 = 150":SOUND 0,81,10
,8
60 IF A=100 THEN ? "Register 536 = 100":SOUND 0,68,10
,8
70 IF A=50 THEN ? "Register 536 = 50":SOUND 0,61,10,8
80 IF A=0 THEN ? "Register 536 = 0":SOUND 0,0,0,0:END
90 GOTO 20
```

Der Einsatz des HI-Byte (also ein langsamer Takt) ist z.B. nützlich, wenn in einem Spiel eine Aufgabe in einem bestimmten Zeitraum bewältigt werden soll. Sobald die Zeit abgelaufen ist (das Timer-Register auf 0 steht), kann eine Melodie für Sieg oder Niederlage ertönen; es können Punkte hinzugerechnet oder abgezogen werden usw.

Ein schneller Takt (das LO-Byte benötigt etwa 2 1/2 Sekunden von 255 bis 0) ist nützlich für Sound-Effekte oder Actionabläufe in Spielen, die sich auf diese Weise takten lassen. Die Einsatzmöglichkeiten der Timer sind praktisch unbegrenzt und hängen ganz von den Aufgaben in Ihren Programmen ab. Bei den nachfolgenden Anwendungs-Beispielen wird im ersten Programm ein schneller Musik-Rhythmus erzeugt und im zweiten Programm eine akustische Zeitmessung durchgeführt.

```
0 REM RHYTHMUS.001
10 A=536
20 POKE A,255
30 B=PEEK(A)
40 IF B>=245 THEN SOUND 0,200,12,10:GOTO 30
50 SOUND 0,10,0,10:FOR Z=0 TO 25:NEXT Z
60 SOUND 0,0,0,0:FOR Z=0 TO 25:NEXT Z
70 SOUND 0,10,0,10:FOR Z=0 TO 25:NEXT Z
80 SOUND 0,0,0,0:FOR Z=0 TO 25:NEXT Z
90 GOTO 20
```

```
0 REM TIMER.001
10 A=536
20 POKE A,255
30 B=PEEK(A)
40 IF B>=100 THEN SOUND 0,10,10,10
50 IF B<100 THEN GOTO 100
60 FOR Z=0 TO 50:NEXT Z
70 SOUND 0,0,0,0
80 FOR Z=0 TO 50:NEXT Z
90 GOTO 30
100 SOUND 0,10,10,10
110 FOR Z=0 TO 500:NEXT Z
120 SOUND 0,0,0,0
```

2.2 Das Audiokontroll-Register 53768 (AUDCTL)

Neben den einfachen Klang-Möglichkeiten und Geräusch-Effekten, die durch das SOUND-Statement programmiert werden, verfügen die ATARI-Computer über ein weiteres Klang-Spektrum, das allerdings nur durch spezielle POKE-Anweisungen zur Verfügung steht. Diese neuen Sounds werden für alle vier Tongeneratoren über das Register 53768 (AUDCTL) gesteuert.

Die folgende Tabelle zeigt, mit welchen Funktionen die einzelnen Bits dieses Registers belegt sind.

AUDCTL 53768								
Bit	7	6	5	4	3	2	1	0
Dezimal	128	64	32	16	8	4	2	1
	Zusätzlicher Verzerrungsgrad; Anwählen mit SOUND x,x,8,x	1,79 MHz Eingabetakt auf AUDF0 (normal 64 KHz)	1,79 MHz Eingabetakt auf AUDF2 (normal 64 KHz)	16-Bit Frequenz-Auswahl auf AUDF1 und AUDF0	16-Bit Frequenz-Auswahl auf AUDF3 und AUDF2	Hochpaßfilter auf AUDF0, gesteuert von AUDF1	Hochpaßfilter auf AUDF3, gesteuert von AUDF2	15 KHz Eingabetakt (normal 64 KHz)

2.2.1 Neues Frequenz-Spektrum (AUDCTL, Bit 0)

Jeder Computer arbeitet mit einem bestimmten Systemtakt, durch den die vielen einzelnen Arbeitsleistungen intern synchronisiert werden. Beim ATARI ist dies 1,79 MHz. Außer diesem Haupttaktgeber gibt es noch mehrere Nebentaktgeber für spezielle Funktionen. Auch die Tongeneratoren werden durch einen Nebentaktgeber gesteuert; und zwar normalerweise mit 64 KHz.

Wenn Sie in einer SOUND-Anweisung z.B. die Frequenz 4 aufrufen, wird aus dem 64 KHz-Takt jeder vierte Puls herausgefiltert und zur Ausgabe gebracht. Auf diese Weise können die verschiedenen Notenwerte erzeugt werden. Sobald jedoch Bit 0 im AUDCTL gesetzt wird, reduziert sich der 64 KHz-Takt auf 15 KHz. Wenn Sie nun die Frequenz 4 aufrufen, wird ebenfalls jeder vierte Puls herausgefiltert; allerdings bringt

15.000 : 4 einen wesentlich tieferen Klang als 64.000 : 4. Versuchen Sie dazu bitte folgendes Programm-Beispiel:

```
10 ? CHR$(125)
20 SOUND 0,40,10,10
30 ? "Tonanweisung mit 64 KHz getaktet":?
40 FOR Z=0 TO 500:NEXT Z
50 POKE 53768,1
60 ? "Die gleiche Tonanweisung mit 15 KHz   getaktet"
:?
70 FOR Z=0 TO 500:NEXT Z
```

10: Löscht den Bildschirm.

20: Der BASIC-Frequenzwert 40 wird unverzerrt in Lautstärke 10 aufgerufen.

40: Eine Verzögerungs-Schleife

50: Der Eingabetakt für die Audiofrequenz-Register wird von 64 KHz auf 15 KHz abgesenkt.

Die Anweisung im AUDCTL gilt für alle Tongeneratoren. Sie können das neue Klang-Spektrum z.B. für Sound-Effekte einsetzen, die sich mit 64 KHz-getakteten Tongeneratoren nicht erzeugen lassen.

2.2.2 Hochpaßfilter (AUDCTL, Bit 1, Bit 2)

Mit einem Hochpaßfilter lassen sich aus einer Eingabefrequenz die hochtonigen Schwingungen herausfiltern, während die tiefen Töne unterdrückt werden.

Auch beim ATARI-Computer können Sie einen Hochpaßfilter programmieren. Die Effekte, die dadurch entstehen, lassen sich besonders gut in Spielen einsetzen. Sie zeichnen sich in der Regel durch einen schrillen, schnarrenden Klang aus, was z.B. als Action-Untermalung interessant ist.

Durch Setzen von Bit 1 im AUDCTL erreichen Sie, daß der Tonkanal 1 nur Töne passieren läßt, die über den Frequenzwerten aus Tonkanal 3 liegen. Das Setzen von Bit 2 hingegen läßt auf Tonkanal 0 nur Töne passieren, die über den Frequenzwerten aus Tonkanal 2 liegen.

Leider ist es nicht möglich, einen Hochpaßfilter zusammen mit normalen SOUND-Anweisungen zu programmieren. Sie müssen vielmehr die einzelnen Audiofrequenz-Register und die dazugehörigen Kontroll-Register (für Lautstärke und Verzerrung) direkt ansprechen.

Die folgende Tabelle zeigt die Bit-Belegung und Funktionen der Audiofrequenz- und Audiosteuer-Register.

Audiofrequenz-Register
AUDF0 53760 (entspricht SOUND 0,x,.,.)
AUDF1 53762 (entspricht SOUND 1,x,.,.)
AUDF2 53764 (entspricht SOUND 2,x,.,.)
AUDF3 53766 (entspricht SOUND 3,x,.,.)
Dezimalwerte 0 bis 255 (BASIC-Frequenz-Wahl)

Audiosteuer-Register								
AUDC0 53761			AUDC1 53763			AUDC2 53765		AUDC3 53767
Bit	7	6	5	4	3	2	1	0
Dezimal	128	64	32	16	8	4	2	1
Verzerrung 0	-	-	-	Direkte Steuerung der Lautsprecher-Membran	Lautstärke			
Verzerrung 2	-	-	x					
Verzerrung 4	-	x	-					
Verzerrung 6	-	x	x					
Verzerrung 8	x	-	-					
Verzerrung 10	x	-	x					
Verzerrung 12	x	x	-					
Verzerrung 14	x	x	x					

Soll z.B. ein Hochpaßfilter über AUDF1 für die Töne aus AUDF3 aktiviert werden, müssen Sie folgendes Programm schreiben:

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,2:REM Hochpassfilter auf AUDF1
30 POKE 53762,100:REM AUDF1
40 POKE 53763,47:REM AUDC1
50 POKE 53766,200:REM AUDF3
60 FOR Z=0 TO 1000:NEXT Z
```

Exakt das Gleiche können Sie auch erreichen, wenn Sie Bit 2 im AUDCTL setzen und einen Hochpaßfilter auf AUDF0 für die Töne aus AUDF2 erhalten wollen. In diesem Fall müssen Sie das Listing folgendermaßen ändern:

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,4:REM Hochpassfilter auf AUDF0
30 POKE 53760,100:REM AUDF0
```

```
40 POKE 53761,47:REM AUDC0
50 POKE 53764,200:REM AUDF2
60 FOR Z=0 TO 1000:NEXT Z
```

Sie können mit diesen Beispielen stundenlang experimentieren: Ändern Sie die Frequenz-Angaben in den verschiedensten Kombinationen und setzen Sie andere Verzerrungsgrade ein (s. Tabelle). Natürlich ist es auch möglich, beide Hochpaßfilter gleichzeitig zu aktivieren. Fügen Sie einfach die beiden zuvor gezeigten Listings zusammen (ändern Sie aber eine der Frequenz-Angaben, da sonst beide Sounds identisch klingen).

Versuchen Sie auch einmal diese Kombination:

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,6:REM Hochpassfilter auf AUDF1/AUDF0
30 POKE 53762,150:REM AUDF1
35 POKE 53760,5:REM AUDF0
40 POKE 53763,47:REM AUDC1
45 POKE 53761,74:REM AUDC0
50 POKE 53766,100:REM AUDF3
55 POKE 53764,240:REM AUDF2
60 FOR Z=0 TO 1000:NEXT Z
```

2.2.3 16-Bit Frequenz-Wahl (AUDCTL, Bit 3, Bit 4)

In normalen SOUND-Anweisungen können Sie Frequenz-Angaben von 0 bis 255 machen. Da sich die hohen Töne nur begrenzt einsetzen lassen (vergl. Klaviatur S. 29), stehen für die Musik-Anwendung etwa 3 1/2 Oktaven zur Verfügung.

Durch Setzen von Bit 3 im AUDCTL werden die Audiofrequenz-Register 2 und 3 verbunden, wodurch eine 16-Bit Frequenz-Wahl möglich wird (LO-Byte/HI-Byte). Es steht damit ein Tonumfang von etwa 8 Oktaven zur Verfügung. Durch Setzen von Bit 4 im AUDCTL werden AUDF0 und AUDF1 verbunden.

Sie können bei Ihrer Musik-Programmierung also wählen zwischen vier Tongeneratoren mit je 3 1/2 Oktaven Tonumfang oder zwei Tongeneratoren mit je acht Oktaven Tonumfang. Darüber hinaus ist auch die Kombination aus 16-Bit Frequenz-Wahl und 8-Bit Frequenz-Wahl möglich; Ihnen stehen dann z.B. AUDF2/AUDF3 als Bass-Stimme sowie AUDF0 und AUDF1 als Oberstimmen zur Verfügung.

Mit dem folgenden Programm-Beispiel werden die Audiofrequenz-Register 2 und 3 durch Setzen von Bit 3 verbunden und ein tiefes C gespielt (vergl. Frequenz-Tabelle mit LO-Byte und HI-Byte-Angaben nächste Seite).

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,8:REM 16-Bit-Frequenzwahl
30 POKE 53764,202:REM AUDF2 (LO)
40 POKE 53766,3:REM AUDC3 (HI)
50 POKE 53767,170:REM AUDC3
60 FOR Z=0 TO 1000:NEXT Z
```

Wenn Sie stattdessen AUDF0 und AUDF1 für eine 16-Bit Frequenz-Wahl verbinden wollen, muß das Listing folgendermaßen geändert werden:

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,16:REM 16-Bit-Frequenzwahl
30 POKE 53760,202:REM AUDF0 (LO)
40 POKE 53762,3:REM AUDC1 (HI)
50 POKE 53763,170:REM AUDC1
60 FOR Z=0 TO 1000:NEXT Z
```

Auf der folgenden Tabelle finden Sie die LO-Byte und HI-Byte-Werte für die Musik-Noten, die Sie mit normalem BASIC nicht erreichen können.

242 = Note C (letzte reine Note in BASIC)

Frequenz	Note	LO-Byte	HI-Byte	Frequenz	Note	LO-Byte	HI-Byte
257	H	1	1	727	F	215	2
272	Ais	16	1	770	E	2	3
288	A	32	1	816	Dis	48	3
305	Gis	49	1	864	D	96	3
324	G	68	1	916	Cis	148	3
343	Fis	87	1	970	C	202	3
363	F	107	1				
385	E	129	1				
408	Dis	152	1				
432	D	176	1				
458	Cis	202	1				
485	C	229	1				
514	H	2	2				
544	Ais	32	2				
577	A	65	2				
611	Gis	99	2				
647	G	135	2				
686	Fis	174	2				

2.2.4 Eingabetakt mit 1,79 MHz (AUDCTL, Bit 5, Bit 6)

So wie durch Setzen von Bit 0 im AUDCTL der Eingabetakt für die Audirofrequenz-Register von 64 KHz auf 15 KHz herabgesetzt werden kann, so ist es mit Bit 5 möglich, den Eingabetakt auf 1,79 MHz zu erhöhen. Dies gilt für die Frequenzen aus AUDF2. Wird hingegen Bit 6 im AUDCTL gesetzt, erhöht sich der Eingabetakt von 64 KHz auf 1,79 MHz für die Frequenzen aus AUDF0.

Folgendes Listing zeigt die Wirkung dieser Manipulation:

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,32:REM Eingabetakt mit 1,79 MHz
30 POKE 53765,170:REM AUDC2
40 POKE 53764,FRQ:REM AUDF2
50 FRQ=FRQ+1
60 IF FRQ=255 THEN END
70 GOTO 40
```

Das gleiche Programm, wenn nicht auf AUDF2, sondern auf AUDF0 1,79 MHz getaktet werden soll:

```
10 SOUND 0,0,0,0:REM Null-Tonanweisung
20 POKE 53768,64:REM Eingabetakt mit 1,79 MHz
30 POKE 53761,170:REM AUDC0
40 POKE 53760,FRQ:REM AUDF0
50 FRQ=FRQ+1
60 IF FRQ=255 THEN END
70 GOTO 40
```

Nach RUN (RETURN) merken Sie, daß ein Teil der Töne jenseits des hörbaren Bereichs liegen. Der Einsatz des 1,79 MHz-Taktes läßt sich in dieser Form also nur für Sound-Effekte einsetzen.

Interessant wird es, wenn Sie zusätzlich noch eine 16-Bit Frequenz-Wahl programmieren. Sie erhalten dadurch die Möglichkeit, hohe Notenwerte wesentlich feiner abzustimmen als mit normalem BASIC. Selbstverständlich können Sie auch alle Frequenzen in den verschiedenen Verzerrungen abspielen lassen. Sie erschließen sich damit ein völlig neues Reservoir an Sound-Effekten. Doch bevor Sie sich ans Experimentieren machen, sollten Sie sich ein paar Tage Urlaub nehmen. Sie können nämlich mit der Direkt-Programmierung der Audiofrequenz-Register über 1,2 Millionen verschiedene Sounds erzeugen.

Noch ein Tip: Notieren Sie sich die POKE-Werte von gelungenen Sound-Effekten, oder speichern Sie sich diese gleich ab.

2.2.5 Neuer Verzerrungs-Grad (AUDCTL, Bit 7)

Es wurde bereits erklärt, daß zur Erzeugung verschiedener Töne einzelne Frequenzen aus dem Eingabetakt der Audiofrequenz-Register (normal 64 KHz) herausgefiltert werden. Man spricht hier von einer Frequenz-Division (obwohl es sich um keine wirkliche Division handelt).

Nach der Frequenz-Division gelangen die Pulse jedoch nicht gleich zum Lautsprecher; vielmehr werden sie einem Poly-Counter (Mehrfachzähler) zugeführt, um sich weiterbearbeiten zu lassen. Jeder Tonkanal verfügt über drei verschiedene Poly-Counter, die je nach gewählter Verzerrung unterschiedlich arbeiten. Die genaue Funktionsweise dieser Poly-Counter kann hier nicht erklärt werden (eine ausführliche Beschreibung finden Sie im "ATARI Sound- und Musik-Buch"). Das Geheimnis von Bit 7 des AUDCTL besteht darin, daß einer dieser Poly-Counter verändert wird und dadurch ein völlig neuer Verzerrungs-Grad entsteht, der mit normalen SOUND-Anweisungen aufgerufen werden kann.

Versuchen Sie dazu bitte folgendes Demonstrations-Programm:

```
10 SOUND 0,5,8,15
20 FOR Z=0 TO 500:NEXT Z
30 POKE 53768,128
40 FOR Z=0 TO 500:NEXT Z
```

Zunächst wird hier die BASIC-Frequenz 5 in der Verzerrung 8 gespielt (Zeile 10) und danach Bit 7 im AUDCTL gesetzt (Zeile 30). Der Sound hört sich nun viel rhythmischer an.

Sie können auf diese Weise sämtliche Frequenzen spielen, die normalerweise mit Verzerrung 8 gewählt werden. Darüber hinaus ist es auch möglich, diesen neuen Verzerrungs-Grad mit den anderen hier vorgestellten Sound-Manipulationen zu kombinieren (z.B. 16-Bit Frequenz-Wahl oder Hochpaßfilter).

3 Grafik

Um mit dem Computer kommunizieren zu können, wird ein Peripherie-Gerät benötigt. Das ist heute durchweg der Bildschirm, der einen wesentlich schnelleren Austausch ermöglicht als z.B. ein Drucker. Die Darstellung der Informationen auf der Mattscheibe verlangt vom Rechner jedoch einen erheblich größeren Arbeitsaufwand, denn er muß nicht nur die kompletten Daten für die bildhafte Darstellung ermitteln; da der Monitor selbst keinen Speicher hat, muß das Video-Signal auch kontinuierlich vom Computer gesendet werden. Neben der Datenfülle, die ein Video-Bild enthält, ist die Synchronisation zwischen den beiden Geräten eine diffizile Angelegenheit.

Im wesentlichen setzt der Rechner das Fernsehbild aus drei Informationsquellen zusammen. Zum einen wird eine Display-Liste verwendet, die den Video-Chip steuert. Diese Display-Liste bestimmt die Aufteilung des Bildschirms und besorgt die Synchronisation mit dem Arbeitstakt der Bildröhre. Als einziger Computer seiner Größenordnung stellt ATARI sechzehn verschiedene Grafik-Betriebsarten bereit, die sich in der Bildauflösung (Anzahl der Schreibstellen in horizontaler und vertikaler Richtung), Anzahl der gleichzeitig darstellbaren Farbtöne und der Darstellung von Grafikpunkten und/oder Schriftzeichen unterscheiden.

Sollen Schriftzeichen auf dem Bildschirm dargestellt werden, so muß irgendwo im Speicher des Rechners abgelegt sein, wie diese Zeichen auszusehen haben. (Natürlich muß auch eine Routine vorhanden sein, welche die Ausgabe der Zeichen auf dem Bildschirm organisiert und beim Schreiben von Programmen, z.B. im Editier-Modus GRAPHICS 0, den automatischen Zeilenvorschub etc. organisiert, aber davon wird in diesem Buch nicht die Rede sein.)

Die dritte wichtige Informationsquelle ist der Bildschirmspeicher. Hier ist abgelegt, an welcher Stelle des Bildschirms ein Grafikpunkt einer bestimmten Farbe bzw. ein bestimmtes Zeichen aus dem Zeichenvorrat dargestellt werden soll. Abgelegt sind hier lediglich die COLOR-Werte der Grafikpunkte. Welcher Farbton welchem COLOR-Wert zugeordnet ist, bestimmen die Farbregister.

3.1 Display-Liste (DL)

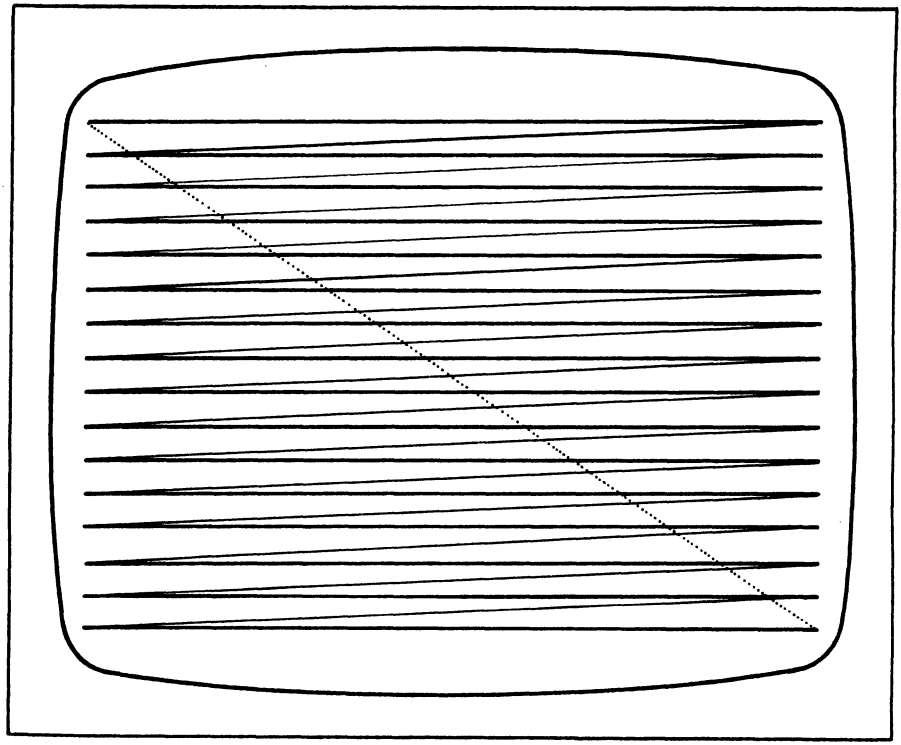
Die Display-Liste ist ein kurzes Maschinensprache-Programm, das den Video-Chip und damit die Signal-Abgabe an den Monitor steuert. Es beginnt mit einem Zeiger (Vektor) auf die Start-Adresse des Bildschirmspeichers, enthält dann Informationen für die Bildauflösung und endet mit einem Rücksprungbefehl an den Anfang der Display-Liste. Dieser Rücksprung wird aber erst ausgeführt, wenn die Bildröhre einen Vertical Blank ausführt, also immer wenn der Kathodenstrahl unten rechts auf der Mattscheibe angekommen ist und nach oben links zurückspringt, um einen neuen Durchlauf zu beginnen. Auf diese Weise wird der Informationsfluß vom Computer mit dem Bildschirm synchronisiert.

Nach jedem Bilddurchlauf ließe sich also theoretisch die DL, der Bildschirmspeicher oder der Inhalt der Farbregister wechseln. Da eine Änderung z.B. des Farbwertes in Bruchteilen von Microsekunden zu erledigen ist, kann man sogar die laufende Bearbeitung der DL unterbrechen, Farbwerte verändern und die DL weiterlaufen lassen. Dieser DLI (display list interrupt) wird während des Horizontal Blank eingelegt, also in dem Moment, wo der Kathodenstrahl vom rechten Rand der Mattscheibe an den Beginn der nächsten Abtastzeile springt.

Änderung der Farbwerte im DLI ist nur in Maschinensprache möglich, weil der Inhalt der Farbregister sonst nur nach dem Vertical Blank gelesen werden. Durch DLI's ist es möglich, in jeder Bildzeile einen anderen Farbwert aufzurufen und auf diese Weise alle ATARI-Farben gleichzeitig zu zeigen:

```
100 DIM M$(24):POKE 710,0:POKE 752,1
110 ? CHR$(125)
120 FOR J=1 TO 24:READ A:M$(J,J)=CHR$(A):NEXT J
130 A=USR(ADR(M$))
140 END
150 DATA 162,0,173,11,212,201,32,208,249,141,10,212,1
42,24,208,232,232,208,246,142,24,208,240,232
```

Dieses kleine BASIC-Programm verarbeitet die Maschinensprache-Routine als String (M$). Die DATA in Zeile 150 enthalten die Anweisung für den DLI und die Änderung des Farbwertes nach jedem Horizontal Blank.



Ein gelenkter Kathodenstrahl tastet die Vorderfläche der Bildröhre fortlaufend ab. Er beginnt in der Ecke oben links und schreibt eine Zeile. Am rechten Rand angelangt, wird er kurz abgeschaltet und springt nach links, an den Anfang der nächsten Zeile. Diese kurze Schreibunterbrechung heißt Horizontal Blank.

Ist der Kathodenstrahl in der rechten unteren Ecke angelangt, wird er für einen etwas längeren Moment ausgeschaltet, um in die Ausgangsstellung nach oben links zurückzukehren. Dieser Vorgang wird Vertical Blank genannt.

Der Kathodenstrahl tastet die Bildfläche in einer Sekunde fünfzig Mal ab. Die Blanks sind also ausgesprochen kurz. Wegen der hohen Arbeitsgeschwindigkeit der Elektronik bieten sie trotzdem genügend Raum, um z.B. das Video-Text-Signal aufzunehmen.

3.1.1 Struktur der DL

Ein Fernsehbild besteht (nach der bei uns gültigen CCIR-Norm) aus 625 Zeilen mit insgesamt etwa einer halben Million Bildpunkten. Da in einem Durchlauf nur jede zweite Zeile abgetastet wird (Halbbild), halbiert sich der Informationsgehalt eines Einzelbildes. Doch selbst wenn jeder dieser 250.000 Punkte nun in einer Farbe leuten oder nicht leuchten können soll, so ergibt das immer noch eine Informationsmenge von ca. 31 kByte und damit ist heute noch jeder Heim-Computer reichlich überfordert. (Die gängigen 64-kByte-Geräte lassen dem Benutzer durchschnittlich 32 kByte RAM übrig und ein bißchen Platz soll ja auch noch für ein Programm bleiben.)

Heim-Computer nutzen deshalb nur einen Teil der Bildröhre. Beim ATARI sind es genau 192 Zeilen, auf denen er 320 Punkte anspricht. Um diese Auflösung mit entweder schwarzen oder weißen Punkten zu füllen, braucht es immerhin noch 7.680 Byte (ca. 7,5 KByte) und das ist rund ein Viertel des verfügbaren Speichers.

Deshalb gibt es neben der HI(gh) RES(olution) noch diverse Bildauflösungen, die weniger Kapazität verschlingen, weil die ansprechbaren Grafikpunkte größer sind. Mehrere auf einer Zeile nebeneinander liegende Bildpunkte werden zusammengefaßt und mehrere Abtastzeilen der Bildröhre werden mit der gleichen Bildinformation versort, so daß eine Grafik-Modus-Zeile entsteht, die zwei, vier oder acht Abtastzeilen hoch ist. Auf diese Weise entstehen Grafikpunkte, die ein, zwei, vier oder acht Bildpunkte breit und ein, zwei, vier oder acht Abtastzeilen hoch sind.

Wie die im Bildschirmspeicher des Computers liegenden Informationen auf dem Monitor umgesetzt werden sollen, ist in der Display-Liste vorprogrammiert. Wird z.B. GRAPHICS 4 aufgerufen, dann beträgt die Auflösung 80 mal 40 (bzw.48) Grafikpunkte. Jedes Pixel ist also vier Bildpunkte breit und vier Bildschirmzeilen hoch. Wenn der Video-Chip nun die Daten aus dem Bildschirmspeicher liest, dann bestimmt jedes Bit den COLOR-Wert für vier Bildpunkte (80*4=320) und die Daten für eine Bildschirmzeile werden viermal hintereinander an die Bildröhre abgegeben (4*48=192).

Da ohnehin nur ein Teil der Bildfläche genutzt wird, beginnt das Computer-Bild nicht am oberen Rand des Bildschirms, sondern 24 Abtastzeilen tiefer. Die DL beginnt deshalb mit der Anweisung, dreimal acht leere Bildschirmzeilen zu schreiben. Es folgt die LMS- (load memory scan) Anwei-

sung ("hole Daten aus Speicher"), der natürlich die Information folgen muß, bei welcher Adresse beginnend diese Daten liegen, im Falle Grafik also die Start-Adresse des Bildschirmspeichers. Um eine Adresse zu erfassen werden zwei Byte benötigt die in der Reihenfolge LO (low Byte), HI (high Byte) abgelegt werden. Adresse = LO+HI*256.

LO (low Byte), HI (high Byte). Die Start-Adresse errechnet sich LO+HI*256.

Es folgen so viele Byte wie Zeilen auf dem Bildschirm entstehen sollen. Jedes Byte bestimmt mit einem speziellen Wert, ob es sich um die Darstellung von Schriftzeichen oder Grafikpunkten handelt, wie viele Farbtöne darstellbar sind, wie viele Schreibpositionen/Grafikpunkte auf einer Zeile liegen und wie oft die Zeile wiederholt werden soll, d.h. wie viele Bildschirmzeilen ein Grafikpunkt hoch ist.

Da das Video-Signal fortlaufend gesendet werden muß, ist die Display-Liste eine endlose Schleife. Sie endet also mit einem Sprungbefehl. Der Sprungbefehl wird aber erst ausgelöst, wenn ein Vertical Blank auftritt, damit Display-Liste und Bildröhre synchron laufen. Nach dem Sprungbefehl folgen wieder zwei Byte mit der Sprung-Adresse (LO, HI) und zwar an den Anfang der Display-List.

Bevor es mit haufenweise Zahlen weitergeht, sehen Sie sich doch erst einmal ein paar Display-Lists an. Mit jedem GR.-Befehl, den Sie programmieren, wird ja eine entsprechende Routine im Speicher abgelegt:

```
0 REM DLPRINT.BI2
10 POKE 82,0:? CHR$(125):?
20 ? " Die Display-Liste welchen Grafikganges"
30 ? :? " wollen Sie sich ansehen?"
40 ? :? " Bitte eine Zahl von 0 bis 31 eingeben:"
50 ? "_____":? :?
60 INPUT G
70 GRAPHICS G:DIM P(201)
80 DL=PEEK(560)+PEEK(561)*256
90 FOR I=0 TO 201
100 P(I)=PEEK(DL+I)
110 NEXT I
200 GRAPHICS 0:POKE 82,0
210 FOR I=0 TO 14:POKE 675+I,136:NEXT I
220 FOR I=0 TO 201
230 IF P(I)=0 THEN IF P(I+1)=0 THEN END
240 ? P(I);CHR$(127);
250 NEXT I
```

70: Hier wird der eingegebene Grafikgang eingeschaltet und damit die entsprechende Display-Liste aufgerufen.

80: Die Start-Adresse der Display-Liste ist in den Adressen 88 (LO) und 89 (HI) abgelegt.

90: Die Daten-Byte aus der Display-Liste werden in der Variablen P(n) gespeichert, denn um die Werte auf den Bildschirm zu bringen, muß GR.0 eingeschaltet werden und damit wird auch die Display-Liste von GR.0 im Speicher abgelegt.

200: Während die Display-Liste gelesen wird, ist der entsprechende Grafik-Modus eingeschaltet. Das dauert einen kleinen Moment. Dann wird der Text-Modus GR.0 eingeschaltet.

210: schreibt TAB-Marken in die entsprechenden Register.

230: stellt das Ende der Display-Liste fest.

240: schreibt die Display-Liste auf den Bildschirm.

Sie sehen, daß alle Standard-Display-Lists mit drei Byte beginnen, die den Wert 112 haben. Sie veranlassen die Ausgabe von je acht leeren Bildschirmzeilen. Es folgt der Wert 64 (LMS) erhöht um die Kennzahl für die Darstellung der ersten Arbeitszeile. Für jeden Grafikgang gibt es eine andere Steuerzahl (s. Tabelle).

Wenn Sie einen Grafik-Modus mit Textfenster aufrufen, sehen Sie, daß die vier Zeilen in GR.0 auch durch ein LMS eingeleitet werden, denn das Textfenster hat einen eigenen Bildschirmspeicherbereich. Deswegen ist es möglich, das Textfenster aus- und wieder einzublenden, ohne daß die Daten verloren gehen.

Bei den Grafik-Betriebsarten, die sehr viel Speicherplatz belegen, können Sie auch sehen, daß die Display-Liste geteilt ist. Grund dafür ist, daß der Bildschirmspeicher keine 4-k-Grenze überschreiten darf (keine Adresse des Bildschirms darf durch 4096 teilbar sein). Ist das der Datenfülle wegen unumgänglich, muß diese Grenze durch einen erneuten LMS-Befehl in der Display-Liste überbrückt werden.

Die Grafik-Modi 9, 10 und 11 verwenden die gleiche Display-Liste wie GR.8. Die beiden höchstwertigen Bits von Adresse 623 erfassen den Unterschied:

Bit 7:	0	Bit 6:	0	(dezimal 0)	GRAPHICS 8
Bit 7:	0	Bit 6:	1	(dezimal 64)	GRAPHICS 9
Bit 7:	1	Bit 6:	0	(dezimal 128)	GRAPHICS 10
Bit 7:	1	Bit 6:	1	(dezimal 192)	GRAPHICS 11

	Grafikp./Zeile	Farben	Zeilen/Grafikp.	GRAPHICS (BASIC)	Byte/Zeile		+	+	+	+	+	+	+	+	HSCROL
								+	+		+	+		+	VSCROL
									+	+	+	+		+	LMS
												+	+	+	DLI
BLK	40	4	8	0	40	0	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242	1 Zeile leer
BLK	40	2	10	–	40	16	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243	2
BLK	40	4	8	12	40	32	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244	3
BLK	40	4	16	13	40	4 bis 7	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245	4 bis 7
BLK	20	5	8	1	20		22	38	54	70	86	102	118	134	150	166	182	198	214	230	246	8
BLK	20	5	16	2	20	112	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247	
JMP						1								129								Sprung
JVB						65								193								Sprung+VB
CHR	40	2	8	0	40	2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242	Character Mode (Zeichen)
CHR	40	2	10	–	40	3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243	
CHR	40	4	8	12	40	4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244	
CHR	40	4	16	13	40	5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245	
CHR	20	5	8	1	20	6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246	
CHR	20	5	16	2	20	7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247	
MAP	40	4	8	3	10	8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248	Memory Map Mode (Grafik)
MAP	80	2	4	4	10	9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249	
MAP	80	4	4	5	20	10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250	
MAP	160	2	2	6	20	11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251	
MAP	160	2	1	14	20	12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252	
MAP	160	4	2	7	40	13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253	
MAP	160	4	1	15	40	14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254	
MAP	320	2	1	8	40	15	31	46	63	79	95	111	127	143	159	175	191	207	223	239	255	

Die Tabelle zeigt Ihnen alle 8-Bit-Kommandos (umgerechnet in Dezimalwerte), die den Video-Prozessor ANTIC steuern. Es gibt Kommandos für ein bis acht leere Bildschirmzeilen. Sie bewirken, daß kein Signal gesendet wird, die Bildröhre bleibt schwarz. Den Wert 112 (acht Leerzeilen) haben Sie ja schon bei den Standard-Display-Lists kennengelernt.

Als nächstes gibt es zwei Sprungbefehle. Der eine (JMP) wird sofort ausgeführt, den werden Sie später noch in einem Programm kennenlernen, der andere (JVB) wartet auf das Vertical Blank der Bildröhre und sorgt so für die Synchronisation zwischen beiden Geräten. Jedem Sprungbefehl muß die Sprungadresse folgen. Da Adressen bis 32767 möglich sind, werden für die Adresse zwei Byte benötigt, die in der Reihenfolge LO, HI dem Sprung folgen.

Die Kommandos für die Darstellungsart zerfallen in zwei Gruppen. ANTIC 2 bis 7 bringen definierte Zeichen auf den Bildschirm. ANTIC 8 bis 15 dienen der Darstellung von Grafikpunkten. Beide Betriebsformen beziehen die Daten, was abgebildet werden soll, aus dem Bildschirmspeicher. Wie der Bildschirmspeicher organisiert ist, lesen Sie in Abschnitt 1.3. Der Character Mode braucht darüber hinaus noch Daten über die Struktur der aufgerufenen Zeichen. Die Gestalt der Zeichen des ATARI-Zeichensatzes, also die Zeichen, die bei entsprechendem Tastendruck auf dem Bildschirm erscheinen (ATASCII-Code) sind im ROM-Bereich gespeichert. Wie man an diese Daten herankommt und wie man einen eigenen Zeichensatz definieren kann, lesen Sie im Kapitel 1.2.

Woher sich ANTIC die Bilddaten holen soll, sagt ihm die LMS-Anweisung (load memory scan). Sie hat den (Dezimal-) Wert 64 und wird der Kennzahl für die erste Arbeitszeile dazuaddiert. In der Tabelle finden Sie die entsprechenden Werte, wenn Sie in den vier Kopfzeilen die Spalte suchen, in der nur in der Zeile LMS ein Pfeil nach unten zeigt.

Wenn ANTIC ein Daten-Byte empfängt, bei dem in den Bits 0 bis 3 ein Character Mode oder Memory Map Mode gesetzt ist und zusätzlich das Bit 6 (dezimal 64), dann verarbeitet er die beiden folgenden Byte als Vektor auf die Start-Adresse des Bildschirmspeichers. Durch das Setzen von Bit 7 (dezimal 128) kann die Display-Liste unterbrochen werden (DLI, display list interrupt). Durch Setzen von Bit 5 (dezimal 32) wird vertikales Rollen (VSCROL, vertical scrolling) des Bildschirminhaltes ausgelöst und durch Bit 4 (dezimal 16) horizontales Scrollen (HSCROL).

In den vorderen Spalten der Tabelle sind die verschiedenen Betriebsarten charakterisiert. Die erste Spalte gibt an, wie

viele Schreibstellen bzw. Grafikpunkte auf einer Zeile liegen. Die zweite Spalte nennt die Anzahl von Farben, die gleichzeitig auf dem Bildschirm darstellbar sind. Die dritte Spalte gibt Aufschluß darüber, aus wie vielen Bildschirmzeilen sich eine Arbeitszeile zusammensetzt. Im Memory Map Mode bedeutet das, wie oft die Daten für eine Bildschirmzeile wiederholt gesendet werden. In der nächsten Spalte sehen Sie, mit welcher GRAPHICS-Kennzahl Sie den korrespondierenden ANTIC-Mode ansprechen können. Und die letzte Spalte sagt, wieviel Bildschirmspeicher eine Zeile in der betreffenden Betriebsart belegt.

3.1.2 Ändern der DL

Jetzt, wo Sie wissen, wie die DL (display list) aufgebaut ist und wie ANTIC herumkommandiert wird, können Sie es bestimmt kaum abwarten, darin herumzuprogrammieren. (Mir geht es ja nicht anders.)

Besonders jene Leser, die noch ein älteres ATARI-Modell ihr stolzes Eigen nennen, werden gerne hören, daß Sie mit nur wenigen BASIC-Zeilen die Betriebsarten GRAPHICS 12, 13, 14 und 15 auch zur Hand haben können. Man nehme:

```
0 REM GR14SIMU.BI2
10 GRAPHICS 24
20 DL=PEEK(560)+256*PEEK(561)
30 FOR I=0 TO 201
40 IF PEEK(DL+I)=15 THEN POKE DL+I,14
50 IF PEEK(DL+I)=79 THEN POKE DL+I,78
60 NEXT I
70 COLOR 1
80 PLOT 0,0:DRAWTO 191,191
90 GOTO 90
```

10: Ob Sie GRAPHICS 8 oder 24 (mit Textfenster) wählen, ist unerheblich. Mit diesem Befehl wird nur eine DL aufgerufen, die anschließend verändert werden soll.

20: errechnet die Start-Adresse der DL

30: GRAPHICS 8 hat die längste Display-Liste, sie ist genau 202 Byte lang.

40: Wo immer in der DL eine 15 steht (Modus-Byte-Zahl für eine GRAPHICS-8-Zeile), wird eine 14 abgelegt,

50: wo eine 79 (15+64) liegt, kommt eine 78 (14+64) hin.

70 bis 90: ziehen eine Linie in GRAPHICS 14.

Wenn Sie aufgepaßt haben und die Tabelle aus dem vorigen Abschnitt noch einmal ansehen, werden Sie hoffentlich einwenden, daß der ANTIC Code für eine Zeile in GRAPHICS 14 nicht 14 sondern 12 ist. Und dann wollen Sie bestimmt wissen, warum in den Zeilen 40 und 50 trotzdem die 14 in alle Adressen gePOKEt wurde?

Gut! Sie erinnern sich, daß die DL von GRAPHICS 8 in der Mitte einen zweiten LMS-Befehl enthielt? Eine Zeile in GR.14 belegt 20 Byte im Bildschirmspeicher, eine Zeile in GR.8 jedoch 40. Die Standard-DL für GR.8 zeigt hinter dem zweiten LMS auf eine Adresse, die 94*40 Byte tiefer liegt als die Start-Adresse, GRAPHICS 14 benötigt aber nur 94*20 Byte für Bild-Daten, um den ersten Teil der DL zu füllen. Welche Folgen das für den Bildschirminhalt hat, sehen Sie sich am besten selbst einmal an. Schreiben Sie in Zeile 40 eine 12 und in Zeile 50 eine 76.

Natürlich können Sie den Vektor nach dem zweiten LMS-Befehl auch ändern. Er müßte 94*20=1880 Byte höher zeigen. Aber da ist es so bequemer, wie es im vorliegenden Programm gelöst ist.

Hinzu kommt noch, daß mit nur einer zusätzlichen Zeile mit dem sonst gleichen Programm auch die Grafik-Betriebsart 15 simuliert werden kann:

```
0 REM GR15SIMU.BI2
10 GRAPHICS 24
20 DL=PEEK(560)+256*PEEK(561)
30 FOR I=0 TO 201
40 IF PEEK(DL+I)=15 THEN POKE DL+I,14
50 IF PEEK(DL+I)=79 THEN POKE DL+I,78
60 NEXT I
70 POKE 87,7
80 COLOR 1
90 PLOT 0,0:DRAWTO 40,60
100 COLOR 2
110 PLOT 10,0:DRAWTO 50,60
120 COLOR 3
130 PLOT 20,0:DRAWTO 60,60
140 GOTO 140
```

70: Mit einer 7, hier in das Register 87 gePOKEt, wird dem Computer vorgegaukelt, der Grafikgang 7 sei eingelegt, also stellt er brav vier verschiedene COLORs zur Auswahl, wovon Sie sich in den Zeilen

80 bis 140: selbst überzeugen können.

Um den Text-Modus GRAPHICS 12 zu simulieren, ist die DL von GR.0 gut geeignet:

```
0 REM GR12SIMU.BI2
10 GRAPHICS 0
20 DL=PEEK(560)+PEEK(561)*256
30 FOR I=0 TO 31
40 IF PEEK(DL+I)=2 THEN POKE DL+I,4
50 IF PEEK(DL+I)=66 THEN POKE DL+I,68
60 NEXT I
70 ? "1234567890"
80 ? "abcdefghik"
90 ? "1234567890"
100 ? "abcdefghik"
```

30: Die DL von GR.0 ist 32 Byte lang.

40: Die Modus-Byte-Zahl für eine Zeile in GR.12 ist 4

50: Eine GR.12-Zeile (4) und LMS (64) ergeben 68

90 und 100: Wenn Sie in diesem beiden Zeilen den String mit negativen Zeichen füllen, dann bekommen Sie auch alle vier Farben auf den Bildschirm. Wie das bunte Durcheinander zustande kommt und warum man die Zeichen kaum noch erkennen kann und viel wichtiger noch, wie man das ändern kann, lesen Sie im Kapitel 1.4.

Die Display-Liste für GRAPHICS 13 sieht nicht viel anders aus. Die Modus-Byte-Zahl ist hier 5:

```
0 REM GR13SIMU.BI2
10 GRAPHICS 0
20 DL=PEEK(560)+PEEK(561)*256
30 FOR I=0 TO 31
40 IF PEEK(DL+I)=2 THEN POKE DL+I,5
50 IF PEEK(DL+I)=66 THEN POKE DL+I,69
60 NEXT I
70 ? "1234567890"
80 ? "abcdefghik"
90 ? "1234567890"
100 ? "abcdefghik"
```

Nun erschöpft sich die Kunst des Programmierens aber nicht damit, vorgekaute Display-Lists ein wenig zu verändern. Wenn Sie Experimentierfreude (und Mut) haben, können Sie sich eine beliebige eigenen DL zusammenPOKEn und dabei Zeilen aus allen Grafik-Töpfen zusammenpanschen. Allerdings gibt es eine ganze Menge, was man dabei falsch machen kann. Eine gute Portion Ausdauer sollten Sie schon mitbringen, denn manche gut gemeinte Display-Liste wird nur ein

schreckliches Durcheinander auf die Mattscheibe schreiben. Aber zu Ihrer Beruhigung, was immer Sie mit der DL anstellen und wie sehr das verursachte Bild den Sinnen weh tun mag, Ihr kostbarer Fernsehapparat oder Monitor kann davon nicht beschädigt werden. Dem Kathodenstrahl ist es so völlig gleichgültig, was er in die Phosphor-Schicht brennt, das können Sie sich gar nicht vorstellen.

Das folgende Programmbeispiel zeigt Ihnen, wie eine Display-Liste aus verschiedenen Grafik-Modi gebaut werden kann: kann:

```
0 REM DISLIST.BI2
10 GRAPHICS 20:POKE 87,2
20 DL=PEEK(560)+PEEK(561)*256
30 POKE DL+0,112
40 POKE DL+1,112
50 POKE DL+2,112
60 POKE DL+3,70
70 POKE DL+4,PEEK(88)
80 POKE DL+5,PEEK(89)
90 POKE DL+6,6
100 POKE DL+7,6
110 POKE DL+8,7
120 POKE DL+9,7
130 POKE DL+10,7
140 POKE DL+11,7
150 POKE DL+12,2
160 POKE DL+13,2
170 POKE DL+14,7
180 POKE DL+15,7
190 POKE DL+16,7
200 POKE DL+17,7
210 POKE DL+18,6
220 POKE DL+19,6
230 POKE DL+20,6
240 POKE DL+21,65
250 POKE DL+22,PEEK(560)
260 POKE DL+23,PEEK(561)
310 FOR X=0 TO 19
320 FOR Y=0 TO 11
330 COLOR 48:PLOT X,Y
340 NEXT Y
350 NEXT X
360 SM=PEEK(88)+PEEK(89)*256
370 FOR J=0 TO 119:POKE SM+240+J,33:NEXT J
500 GOTO 500
```

10: Die DL wird im tiefsten Anwender-RAM-Bereich des Speichers abgelegt, also oberhalb der Adresse 40960. Direkt auf die DL-Daten folgt der Bildschirminhalt. Dieser gesamte Speicherbereich muß gegen Überschreiben von oben gesichert werden. Das erreichen Sie, indem Sie einen Grafikgang aufrufen, der einen größeren Speicherbedarf hat als die DL-Liste plus Bildschirmspeicher, die Sie kreieren wollen. Nur zu diesem Zweck wird hier GR.20 aufgerufen. Mit einer 2 in Adresse 87 wird dem Rechner dann nahegelegt, sich auf die Belange von GRAPHICS 2 einzurichten.

20: Start-Adresse der Display-Liste

30 bis 50: Am Anfang der DL stehen die vertrauten dreimal acht Leerzeilen. Diese Werte müßten hier nicht gePOKEt werden, da ja mit GR.20 auch die entsprechende DL aufgerufen wurde, also stehen in diesen Adressen bereits die Werte 112. Das Programm enthält diese drei Zeilen nur, um einen vollständigeren Überblick zu geben. Und um den Hinweis zu ermöglichen, daß die Display-Liste nicht notwendig mit 24 leeren Zeilen beginnen muß.

60: LMS (64) plus eine Zeile GR.1 (6) = 70

70 und 80: Die Start-Adresse des Bildschirmspeichers (screen memory) findet sich in den Adressen 88 (LO) und 89 (HI).

90 bis 230: Eine bunte Mischung aus GR.1, GR.2 und GR.0. Natürlich ist jede andere Mischung denkbar, Sie müssen nur darauf achten, daß die Summe aller Arbeitszeilen, die Ihre eigene DL enthält umgerechnet in Bildschirmzeilen, die sie belegen, den Wert 192 nicht überschreitet. Ist die DL kürzer, so macht das nichts, da ANTIC ja auf den Vertical Blank wartet. Ist die DL aber länger, so sendet ANTIC mehr Signale als der Bildschirm in einem Durchlauf verarbeiten kann, die Bildinformationen sind länger als das Bild: das Bild fängt an zu laufen.

240: JVB, der Sprungbefehl, der auf den Vertical Blank wartet.

250 und 260: Und die Adresse für den Sprung, die Start-Adresse der Display-Liste.

310 bis 350: Jetzt soll der Bildschirm noch beschrieben werden. Das Betriebssystem glaubt sich durch POKE 87,2 im Garfik-Modus 2. Mit COLOR 48 rufen wir ATASCII 48 in der Farbe aus Register 708 auf. Spalte um Spalte wird die Mattscheibe vollgenullt. Spaltenweise! Bitte beachten Sie, daß in den beiden GRAPHICS-0-Zeilen in der Mitte des Bildes eine Null am linken Rand und dann eine Null in der Mitte ge-

schrieben wird. Das Betriebssystem geht davon aus, daß es sich in GR.2 bewegt und entsprechend ordnet es den Inhalt der Zellen aus dem Bildschirmspeicher Positionen auf dem Bildschirm zu. In GRAPHICS 2 ist aber eine Zeile nur 20 Spalten breit. Folglich interpretiert das Betriebssystem eine GRAPHICS-0-Zeile (40 Schreibpositionen) als zwei Arbeitszeilen zu je 20 Spalten.

Die Bildschirmdaten sind im Speicher hintereinander aufgereiht, eine flächige Anordnung nehmen sie erst auf dem Bildschirm ein. Der eingeschaltete Grafik-Modus setzt für das Betriebssystem fest, wieviele Byte aus dem Bildschirmspeicher eine Arbeitszeile füllen.

Deswegen können in der selbstgebastelten DL auch nicht beliebige Grafik-Zeilen wechseln, wenn ein lesbares Bild möglich werden soll. Wenn z.B., um es gleich am Extrem vorzuführen, in der vorliegenden DL in einer Zeile GR.0 durch GR.8 ersetzt würde, dann käme die Anordnung der Daten aus dem Bildschirmspeicher vollkommen aus dem Gleichschritt.

Der Grafik-Modus, in dem sich das Betriebssystem wähnt, begrenzt auch die zulässigen Cursor-Positionen und damit die Werte für PLOT-, DRAWTO- und POSITION-Kommandos. GRAPHICS 2 läßt nur 12 Zeilen (und 20 Spalten) zu. Da die vorliegende DL 16 Zeilen hat und die beiden GRAPHICS-0-Zeilen doppelt gezählt werden müssen, lassen sich die unteren sechs Zeilen nicht mit der FOR-NEXT-Schleife füllen. Da bleibt dann nur die Möglichkeit, die gewünschten Werte direkt in den Bildschirmspeicher zu POKEn.

360: Der Vektor auf die Start-Adresse der Screen Memory.

370: Zwölf Zeilen á 20 Byte werden durch die Zeilen 310 bis 350 beschrieben. Weitere sechs Zeilen á 20 Byte (=120 Byte) sollen mit Daten gefüllt werden. Zeichen nimmt der Bildschirmspeicher im internen Code auf. Der Wert 33 entspricht dem Buchstaben A (ATASCII 65). Die Schleife hier beschreibt 120 Speicherzellen (0 bis 119) mit dem Wert 33 und beginnt 240 Bytes unter der Start-Adresse der SM (screen memory).

Falls Sie die Behauptungen über den Bildschirmspeicher jetzt etwas verwirren, möchte ich Sie auf das Kapitel 1.3 vertrösten, wo auf diese Fragen ausführlicher eingegangen werden soll.

Nun zwingt Sie natürlich niemand, in Zeile 10 eine 2 in die Adresse 87 zu POKEn. Ich habe es ja auch nur getan, um Ihnen die ganze Problematik erklären zu können. Wenn Sie nämlich dem Betriebssystem einreden, es befände sich in

GRAPHICS 1, dann können Sie zwar auch nur 20 Spalten, aber 24 Zeilen ansprechen. Und das reicht schließlich für die vorliegende DL völlig aus. Ändern Sie also:

```
10 GRAPHICS 20:POKE 87,1
```

und fangen Sie das Programm in Zeile 355 auf:

```
355 GOTO 355
```

Jetzt können Sie in Zeile 320 den Endwert für den Schleifenzähler Y vergrößern. Zählen Sie selbst aus, wieviele Zeilen vollzuschreiben sind. (Aber vergessen Sie nicht, die beiden GRAPHICS-0-Zeilen doppelt zu zählen!)

3.1.3 Subroutine in der DL

Das Verändern der Display-Liste erschließt grenzenlose neue Möglichkeiten, aber vom Programmierer wird eine ganze Menge mehr Aufwand und Aufmerksamkeit gefordert. Die ANTIC-Kommandos müssen bekannt sein und richtig eingesetzt werden, die Eigenarten des Bildschirmspeichers müssen berücksichtigt werden (4-k-Grenze) und die Aufteilung der Bildschirmzeilen in Arbeitszeilen mit den Folgen für die Anordnung der linear gespeicherten Bild-Daten auf der Fläche des Bildschirms muß wohl geplant werden. Schließlich müssen die Daten im Bildschirmspeicher evt. noch neu organisiert werden, um zu vernünftigen Arbeitsabläufen zu kommen, wenn die neu geschaffene Display-Liste mit Leben in Form von Grafikpunkten und/oder Zeichen, womöglich selbstdefinierten, gefüllt werden soll.

Eine andere eindrucksvolle Möglichkeit bietet eine Subroutine in der Display-Liste. Mit diesem Unterprogramm werden die von den Standard-Display-Listen nicht beschriebenen Bereiche am oberen und unteren Rand des Bildschirms gefüllt. Das ist deshalb so interessant, weil der Arbeitsaufwand dafür relativ gering ist. Zum anderen handelt es sich aber um ein Unterprogramm, das von der eigentlichen DL ganz unabhängig arbeitet und seine Bild-Daten aus einem eigenen kleinen Bildschirmspeicher bezieht. Deshalb bleibt die Gestaltung oberhalb oder unterhalb des normalen Bildbereichs unverändert stehen, während sich der von der regulären DL verarbeitete Bildinhalt verändert.

Mit diesem Unterprogramm können Sie also z.B. eine stehende Kopfzeile programmieren, die eine Überschrift enthält, während Sie im GRAPHICS-0-Bereich ein Programm eintippen, Text ausgeben, ein Spiel oder eine Grafik ablaufen lassen.

Lassen Sie sich nicht von dem folgenden Listing verwirren. Es sieht auf den ersten Blick verwirrender aus, als es in Wirklichkeit ist:

```
0 REM DISLTOP1.BI2
10 GRAPHICS 0:POKE 559,0
20 RESTORE 60
30 FOR I=0 TO 27
40 READ A:POKE 1536+I,A
50 NEXT I
60 DATA 0,0,70,8,6,1,0,0
70 DATA 16,17,18,19,20,149,150,151,152,153,80,81,82,8
3,84,213,214,215,216,217
200 DL=PEEK(560)+PEEK(561)*256
210 POKE DL,1:POKE DL+1,0:POKE DL+2,6
220 POKE 1542,PEEK(560)+2:POKE 1543,PEEK(561):POKE 55
9,34
```

10: Hier wird der Grafik-Modus eingeschaltet, in dem der normale Bildschirmausschnitt betrieben werden soll. Mit einer 0 in Register 559 wird ANTIC abgeschaltet. Das empfiehlt sich bei allen Eingriffen in die Display-Liste, weil es sonst u.U. geschehen kann, daß der Prozessor durcheinandergerät und die erwarteten Ergebnisse ausbleiben.

30 bis 50: lesen 28 Daten-Byte und POKEn sie in die Adresse 1536 und die darunter folgenden. Im obersten Speicherbereich ist ein kleiner Abschnitt als Anwender-RAM freigelassen. Es handelt sich um die Adressen 1152 bis 1791. Hier können kurze Maschinensprache-Routinen abgelegt werden, ohne daß Gefahr besteht, daß sie überschrieben werden, denn ein BASIC-Programm kann in diesen Speicherbereich nicht eind-dringen.

60: Diese Zeile enthält die Subroutine für die Display-Liste. Sie ist genauso aufgebaut wie die reguläre DL. Weil sie nur wenige Zeilen auf dem Bildschirm mit Daten versorgt, ist sie ganz kurz.

Oberhalb des üblichen Bildschirmbereichs bleiben bekanntlich 24 Bildschirmzeilen leer. Diese 24 Zeilen können durch diese Subroutine gefüllt werden. Das DATA 70 löst ein LMS (64) aus und richtet eine Zeile in GR.1 ein. Die beiden folgenden Byte (8,6) zeigen auf die Start-Adresse des Bildschirmspeichers. Natürlich brauchen wir für diese Sub-DL einen gesonderten Speicherbereich, der die Bilddaten enthält. Wir legen ihn wie üblich direkt hinter die DL. 8 (LO) und 6 (HI) zeigen auf die Adresse 1544 und das ist genau die Speicherzelle in der das erste DATA aus Zeile

70: abgelegt ist. Dies sind nämlich die Daten-Byte des Sub-Bildschirmspeichers. In eine GRAPHICS-0-Zeile passen 20 Zeichen, deshalb finden sich hier 20 DATA. Zusammen mit den acht Byte der Sub-DL sind das die 28 Byte, die in den Zeilen 30 bis 50 gePOKEt worden sind.

Die Sub-DL in Zeile 60 setzt sich mit einer 1 fort, die einen sofortigen Sprung auslöst. Die beiden 0-en beschreiben die beiden Speicherzellen, in denen das Sprungziel abgelegt werden muß. Diese beiden Byte (LO und HI) werden weiter unten im Programm berechnet und nachträglich gePOKEt.

Es empfiehlt sich, die bislang besprochenen Zeilen am Anfang eines größeren Programmes unterzubringen. Damit ist die Sub-DL zusammen mit ihrer Sub-SM bereits im Speicher vorhanden. Die folgenden Kommandos können dann an der Stelle des Hauptprogrammes gegeben werden, wo die Kopfzeile auf der Bildfläche erscheinen soll.

200: berechnet die Start-Adresse der DL

210: ändert die ersten drei Byte der Display-Liste. Hier steht normalerweise dreimal der Wert 112 (acht Leerzeilen). Diese drei Kommandos werden durch die Folge 1,0,6 ersetzt. Der Wert 1 löst einen Sprung aus, 0 (LO) und 6 (HI) zeigen auf das Sprungziel (1536), nämlich die erste Adresse der Sub-Display-List.

Wenn ANTIC die DL abarbeitet, beginnt er mit einem Sprung in das Unterprogramm, arbeitet die kleine DL dort ab und bringt die Arbeitszeilen mit den dort abgelegten Bild-Daten auf den Schirm. Dann springt er zurück in die reguläre DL und erzeugt das Grafikfenster in der aufgerufenen Betriebsart. Und natürlich wiederholt er diesen Vorgang unentwegt.

Nur die Rücksprungadresse fehlt noch im Unterprogramm! In

220: wird sie in den Adressen 1542 und 1543 abgelegt, das sind genau jene beiden Speicherzellen, die durch die READ-DATA-Anweisung nur vorläufig mit 0-en reserviert wurden. Da nach der Abarbeitung der Sub-DL die reguläre DL bearbeitet werden soll, muß der Rücksprung in die Display-Liste erfolgen, aber nicht an ihren Beginn, sondern zwei Byte tiefer. Zwischen der von der Sub-DL erzeugten GRAPHICS-1-Zeile und dem regulären Bildschirmbereich werden dann nur acht Leerzeilen geschrieben.

Natürlich ist es auch möglich, drei Byte tiefer in die DL zu springen. Dann werden aber gar keine Leerzeilen erzeugt, und der GRAPHICS-0-Bildschirm rutscht um acht Bildschirm-

zeilen nach oben.

Mit POKE 559,34 am Ende von Zeile 220 wird ANTIC wieder eingeschaltet. Solange ANTIC abgeschaltet ist, bleibt der TV-Schirm schwarz. Wenn Sie ein Programm haben, das viel Rechenzeit benötigt, bei dem Sie aber vorübergehend auf eine Bildausgabe verzichten können, dann kann es einen Vorteil bringen, ANTIC abzuschalten, weil das System dadurch entlastet wird und die andere Aufgabe erheblich schneller erledigen kann.

Wenn Sie mit der Sub-DL in einem größeren Programm eine Textzeile wie schon erwähnt einblenden wollen, dann können Sie sie auch wieder ausblenden. Sie brauchen dazu nur die ersten drei Byte der Standard-Display-List wieder mit 112 zu füllen, und der Sprung in die Subroutine unterbleibt.

Das folgende Programm zeigt Ihnen, wie Sie eine GRAPHICS-2-Zeile und eine GRAPHICS-1-Zeile oberhalb des Bildschirmfensters unterbringen:

```
0 REM DISLTOP2.BI2
10 GRAPHICS 0:POKE 559,0
20 RESTORE 60
30 FOR I=0 TO 68
40 READ A:POKE 1536+I,A
50 NEXT I
60 DATA 0,0,71,9,6,6,1,0,0
70 DATA 16,17,18,19,20,149,150,151,152,153,80,81,82,8
3,84,213,214,215,216,217
80 DATA 33,34,35,36,37,38,39,40,41,42,97,98,99,100,10
1,102,103,104,105,106
90 DATA 33,34,35,36,37,38,39,40,41,42,97,98,99,100,10
1,102,103,104,105,106
200 DL=PEEK(560)+PEEK(561)*256
210 POKE DL,1:POKE DL+1,0:POKE DL+2,6
220 POKE 1543,PEEK(560)+2:POKE 1544,PEEK(561):POKE 55
9,34
```

60: Durch das zusätzliche Kommando für eine weitere Zeile ist die Sub-DL jetzt um ein Byte länger. Der Zeiger auf die Bildschirm-Daten muß deshalb um 1 erhöht werden. Die GRAPHICS-2-Zeile verbirgt sich zusammen mit dem LMS in der 71, des folgen LO- und HI-Byte des Zeigers, eine 6 für eine GRAPHICS-1-Zeile und nach dem Rücksprungbefehl der Vektor für den Rücksprung, der nun auch ein Byte tiefer abgelegt werden muß, d.h. in Zeile

220: muß der Zeiger auf die reguläre Display-Liste um dieses eine Byte tiefer gePOKEt werden.

Auf entsprechende Weise kann nun natürlich auch unterhalb des gewöhnlichen Grafikfensters ein zweites Fenster improvisiert werden. Dazu wird die DL an ihrem Ende, vor dem Rücksprung an ihren Anfang für die Subroutine verlassen:

```
0 REM DISLBOT1.BI2
10 GRAPHICS 0:POKE 559,0
20 RESTORE 60
30 FOR I=0 TO 27
40 READ A:POKE 1536+I,A
50 NEXT I
60 DATA 0,0,71,8,6,1,0,0
70 DATA 16,17,18,19,20,149,150,151,152,153,80,81,82,8
3,84,213,214,215,216,217
200 DL=PEEK(560)+PEEK(561)*256
210 POKE DL+29,1:POKE DL+30,0:POKE DL+31,6:POKE DL+32
,65:POKE DL+33,PEEK(560):POKE DL+34,PEEK(561)
220 POKE 1542,PEEK(560)+32:POKE 1543,PEEK(561):POKE 5
59,34
```

Die gesamte Konstruktion der Sub-DL ist identisch mit der bereits beschriebenen Vorgehensweise. Eine Änderung gibt es erst in Zeile

210: Die Sprunganweisung (1) und die beiden Byte für die Sprungadresse werden in die Display-Liste vor dem Sprung nach Vertical Blank (65) abgelegt. Am Anfang der DL war durch die drei Byte für Leerzeilen genügend Platz für die Sprunganweisung vorhanden. In dieser Version wird die reguläre DL um drei Byte länger. Der Sprung nach Vertical Blank mit dem Sprungziel muß also auch um drei Byte nach unten verlegt werden.

Daraus ergibt sich nun aber ein kleines Problem. Direkt hinter der Display-Liste beginnt der Bildschirmspeicher von GRAPHICS 0. Und das Betriebssystem hat auch keinen Anlaß, daran etwas zu ändern. D.h. die drei zusätzlichen Byte der DL werden in die ersten drei Zellen des Bildschirmspeichers geschrieben. Die entsprechenden Werte nach dem internen Code in ATARI-Standard-Zeichen umgewandelt erscheinen in der linken oberen Ecke des Bildschirms. Das ist deshalb kein besonderes Problem, weil ein einfacher CLEAR-Befehl (CHR$(125)) die unerwünschten Zeichen von der Mattscheibe putzt.

220: Natürlich muß auch der Rücksprung-Vektor aus der Subroutine geändert werden.

Im unteren Bereich des Bildschirms können etwa weitere 20 Zeilen genutzt werden. Genügend Platz für eine GRAPHICS-2-

Zeile oder zwei GRAPHICS-1-Zeilen. Auf diese Weise können so zusätzlich zu den gewöhnlichen 192 Zeilen am oberen Rand weitere 24, am unteren Rand weitere 20 gefüllt werden. Insgesamt werden dann 236 Bildschirmzeilen genutzt.

Abschließend noch eine Bemerkung zu den Daten, welche die Schriftzeichen in die zusätzlichen Zeilen hineinbringen: Die Bild-Daten der Sub-DL werden selbstverständlich genauso verarbeitet wie Daten aus dem regulären Bildschirmspeicher. und diese Daten enthalten die Zeichen, die an der korrespondierenden Stelle auf der Mattscheibe erscheinen sollen in Form des internen Codes. Das gilt für GR.0. Im Anhang finden Sie eine Tabelle, aus der Sie den internen Code eines Zeichens ablesen können.

Das gilt aber auch für GR.1 und GR.2. Hier ist jeweils nur der halbe Standard-Zeichensatz aufrufbar, entweder die Ziffern, Satzzeichen und Großbuchstaben oder aber der Rest des Zeichensatzes. Die 256 verfügbaren Zahlenwerte sind so verteilt, daß es für jedes der 64 verfügbaren Zeichen vier Werte gibt, von denen sich jeder auf ein anderes Farbregister bezieht, das entsprechende Zeichen also in wechselnden Farben darstellt. Auch für diesen GRAPHICS-1/2-Code finden Sie im Anhang eine übersichtliche Tabelle.

Diese GRAPHICS-1/2-Codezahlen, die zusammen mit dem COLOR- verwendet werden, um in BASIC die Zeichen aufzurufen und mit PLOT (und DRAWTO) in das Grafikfenster zu bewegen, werden nun zur Ablage im Bildschirmspeicher gleichfalls nach dem Muster des internen Codes umgewandelt. Da diese Umwandlung zwar ganz systematisch, aber doch unübersichtlich und schwer zu memorieren ist, finden Sie auch für den internen GRAPHICS-1/2-Code eine Tabelle im Anhang.

3.2 Zeichensatz

Die Kommunikation zwischen Computer und Benutzer erfolgt in Form geschriebener Zeichen. Dem Benutzer steht die Tastatur zur Verfügung, über die er Kommandos und Daten eingibt. Zur Kontrolle erscheinen die eingegeneben Zeichen in der Regel auf dem Bildschirm. Der Output des Rechners wird heute ebenfalls meist auf einem Monitor dargestellt. Es geht also nicht umhin, der Rechner muß das ABC lernen. Und wenn Sie sich mit Ihrem Blechkameraden besser verstehen wollen, dann sollten Sie wissen, was in seinem Inneren vor sich geht, wenn er sich mit Ihnen unterhält.

Die Probleme fangen damit an, daß die künstliche Intelligenz nur zählen kann. Und zwar von 0 bis 1. Um diesen Schwall digitaler Informationen etwas besser in den menschlichen Griff zu bekommen, werden die Dualzahlen (0/1) zu Hexadezimalzahlen (0 bis F) gebündelt. Vier (duale) Bit ergeben einen Hex-Wert.

Heim-Computer arbeiten heute alle mit 8-Bit-Prozessoren, d.h. sie können acht Bit gleichzeitig verarbeiten, das sind Werte von 00000000 bis 11111111 dual, von 00 bis FF hexadezimal oder von 0 bis 255 dezimal. Im ATARI-BASIC werden alle Eingaben in dezimaler Form verlangt, so daß der gesamte Komplex der Hexadezimalrechnung hier ausgeklammert werden kann.

Die gängige 8-Bit-Bündelung wird als ein Byte bezeichnet. Bei einem 8-Bit-Prozessor ist jedes Datenwort ein Byte groß und jeder Speicherplatz kann ein Byte aufnehmen. Die Speicherzellen sind fortlaufend numeriert. Der im ATARI verwendete 6502 Mikroprozessor kann Speicherplätze von 0 bis 32676 verwalten.

Jede Taste/Tastenkombination des Keyboard ist mit einem Zahlenwert von 0 bis 255 gekennzeichnet, belegt also ein Byte Information. Der gesamte Zeichensatz ist eine Zuordnung aller verfügbaren Zeichen zu einem acht Bit großen Dezimalwert. Und für diese Zuordnung gibt es eine internationale Norm, den ASCII-Code.

Dieser Code sollte einmal im Fernschreibverkehr sicherstellen, daß sendende und empfangende Geräte die gleiche (Zeichen-) Sprache sprechen. Die Computer-Techniker haben diesen Standard übernommen. Allerdings haben sie sich auch verschiedentlich Abweichungen erlaubt.

So dienen nämlich die ASCII-Werte 0 bis 31 der Steuerung des Druckers. Funktionen wie Zeilenvorschub, Klingelzeichen

etc. werden durch diese Werte ausgelöst. Das sind Funktionen, die der Heim-Computer nicht benötigt. Und weil Computer-Techniker notgedrungen (informations-) sparsame Menschen sind, werden die ASCII-Werte 0 bis 31 meist in irgendeiner anderen Weise genutzt, meist für sog. Block- oder Pseudo-Grafik.

Während der ASCII-Code 7 die Signalglocke des Druckers erklingen läßt, schreibt CHR$(7), also der in ein Zeichen umgewandelte Dezimalwert 7 beim ATARI eine diagonale Linie auf den Bildschirm. Stolz haben die Konstrukteure ihren Zeichensatz ATASCII-Code getauft. Klingt ja auch ganz toll. Die Werte 0 bis 31 stellen Grafikzeichen dar, 32 ist das Leerzeichen, es folgen Satzzeichen, Ziffern, Großbuchstaben und Kleinbuchstaben. Eine vollständige Übersicht finden Sie im Anhang.

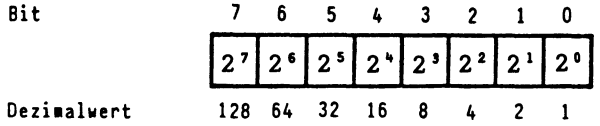
Insgesamt stehen 128 verschiedene Zeichen zur Verfügung. Dazu zählen auch solche Funktionen wie die Cursor-Steuerung oder das Bildschirm-Löschen. Die Zeichen des Zeichensatzes haben also nur sieben Bit Information. Die zweite Hälfte des Zeichensatzes (ATASCII 128 bis 255) ist mit dem ersten identisch, die Zeichen werden lediglich invertiert dargestellt, also dunkel auf hellem Grund. Das achte Bit bestimmt, ob ein Zeichen normal oder invertiert erscheinen soll. Der Computer hat aber nur 128 Zeichen zu lernen.

3.2.1 Bit-Muster

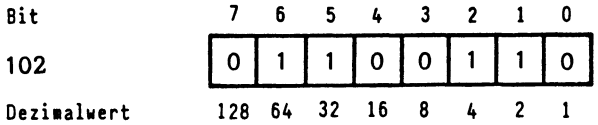
Der Rechner selbst würde sich in seiner zahlen-asketischen Bescheidenheit mit den ATASCII-Werten zufrieden geben, wenn aber Schriftzeichen auf dem Bildschirm erscheinen sollen, dann müssen der Maschine deren Muster eingegeben werden. Auf der Mattscheine erscheinen die Zeichen in einer Schreibposition von acht mal acht Bildpunkten. ATARI nutzt die Bildröhre in einer Breite von 320 Punkten und einer Höhe von 192 Zeilen. Daraus errechnet sich, daß bei der Textdarstellung 40 Zeichen nebeneinander in 24 Zeilen untereinander dargestellt werden können.

Und jetzt bringen wir dem Rechner das Schreiben bei:

Ein Byte enthält acht digitale Informationen (Bit). Ein Bildpunkt kann leuchten oder nicht leuchten, eingeschaltet oder nicht eingeschaltet sein, hat also ebenfalls einen Informationswert von einem Bit. Ein Byte kann demnach die Informationen für acht Bildpunkte (pixel) aufnehmen. Und das geht so:

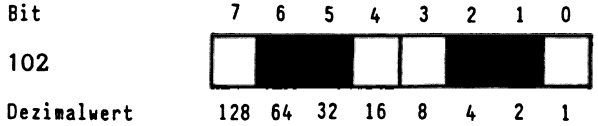


Ein Byte setzt sich aus acht Binärstellen von 2^0 (LSB, least significant bit, niederwertiges Bit) bis 2^7 (MSB, most significant bit, höchstwertiges Bit) zusammen. Jedes dieser acht Bits (flags) kann gesetzt (eingeschaltet = 1) oder nicht gesetzt (ausgeschaltet = 0) sein. Der Zustand eines Byte könnte so aussehen:

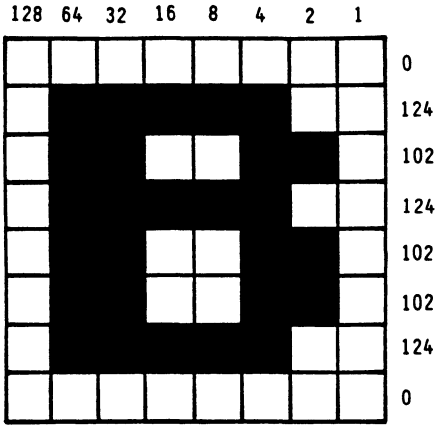


Diese Dualwerte des Byte zu einem Dezimalwert umgerechnet ergeben 0+64+32+0+0+4+2+0=102. Um umgekehrt, wird in eine Speicherzelle der Wert 102 geschrieben (POKE n,102), dann werden dort die Bits entsprechend der Abbildung gesetzt.

Auf dem Bildschirm wird jedes Bit in einen Bildpunkt umgesetzt. Der Dezimalwert 102 verursacht dieses Bitmuster auf dem Bildschirm:



Da der Monitor zeilenweise arbeitet, werden die Bildpunkte auch zeilenweise zu Bytes zusammengefaßt. Da eine Schreibposition auf dem Bildschirm acht Zeilen hoch ist, besteht jedes Zeichen aus acht Byte:



Ein bestimmter Teil im oberen Bereich des Speichers enthält die Daten für den ATARI-Standard-Zeichensatz. Es handelt sich um einen ROM-Bereich, die hier abgelegten Daten können also nur gelesen, nicht aber verändert werden.

Der Zeichensatz beginnt mit der Adresse 57344. Dort liegt das 0-te Byte des 0-ten Zeichens. In den folgenden Adressen liegen die übrigen sieben Byte des 0-ten Zeichens. In der nächsten Adresse findet sich dann das 0-te Byte des ersten Zeichens usf. bis zum siebten Byte des 127. Zeichens. Da die Daten von 128 Zeichen gespeichert sind und jedes Zeichen acht Byte beelgt, benötigt der Zeichensatz genau 1 kByte (Kilo-Byte = 1024 Byte) Speicher und endet somit in der Adresse 58368.

Wenn die Daten eines bestimmten Zeichens gesucht werden, dann läßt sich leicht ausrechen, in welchen Speicherzellen sie liegen, denn der ATASCII-Wert gibt die Reihenfolge der Zeichen an, jedes Zeichen ist acht Byte lang, also muß der ATASCII-Wert mit 8 multipliziert werden, um die erste Adresse des gesuchten Zeichens zu finden.

Einen Moment mal! So einfach geht das nun auch wieder nicht. Die Daten des Zeichensatzes liegen nämlich nicht in der Reihenfolge des ATASCII-Codes im Speicher, sondern sind in drei Blöcken versetzt abgelegt. Diese Reihenfolge wird interner Code genannt. Hier eine Tabelle zur Umrechnung:

ATASCII-Werte umwandeln in internen Code

ATASCII	0	bis	31	und	128	bis 159	+64
ATASCII	32	bis	95	und	160	bis 223	-32
ATASCII	96	bis	127	und	224	bis 255	identisch

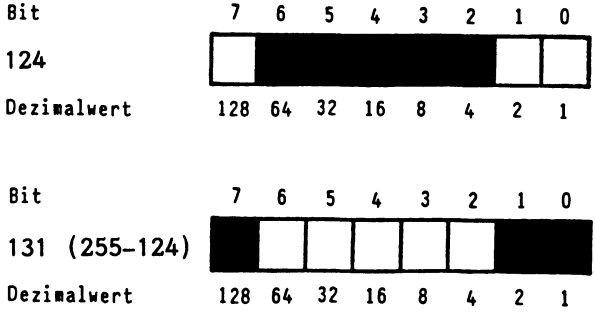
Internen Code umwandeln in ATASCII-Werte

Intern	0	bis	63	und	128	bis 191	+32
Intern	64	bis	95	und	192	bis 223	-64
Intern	96	bis	127	und	224	bis 255	identisch

Da die Werte des internen Codes angeben, in welcher Entfernung von der Start-Adresse des Zeichensatzes die Daten des entsprechenden Zeichens liegen, nennt man diesen Wert auch Offset. Um die Daten des Zeichens n zu finden, muß der Inhalt der Adresse 57344+n*8 und der folgenden sieben Speicherzellen gelesen werden:

```
FOR I=0 TO 7:PEEK(57344+OFF*8+I):NEXT I
```

Die Daten für die zweite Hälfte des Zeichensatzes brauchen nicht extra gespeichert zu werden, da sie sich ganz einfach aus den vorhandenen Daten ermitteln lassen. Wird nämlich das Bit-Muster invertiert, aus 0-en werden 1-en, aus 1-en werden 0-en, ist der Dezimalwert des invertierten Bit-Musters gleich der Differenz von 255 zum Dezimalwert des normalen Bit-Musters:



Mit dem folgenden Programm werden die Zeichensatz-Daten aus dem Speicher gelesen und dann in vergrößerter Form auf dem Bildschirm dargestellt. Gleichzeitig erscheinen im Textfenster die acht Daten-Byte des abgebildeten Zeichens:

```
0 REM BITMSTR.BI2
10 GRAPHICS 3:DIM F(7,7):POKE 765,2:POKE 712,0:POKE 7
10,0:POKE 709,50:POKE 708,248:POKE 752,1
```

```
20 COLOR 2:PLOT 15,19:DRAWTO 15,4:DRAWTO 0,4:POSITION
 0,19:XIO 18,#6,0,0,"S:"
30 COLOR 2:PLOT 39,19:DRAWTO 39,4:DRAWTO 24,4:POSITIO
N 24,19:XIO 18,#6,0,0,"S:"
40 COLOR 2:PLOT 23,7:DRAWTO 23,4:DRAWTO 16,4:POSITION
 16,7:XIO 18,#6,0,0,"S:"
50 COLOR 2:PLOT 23,19:DRAWTO 23,16:DRAWTO 16,16:POSIT
ION 16,19:XIO 18,#6,0,0,"S:"
60 POKE 765,3:COLOR 3:PLOT 23,15:DRAWTO 23,8:DRAWTO 1
6,8:POSITION 16,15:XIO 18,#6,0,0,"S:"
100 FOR I=0 TO 7:FOR J=0 TO 7:F(I,J)=3:NEXT J:NEXT I
110 OPEN #1,4,0,"K:":GET #1,Z:CLOSE #1
120 IF Z<128 THEN POKE 710,0:POKE 708,248:P=0
130 IF Z>127 THEN Z=Z-128:POKE 710,248:POKE 708,0:P=1
140 IF Z<32 THEN Z=Z+64:GOTO 200
150 IF Z<96 THEN Z=Z-32:GOTO 200
200 FOR I=0 TO 7
210 A=PEEK(57344+I+Z*8)
220 IF A>127 THEN A=A-128:F(0,I)=1
230 IF A>63 THEN A=A-64:F(1,I)=1
240 IF A>31 THEN A=A-32:F(2,I)=1
250 IF A>15 THEN A=A-16:F(3,I)=1
260 IF A>7 THEN A=A-8:F(4,I)=1
270 IF A>3 THEN A=A-4:F(5,I)=1
280 IF A>1 THEN A=A-2:F(6,I)=1
290 IF A>0 THEN F(7,I)=1
300 NEXT I
310 FOR J=0 TO 7:FOR I=0 TO 7:COLOR F(I,J):PLOT I+16,
J+8:NEXT I:NEXT J
320 IF P=0 THEN ? :FOR I=0 TO 6:? PEEK(57344+I+Z*8);"
,";:NEXT I:? PEEK(57344+7+Z*8);"
    "
330 IF P=1 THEN ? :FOR I=0 TO 6:? 255-PEEK(57344+I+Z*
8);",";:NEXT I:? 255-PEEK(57344+7+Z*8);"
    "
350 GOTO 100
```

10: GRAPHICS 3 ermöglicht die vergrößerte Darstellung der Zeichen ohne großen Aufwand. Jedes Bit erzeugt einen Grafik-Punkt, der acht Bildpunkte breit und acht Bildschirmzeilen hoch ist.

20 bis 60: bauen die Bildschirmgrafik auf.

100: setzt die doppelt-indizierte Variable auf 3.

110: fragt die Tastatur ab und liest den ATASCII-Wert der gedrückten Taste in die Variable Z ein.

120: Wenn der Wert Z kleiner als 128 ist, wurde ein Zeichen in normaler Darstellung aufgerufen, entsprechend werden die Farbwerte gePOKEt und in der Variablen P wird diese Tatsache protokolliert.

130: Wurde ein inverses Zeichen aufgerufen (Z größer 127), dann werden die Farbwerte genau umgekehrt gePOKEt. Dadurch entsteht der inverse Eindruck. Auch dieser Sachverhalt wird in P aufgeschrieben.

140 und 150: wandeln den ATASCII-Wert in internen Code um.

200 bis 300: holen aus dem Speicherbereich des Zeichensatzes die acht Daten-Byte heraus, die das aufgerufene Zeichen darstellen. Jedes dieser Bytes wird in seine Bits aufgespalten und in der Variablen F(n,n) aufbewahrt. Nach acht Durchläufen der FOR-NEXT-Schleife befinden sich in dieser doppeltindizierten Variablen acht mal acht Bit Information.

310: wandelt jedes einzelne Bit in einen Grafikpunkt um. Der Wert des Bit (0 oder 1) bestimmt den COLOR-Wert für den Grafikpunkt, die Positionen für den PLOT ergeben sich aus der Stellung des Bit innerhalb des Gesamtzeichens.

320: Gleichzeitig sollen die acht Daten-Byte des Zeichens im Textfenster erscheinen. Wenn P gleich 0 ist (normale Darstellung), dann können die im Zeichensatz gefundenen Daten direkt ins Textfenster gePRINTet werden.

330: Wenn inverse Darstellung vorliegt (P=1), dann wird der inverse Wert (255 minus gefundenem Dezimalwert) ins Textfenster gedruckt.

Mit diesem Wissen über die Speicherung und Darstellung von Zeichen liegt es nahe, einen eigenen Zeichensatz zu gestalten. Das macht Spaß, wenn man die ewig gleiche ATARI-Schrift leid ist und etwas Abwechselung sucht (oder wenn man mit deutschen Umlauten fremdgehen will oder gar mit einem runden 'ß'!) und ist sogar nützlich, wenn man sich mathematische, elektronische, chemische oder sonstige Symbole zurechtlegt.

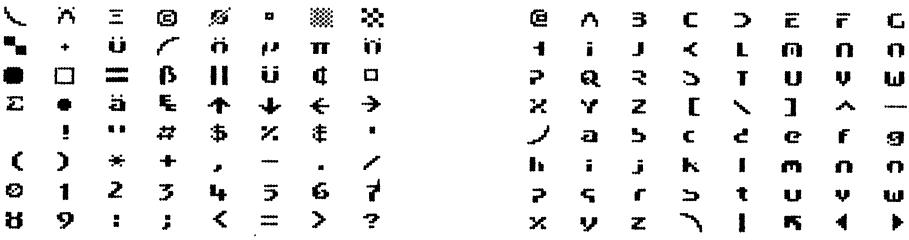
Die Gestaltung des eigenen Zeichensatzes ist pure Fleißarbeit: Bitmuster entwerfen und die Dezimalwerte der Byte ausrechnen. Nur mit dem Abspeichern der Daten ist das so eine Sache. Der Standard-Zeichensatz liegt ja wohl behütet im ROM-Bereich. Deshalb hier erst einmal als Kostprobe einen frei definierten Zeichensatz in DATA-Zeilen:

```
0 REM ARCHIV.DAT
32723 DATA 0,24,24,24,24,0,24,0,0,102,102,102,0,0,0,0
```

```
,0,36,126,36,72,252,72,0
32724 DATA 16,60,80,60,22,86,60,16,0,102,108,24,48,10
2,70,0,16,60,80,60,80,80,60,16
32725 DATA 0,24,24,24,0,0,0,0,6,12,24,24,24,24,12,6,9
6,48,24,24,24,24,48,96
32726 DATA 0,36,24,126,24,36,0,0,0,24,24,126,24,24,0,
0,0,0,0,0,0,24,24,48
32727 DATA 0,0,0,126,0,0,0,0,0,0,0,0,0,24,24,0,0,6,12
,24,48,96,64,0
32728 DATA 0,60,102,106,82,102,60,0,0,24,56,24,24,24,
24,24,56,12,12,24,48,96,124,0
32729 DATA 0,124,24,48,12,12,24,112,0,96,96,108,108,1
26,12,12,0,62,0,60,6,6,12,56
32730 DATA 28,48,96,108,118,102,60,0,0,62,6,12,24,24,
24,24,0,102,102,60,102,102,102,60
32731 DATA 0,60,102,102,54,12,24,48,0,0,24,24,0,24,24
,0,0,0,24,24,0,24,24,48
32732 DATA 6,12,24,48,24,12,6,0,0,0,126,0,0,126,0,0,9
6,48,24,12,24,48,96,0
32733 DATA 0,60,102,12,24,0,24,0,60,98,94,82,94,96,62
,0,0,24,60,36,102,66,66,0
32734 DATA 0,120,12,120,12,12,120,0,0,60,96,96,96,96,
60,0,0,120,12,6,6,12,120,0
32735 DATA 0,124,0,120,96,96,124,0,0,124,0,120,96,96,
96,0,0,60,96,96,98,98,62,0
32736 DATA 0,12,12,60,12,12,12,0,0,24,0,24,24,24,24,0
,0,24,24,24,24,24,112,0
32737 DATA 0,12,24,112,112,24,12,0,0,48,48,48,48,48,6
0,0,0,124,198,214,214,214,198,0
32738 DATA 0,60,102,102,102,102,102,0,0,60,102,102,10
2,102,36,0,0,120,12,12,120,96,96,0
32739 DATA 0,60,102,70,118,108,54,3,0,120,12,12,112,2
4,12,0,0,96,56,12,6,70,60,0
32740 DATA 0,60,24,24,24,24,24,0,0,102,102,102,102,10
2,60,0,0,108,108,108,108,56,16,0
32741 DATA 0,198,198,214,214,214,124,0,0,102,38,28,56
,100,102,0,0,70,110,44,24,24,24,0
32742 DATA 0,124,12,24,48,96,124,0,30,24,24,24,24,24,
24,30,0,64,96,48,24,12,6,0
32743 DATA 120,24,24,24,24,24,24,120,0,8,28,54,99,0,0
,0,0,0,0,255,0,0,0,0
32744 DATA 128,128,192,64,96,48,28,7,66,24,60,36,102,
66,66,0,0,124,0,56,0,0,124,0
32745 DATA 0,124,198,186,162,186,198,124,1,58,100,106
,86,38,92,128,0,0,56,40,56,0,0,0
32746 DATA 170,85,170,85,170,85,170,85,204,204,51,51,
204,204,51,51,240,240,240,240,15,15,15,15
```

```
32747 DATA 0,0,0,16,56,16,0,0,36,0,102,102,102,102,60
,0,7,28,48,96,64,192,128,128
32748 DATA 36,0,60,102,102,102,36,0,0,0,38,102,102,10
8,64,64,0,0,254,108,108,108,108,0
32749 DATA 66,24,102,102,102,102,36,0,0,124,254,254,2
54,254,254,124,0,254,130,130,130,130,130,254
32750 DATA 0,255,255,0,0,255,255,0,56,108,104,108,102
,102,108,96,102,102,102,102,102,102,102,102
32751 DATA 102,0,102,102,102,102,60,0,8,60,104,104,10
4,104,60,8,0,124,68,68,68,124,0,0
32752 DATA 0,126,34,16,48,98,126,0,0,0,56,124,124,124
,56,0,36,0,60,6,54,102,62,0
32753 DATA 120,96,120,96,126,24,30,0,0,24,60,126,219,
24,24,24,0,24,24,24,219,126,60,24
32754 DATA 0,24,48,96,254,96,48,24,48,24,12,254,12,24
,48,0,1,1,3,2,6,12,56,224
32755 DATA 0,0,60,6,54,102,62,0,0,96,96,120,12,12,120
,0,0,0,56,96,96,96,56,0
32756 DATA 0,12,12,60,96,96,60,0,0,0,60,102,108,96,60
,0,0,28,48,56,48,48,48,0
32757 DATA 0,0,62,102,110,54,70,60,0,96,96,104,108,10
8,108,0,0,0,24,0,24,24,24,0
32758 DATA 0,0,24,0,24,24,24,112,0,96,96,116,120,108,
102,0,0,24,24,24,24,24,24,0
32759 DATA 0,0,124,254,214,214,198,0,0,0,60,102,102,1
02,102,0,0,0,60,102,102,102,36,0
32760 DATA 0,0,60,6,6,60,48,48,0,0,60,96,96,60,12,12,
0,0,28,48,48,48,48,0
32761 DATA 0,0,96,60,6,6,124,0,0,24,60,24,24,24,12,0,
0,0,102,102,102,102,60,0
32762 DATA 0,0,108,108,108,56,16,0,0,0,198,214,214,21
4,124,0,0,0,102,44,24,52,102,0
32763 DATA 0,0,102,102,102,44,24,48,0,0,124,24,48,96,
124,0,224,56,12,6,2,3,1,1
32764 DATA 24,24,24,24,24,24,24,24,0,126,120,124,110,
102,6,0,8,24,56,120,56,24,8,0
32765 DATA 16,24,28,30,28,24,16,0
```

Und so sehen die neuen Zeichen aus:



Wie diese Daten in den Speicher des Computers und von dort auf den Monitor kommen, werden wir noch sehen. Mit dem folgenden Programm wurde der Ausdruck der neu definierten Zeichen auf einem Matrix-Drucker erledigt:

```
0 REM DEFCHRPR.BI2
10 DIM B(7)
20 FOR I=0 TO 7:READ M:B(I)=M:NEXT I
30 OPEN #1,8,0,"P:"
40 PUT #1,27:PUT #1,75:PUT #1,8:PUT #1,0
50 FOR I=7 TO 0 STEP -1
60 PUT #1,B(I)
70 NEXT I
80 CLOSE #1
90 GOTO 20
```

Das Programm verarbeitet die Daten als Dot-Grafik. Das bedeutet, daß die Nadeln des Druckkopfes einzeln angesprochen werden, um bestimmte Punktmuster hervorzubringen. Wie das genau geschieht, wird in Kapitel 4 erörtert.

Im normalen Schreibbetrieb empfängt der Drucker die (AT)ASCII-Werte und wandelt sie in Zeichen um. Welche Punktmuster ein Zeichen darstellen, bestimmt dabei sein eigener Zeichensatz. Die Schrift eines Druckers sieht also immer anders aus als die Zeichen, die der Rechner auf dem Bildschirm erzeugt.

Deshalb ist kein Drucker von vorne herein in der Lage, z.B. den vollständigen ATARI-Zeichensatz inkl. Pseudo-Grafikzeichen auf das Papier zu bringen.

Bei der Dot-Grafik wird ein Punktraster gedruckt, der mit der Bildauflösung des Monitors vergleichbar ist. Der wesentliche Unterschied besteht darin, daß die Drucknadeln in einer Spalte untereinander angeordnet sind, also in jedem Arbeitsschritt acht Punkte untereinander drucken oder nicht drucken und deswegen die Bit-Muster für den Drucker auch spaltenweise zu Dezimalwerten verrechnet werden, während die Grafikdaten im Rechner in der Zeile Byte-weise zusammengefaßt werden:

20: Liest acht Daten-Byte aus den DATA-Zeilen.

30: eröffnet einen Schreib-Kanal zum Drucker.

40: Stellt den Drucker auf Dot-Grafik ein und bereitet ihn auf acht Dot-Grafik-Daten vor. Dieses Kommando ist für den radix-10 von star geschrieben. Wenn Sie einen anderen Drucker verwenden, müssen Sie diese Zeile evt. entsprechend umschreiben, denn gerade bei der Dot-Grafik unterscheiden

sich die Formate der Kommandos recht stark. Auch hierüber wird in Kapitel 4 ausführlich gesprochen.

50 bis 70: senden acht Byte an den Drucker, die dort in Grafikpunkte (dot pattern) umgesetzt gedruckt werden.

3.2.2 Neue Zeichensätze

Nun ist es allerdings recht mühsam, einen kompletten Zeichensatz manuell zu ermitteln, denn neben der eigentlichen Gestaltungsarbeit sind sehr viele (exakt 1024) Werte zu berechnen. Und für solche Routinearbeiten ist doch der Computer geschaffen worden.

Der folgende Zeichensatz-Generator übernimmt den unangenehmsten Teil der Arbeit, und Sie können sich weitgehend dem kreativen Aspekt zuwenden. Es empfiehlt sich übrigens, eine Tabelle mit dem ATARI-Zeichensatz und den Werten von 0 bis 127 (in der Reihenfolge des internen Codes!) anzufertigen und darin die neu entworfenen Zeichen zu skizzieren. So haben Sie bei der Entwurfsarbeit immer einen Überblick, welche Zeichen schon umgestaltet sind, und Sie können später leicht feststellen, welche Taste Sie drücken müssen, um ein neu definiertes Zeichen aufzurufen:

```
0 REM TYPGENE.BI2
10 GOSUB 1000
20 GOSUB 2000
30 IF PEEK(53279)<>5 THEN SOUND 0,136,12,2:GOTO 30
40 SOUND 0,0,0,0:GOSUB 3000
100 SOUND 0,0,0,0:SOUND 1,0,0,0:N=0
110 OPEN #1,4,0,"K:":GET #1,BY:SOUND 0,120,14,2
120 IF BY=83 THEN GOSUB 5000
130 IF BY=76 THEN SOUND 0,0,0,0:CLOSE #1:GOSUB 2040
150 GET #1,BI:SOUND 1,140,14,3:CLOSE #1
160 IF BY<65 THEN 100
170 IF BY>72 THEN 100
180 IF BI<48 THEN 100
190 IF BI>55 THEN 100
200 X=70-BI:Y=BY-58:P=40000+X+Y*40:D=PEEK(P)
210 IF D=0 THEN POKE P,128
220 IF D=128 THEN POKE P,0
230 FOR I=0 TO 7:N=N+(PEEK(40022+Y*40-I)/128)*2^I:NEX
T I
240 POSITION 26,Y:? N;"  ":S(Y-7)=N:GOTO 100
1000 ? CHR$(125):POKE 710,114:POKE 82,0:POKE 752,1:?
:? "          ZEICHENSATZ-GENERATOR"
```

```
1010 ? "_____________________________________":? :
?
1020 ? " SIE KOENNEN 128 ZEICHEN NEU GESTALTEN":? :?
1030 ? " TASTEN SIE GLEICHZEITIG <SELECT> UND":?
1040 ? " DAS NEU ZU GESTALTENDE ZEICHEN":? :?
1050 ? " DAS VERAENDERTE ZEICHEN SPEICHERN SIE":?
1060 ? " MIT TASTE <S>"
1070 ? :? :? :? :? " WEITER MIT <START>"
1080 DIM D$(12),F(7,7),S(7),DAT$(15),C$(1),U(127):DAT
$="D1:":C$=CHR$(155)
1090 IF PEEK(53279)<>6 THEN GOTO 1090
1100 ? CHR$(125):? :? "          ZEICHENSATZ-GENERATOR
"
1110 ? "_____________________________________":? :
?
1120 ? " DIE DATEN DER UMGESTALTETETEN ZEICHEN":?
1130 ? " WERDEN ALS BASIC-DATA-ZEILEN AUF DIE":?
1140 ? " DISKETTE IN LAUFWERK 1 GESCHRIEBEN":? :? :?
1150 ? " GEBEN SIE EINEN NAMEN FUER DIE DATEI":?
1160 CLOSE #2:TRAP 1180:POSITION 0,17:INPUT D$
1170 DAT$(LEN(DAT$(1,3))+1)=D$:OPEN #2,8,0,DAT$:GOTO
1200
1180 POSITION 1,15:? ">DATEINAME FEHLERHAFT<
     "
1190 POSITION 1,17:? "                        ":GOTO 1160
1200 ? CHR$(125):? :? "          ZEICHENSATZ-GENERATOR
"
1210 ? "_____________________________________":? :
?
1220 ? " GEBEN SIE DIE NUMMER FUER DIE ERSTE":?
1230 ? " BASIC-DATA-ZEILE":?
1240 TRAP 1260
1250 POSITION 1,11:INPUT ZN:GOTO 1270
1260 POSITION 1,9:? ">FEHLEINGABE<          ":POSITION
 1,11:? "             ":GOTO 1240
1270 IF ZN>32639 THEN GOTO 1260
1300 POSITION 1,15:? "GEBEN SIE DIE SCHRITTWEITE FUER
 DIE":?
1310 ? " ZEILENNUMMERN"
1320 TRAP 1340
1330 POSITION 1,19:INPUT SW:GOTO 1350
1340 POSITION 1,17:? ">FEHLEINGABE<          ":POSITIO
N 1,19:? "             ":GOTO 1320
1350 FOR I=0 TO 127:U(I)=I:NEXT I:RETURN
2000 ? CHR$(125):POKE 710,0
2010 POSITION 15,2:? "  BIT#  "
2020 POSITION 15,4:? "76543210"
```

```
2030 POSITION 14,6:? "┌──────────┐"
2040 POSITION 12,7:? "A │          │"
2050 POSITION 12,8:? "B │          │"
2060 POSITION 10,9:? "B C │          │"
2070 POSITION 10,10:? "Y D │          │"
2080 POSITION 10,11:? "T E │          │"
2090 POSITION 10,12:? "E F │          │"
2100 POSITION 12,13:? "G │          │"
2110 POSITION 12,14:? "H │          │"
2120 POSITION 14,15:? "└──────────┘"
2130 IF BY=76 THEN FOR I=0 TO 7:S(I)=0:NEXT I:POP :GO
SUB 3250
2140 RETURN
3000 OPEN #1,4,0,"K:":GET #1,Z:CLOSE #1
3010 IF Z>127 THEN Z=Z-128
3020 IF Z<32 THEN Z=Z+64:GOTO 3040
3030 IF Z<96 THEN Z=Z-32
3040 FOR I=0 TO 7:FOR J=0 TO 7:F(I,J)=0:NEXT J:NEXT I
3050 FOR I=0 TO 7:A=PEEK(57344+I+Z*8):S(I)=A
3060 IF A>127 THEN A=A-128:F(0,I)=1
3070 IF A>63 THEN A=A-64:F(1,I)=1
3080 IF A>31 THEN A=A-32:F(2,I)=1
3090 IF A>15 THEN A=A-16:F(3,I)=1
3100 IF A>7 THEN A=A-8:F(4,I)=1
3110 IF A>3 THEN A=A-4:F(5,I)=1
3120 IF A>1 THEN A=A-2:F(6,I)=1
3130 IF A>0 THEN F(7,I)=1
3140 NEXT I
3200 FOR J=0 TO 7
3210 FOR I=0 TO 7
3220 IF F(I,J)=1 THEN POSITION 15+I,7+J:? CHR$(160)
3230 NEXT I
3240 NEXT J
3250 FOR I=0 TO 7:POSITION 26,7+I:? S(I);"  ":NEXT I
3260 IF BY=76 THEN POP :GOTO 100
3270 RETURN
5000 SOUND 0,10,6,2:CLOSE #1
5010 ZZ=ZN+Z*SW
5020 ? #2;ZZ;" DATA ";S(0);",";S(1);",";S(2);",";S(3)
;",";S(4);",";S(5);",";S(6);",";S(7);C$
5030 SOUND 0,14,6,2:FOR I=0 TO 127
5040 IF U(I)=Z THEN V=V+1:U(I)=-1
5050 NEXT I
5060 IF V=128 THEN POP :CLOSE #2:GOTO 6000
5070 SOUND 0,0,0,0:POP :GOTO 20
6000 ? CHR$(125):? :? "         ZEICHENSATZ-GENERATOR
"
```

```
6010 ? "_____________________________________":? :
?
6020 ? " ALLE 128 ZEICHEN SIND ERFASST":?
6030 ? " SIE FINDEN DIE DATEI AUF DER DISKETTE":?
6040 ? " MIT: LIST";CHR$(34);DAT$;CHR$(34):? :? :?
6050 ? " WOLLEN SIE NOCH EINEN ZEICHENSATZ":?
6060 ? " GESTALTEN?     <J> / <N>"
6070 OPEN #1,4,0,"K:":GET #1,T:CLOSE #1
6080 IF T=74 THEN 10
6090 IF T=78 THEN ? CHR$(125):END
6100 GOTO 6070
```

10: Das Programm ist komplett in BASIC geschrieben. Um es wenigstens etwas flinker zu machen, stehen die inaktiven Programmteile in den tieferen Zeilen. Die hier angesprungene Subroutine bringt den Vorspann auf die Scheibe.

1000 bis 1070: erklären das Programm.

1080: Die vielen DIMensionierungen benötigen etwas Zeit. Da der Benutzer erst einmal den Bildschirminhalt lesen wird, liegen sie an dieser Stelle im Programm. D.h. während der Benutzer liest, arbeitet der Rechner weiter. Falls die START-Taste gedrückt wird, bevor die DIMensionierungen erledigt sind, tritt eine kleine Verzögerung auf, bis der nächste Bildschirminhalt erscheint.

1090: Abfrage der START-Taste.

1100 bis 1150: Weitere Erläuterungen zum Programm.

1160: Der Benutzer soll jetzt einen Namen für die Datei eingeben, in der die Daten des zu gestaltenden Zeichensatzes erfaßt werden. Weil eine Benutzer-Eingabe falsch sein kann, wird die TRAP-Falle aufgestellt.

1170: Mit dem eingegebenen Dateinamen wird ein Schreib-Kanal zum Diskettenlaufwerk 1 eröffnet. Wurde der Dateiname in einer nicht zulässigen Form eingegeben, führt der OPEN-Befehl zu einem ERROR-, der durch den TRAP zur Zeile 1180 verzweigt.

1200: Auf der nächsten Bildschirmseite fragt das Programm nach einer Zeilennummer für die erste DATA-Zeile, in der die neuen Zeichensatz-Bytes aufgehoben und in das benannte File geschrieben werden.

1300: Außerdem wird danach gefragt, um welchen Wert die folgenden Zeilennummern steigen sollen. (Um es nicht zu vergessen, sei hier schon erwähnt, daß Sie sinnvollerweise als erste Zeilennummer 32637 und als Schrittweite 1 eingeben.

Sie werden schon noch erfahren, warum.)

1350: Schnell noch eine DIMensionierung, bevor es mit RETURN zurückgeht zu Zeile

20: Aber hier wird sogleich das nächste Unterprogramm angesprungen.

2000 bis 2120: bauen die Bildschirm-Grafik auf.

2130: ist nur von Interesse, wenn irgendwann einmal die Taste L gedrückt werden wird (davon später).

2140: Zurück nach Zeile

30: Hier erklingt ein Kontrollton. Erst wenn in der Adresse 53279 der Wert 5 steht, also wenn die Taste SELECT gedrückt wurde, gelangt das Programm aus dieser Zeile heraus und in

40: wird der Ton ausgeschaltet und nach 3000 gesprungen.

3000: Die Tastatur wird abgefragt.

3010: Falls der Benutzer ein inverses Zeichen aufrufen sollte, wird hier der Wert 128 abgezogen, denn inverse Zeichen gibt es im Zeichensatz bekanntlich nicht.

3020 und 3030: Die doppelt-indizierte Variable wird auf 0 gesetzt.

3050 bis 3140: Die acht Daten-Byte des aufgerufenen Zeichens werden aus dem Standard-Zeichensatz gelesen. In der Variablen S(n) wird der Dezimalwert des Byte vermerkt, dann wird es in seine Bits aufgespalten.

3200 bis 3240: Die acht mal acht Bit aus F(n,n) stellen das Zeichen auf dem Bildschirm dar. Ist ein Bit 1, wird ein inverses Leerzeichen (CHR$(160)) auf den Bildschirm gebracht. Das Bit-Muster des aufgerufenen Zeichens wird durch Hintergrund (schwarz) und ausgefüllte Schreibstellen (weiß) dargestellt.

3250: schreibt die Dezimalwerte der einzelnen Byte rechts neben die Grafik.

3270: führt zurück zu

100: wo erst einmal die Tongeneratoren 0 und 1 abgeschaltet werden.

110: eröffnet einen Lese-Kanal zur Tastatur. Der Wert der gedrückten Taste wird gelesen und ein Kontrollton erklingt.

120: Falls die Taste S gedrückt wurde, werden die Daten-Byte des gerade auf dem Bildschirm-Display sichtbaren Zei-

chens als DATA-Zeile in das bereits benannte File geschrieben. Das geschieht in der Subroutine bei 5000.

130: Wenn die Taste L gedrückt wurde, wird das im Display gezeigte Zeichen gelöscht. Alle acht Byte werden auf 0 gesetzt. Dieses Kommando löst einige gewagte Sprünge aus. Zuerst geht es nach 2040. Dort wird das leere Display auf die Mattscheibe gePRINTet und die acht Byte werden auf 0 gesetzt. In 2130 wird die Subroutine verlassen. Der POP-Befehl sorgt dafür, daß es trotzdem kein Durcheinander in der CPU gibt. Es geht dann weiter in 3250, wo die neuen Werte der acht Byte, die gerade auf 0 gesetzt wurden, neben das Display auf den Bildschirm geschrieben werden. 3260 führt wieder mit POP aus der Subroutine heraus und nach 100. Der Vorgang 'Löschen' ist beendet.

150: Wenn weder L noch S angeordnet wurde, dann wird jetzt eine zweite Eingabe erwartet. Beide Eingaben zusammen bezeichnen eine Stelle im Display, einen von acht mal acht Bildpunkten, aus denen sich das zu gestaltende Zeichen zusammensetzen soll. Ein zweiter Kontrollton registriert die Eingabe.

160 bis 190: Nur Eingaben von A (ATASCII 65) bis H (72) und 0 (48) bis 7 (55) werden verarbeitet.

200: Die beiden Eingaben bestimmen wie Koordinaten einen Punkt im Display. In dieser Zeile wird die zugehörige Speicherzelle des Bildschirmspeichers abgefragt.

210: Steht an der angesprochenen Stelle ein 'Punkt' (inverses Leerzeichen, interner Code 128), dann wird ein Leerzeichen (interner Code 0) an die Stelle gebracht

220: und umgekehrt.

230: Dann wird die Zeile, in der die Veränderung stattgefunden hat, abgefragt, der Dezimalwert des neuen Bit-Musters aktualisiert und an die entsprechende POS.ition auf den Bildschirm geschrieben. Dann geht's zurück nach 100.

5000: Hier wird der Acht-Byte-Datensatz eines gestalteten Zeichens als DATA-Zeile auf die Diskette geschrieben. Dazu gibt es wieder einen Kontrollton und der noch offene Datenkanals zur Tastatur wird erst einmal geschlossen. (Detailliertere Informationen über das Arbeiten mit der Disketten-Station finden Sie im Kapitel 5.)

5010: rechnet die Zeilennummer ZZ für die DATA-Zeile aus. ZN ist die vom Benutzer festgelegte erste DATA-Zeilennummer. Z ist der interne Code des aufgerufenen Zeichens, das in-

zwischen neu definiert wurde, und SW ist die ebenfalls vom Benutzer bestimmte Schrittweite der Zeilennummern. Jedes neu gestaltete Zeichen bekommt hier also eine Zeilennummer, die seinem Offset im Zeichensatz entspricht. So liegen die neuen Zeichen gleich in der richtigen Reihenfolge.

5020: schreibt die DATA-Zeile auf die Diskette.

5030 bis 5060: zählen mit, wie viele Zeichen schon neu gestaltet wurden. Die Variable U(n) merkt sich, welches Zeichen schon umgestaltet worden ist, V zählt, wie viele (verschiedene) Zeichen bereits umgestaltet sind. Wenn der komplette Zeichensatz neu entworfen ist, wird das Programm in den Zeilen

6000 bis 6100: beendet.

Als nächstes müssen wir nun dem Computer klar machen, daß er seinen Standard-Zeichensatz mal vergessen und dafür unsere geniale Neuschöpfung verwenden soll. Diese Aufgabe besorgt das folgende kurze Programm:

```
0 REM TYPWEX64.BI2
1 GOSUB 32631
32631 DIM MSP$(80):POKE 756,224:MND=PEEK(106)-4:CST=M
ND*256:POKE 106,MND:GRAPHICS 0
32632 FOR X=1 TO 40:READ ACHR:MSP$(X,X)=CHR$(ACHR):NE
XT X
32633 DATA 104,104,133,213,104,133,212,104,133,215,10
4,133,214,152,72,138,72,162,4,160,0,177,212,145,214
32634 DATA 200,208,249,230,213,230,215,202,208,240,10
4,170,104,168,96
32635 I=USR(ADR(MSP$),224*256,CST)
32636 FOR X=1 TO 63:FOR I=0 TO 7:READ TK:POKE CST+X*8
+I,TK:NEXT I:NEXT X
32767 POKE 756,MND:RETURN
```

1: Das Unterprogramm wird im tiefsten Zeilenbereich abgelegt, damit davor möglichst viel Platz für das eigentliche Programm bleibt, das die neu gestalteten Zeichen nutzen soll.

32631: Das Maschinensprache-Programm wird von BASIC als String behandelt. Der muß natürlich DIMensioniert werden. Adresse 756 ist der Zeiger auf die Zeichenbasis. Multipliziert man den Wert in dieser Adresse mit 256, erhält man die Start-Adresse des Zeichensatzes. Adresse 106 ist RAMTOP, die höchste Adresse des noch verfügbaren RAM-Speichers. Der Wert in dieser Zelle gibt den Speicherplatz in Seiten (pages) an. Eine Speicher-Page umfaßt 256 Byte.

Durch Änderung des Vektors in Adresse 106 wird ein geschützter Speicherbereich geschaffen, da das Betriebssystem den RAM-Bereich immer unterhalb von RAMTOP belegt.

Ein kompletter Zeichensatz benötigt1 kB (=4 pages). PEEK (106) schaut nach, welchen Wert der Vektor momentan hat, dann wird 4 für vier Speicherseiten abgezogen und der neue Wert nach 106 gePOKEt.

Wenn trotz dieser Reservierung Störungen auftreten (bestimmte Operationen können einen Speicherbereich oberhalb von RAMTOP löschen), dann reservieren Sie acht Pages. Dann gibt es mit Sicherheit keine Probleme.

Die Start-Adresse des neu definierten Character-Set ergibt MND*256.

32632: arbeitet das Maschinensprache-Programm ab, das in Form von DATA in Zeile

32633: abgelegt ist.

32635: ruft das Maschinensprache-Programm auf und

32636: liest die Daten-Byte des neuen Zeichensatzes, die in den folgenden DATA-Zeilen abgelegt sind, und schreibt sie, beginnend mit der Start-Adresse CST, in die folgenden Zellen.

Das erste Zeichen des Character-Set verwendet der Rechner als Leerzeichen (egal wie wir es gestalten!). In den meisten Fällen ist es sinnvoll, hier auch ein Leerzeichen abzulegen, damit der Bildschirm 'sauber' bleibt. Für den Rechner ist die Mattscheibe nämlich voller Leerzeichen. Wird das Leerzeichen vom Benutzer mit irgendeinem Bit-Muster gestaltet, so ist der "leere" Bildschirm mit diesem Bit-Muster gefüllt.

Der Kopiervorgang für den neuen Zeichensatz beginnt deshalb gleich mit dem Zeichen 1, so daß die ersten acht Byte im Speicherbereich des neuen Character-Set leer bleiben. Die ersten acht DATA bilden also (nach dem nullten) das erste Zeichen. Wenn Sie das Umkopieren mit dem Zeichen 0 beginnen lassen, dann sollten die ersten acht DATA Nullen sein.

In diesem Beispiel werden nur 63 Zeichen umkopiert. Wenn Sie einen vollständigen Zeichensatz erarbeitet haben, muß die FOR-NEXT-Schleife von 1 TO 127 laufen.

32637: Ab hier ist Platz für die DATA-Zeilen.

32767: Zum Schluß wird dem Rechner gesagt, bei welcher Adresse jetzt der Zeichensatz für ihn liegen soll. Wenn der Zeiger in Adresse 756 nicht verändert wird, dann weist er

mit dem Wert 224 (224*256=57344) auf die Start-Adresse des Standard-Zeichensatzes.

Jeder GRAPHICS-Befehl (und jedes RESET) erneuert den Default- (Einschalt-) Wert in dieser Adresse. Das bedeutet, nach jedem GR.-Befehl muß in 106 die Start-Adresse des frei definierten Character-Set erneuert werden.

Natürlich ist es auch möglich, mehrere Zeichensätze im Speicher abzulegen und durch UmPOKEn des Registers 106 von einer Schrift- oder Zeichenart auf die andere umzuschalten. Auf dem Bildschirm kann aber immer nur der aktuelle Zeichensatz dargestellt werden. Das Mischen verschiedener Zeichensätze ist nicht möglich, da der Rechner im Bildschirmspeicher den ATASCII-Wert erfaßt, nicht aber die Bit-Muster der Zeichen. Werden die Bit-Muster geändert, erfährt jeder ATASCII-Wert eine neue Zuordnung von Bit-Mustern.

Es wäre aber denkbar, mit dem Standard-Character-Set einen Text zu schreiben, der für den Uneingeweihten völlig sinnlos erscheint. Erst durch Umschalten auf einen frei definierten Zeichensatz wird jedem ATASCII-Wert ein neues Zeichen zugeordnet und damit der eigentliche Sinn des verschlüsselten Textes in Klarschrift lesbar. Das ergäbe eine sehr einfache Form elektronischer Verschlüsselung. Nur Empfänger, die den gleichen Zeichensatz laden können, bekommen die Nachricht lesbar auf den Bildschirm. Allerdings ist auf diese Weise nur eine einfache Zeichen/Zeichen-Zuweisung möglich, die jeder Geheimschrift-Experte in wenigen Minuten lösen wird.

Nach dem RETURN aus Zeile 32767 läuft das eigentliche BASIC-Programm, das in Zeile 2 beginnen kann. Setzen Sie in Zeile 32636 eine 127 ein und fügen eine Zeile 10 mit einem END ein. Laden Sie dann die Programme TYPWEX64.B12 und ARCHIV.DAT in den Rechner. Nach einem RUN dauert es einen kleinen Moment, bis der neue Zeichensatz geladen ist. Dann meldet sich der Computer mit READY. Das aber schon in der neuen Schrifttype. Sie können jetzt das Programm im Speicher löschen und ein neues Programm schreiben oder sonst alles machen, was mit dem Gerät möglich ist. Solange kein RESET oder GRAPHICS-Befehl ergeht, können Sie sich an der neuen Schrift erfreuen.

Der veränderte Zeichensatz ist auch in den Grafik-Modi 2 und 3 wirksam. Sie müssen nur nach dem GRAPHICS-Befehl den Zeiger in Adresse 756 erneuern.

Da die Zeichen auf dem Bildschirm in GR.2 doppelt so groß

und farbig sind, bieten frei definierte Zeichensätze hier besondere Spielmöglichkeiten. In GR.2 stehen nur 64 Zeichen zur Verfügung. Die übrigen dreimal 64 ATASCII-Werte rufen die gleichen Zeichen auf, beziehen aber die Farbe dafür aus wechselnden Farbregistern.

Im folgenden Zeichensatz ist jedes einzelne Zeichen als Bild eines kleinen Trickfilms gestaltet. Acht mal acht Punkte lassen zwar nicht viel Gestaltungsfreiraum, aber es sind doch eine Menge witziger Sachen damit möglich.

Natürlich könnte jedes Filmbild aus vier Zeichen zusammengesetzt werden, dann könnten aber nur insgesamt sechzehn verschiedene Bilder gezeigt werden. Allerdings könnte nach diesen sechzehn Bildern der Zeiger für den Zeichensatz (Adresse 106) auf einen Speicherbereich umgePOKEt werden, wo sich ein weiterer Zeichensatz befindet, der weitere sechzehn Bilder ermöglicht usw., bis der Speicher platzt.

```
0 REM TRICKTYP.BI2
32637 DATA 128,0,0,0,0,0,0,0
32638 DATA 192,192,0,0,0,0,0,0
32639 DATA 64,160,64,0,0,0,0,0
32640 DATA 96,144,144,96,0,0,0,0
32641 DATA 112,136,136,136,112,0,0,0
32642 DATA 120,132,132,132,132,120,0,0
32643 DATA 56,68,130,130,130,68,56,0
32644 DATA 60,66,129,129,129,129,66,60
32645 DATA 0,28,34,65,65,65,34,28
32646 DATA 0,0,30,33,33,33,33,30
32647 DATA 0,0,0,14,17,17,17,14
32648 DATA 0,0,0,0,6,9,9,6
32649 DATA 0,0,0,0,0,7,5,7
32650 DATA 0,0,0,0,0,0,3,3
32651 DATA 0,0,0,0,0,0,0,1
32652 DATA 0,0,0,0,0,0,0,0
32653 DATA 128,0,0,0,0,0,0,0
32654 DATA 0,64,0,0,0,0,0,0
32655 DATA 0,0,32,0,0,0,0,0
32656 DATA 0,0,0,16,0,0,0,0
32657 DATA 0,0,0,0,8,0,0,0
32658 DATA 0,0,0,0,0,4,0,0
32659 DATA 0,0,0,0,0,0,2,0
32660 DATA 0,0,0,0,0,0,0,1
32661 DATA 128,0,0,0,0,0,0,0
32662 DATA 0,64,0,0,0,0,0,0
32663 DATA 0,32,64,0,0,0,0,0
32664 DATA 0,16,0,64,0,0,0,0
```

```
32665 DATA 0,8,0,0,32,0,0,0
32666 DATA 0,4,0,0,0,32,0,0
32667 DATA 0,2,0,0,0,0,16,0
32668 DATA 0,1,0,0,0,0,0,4
32669 DATA 2,0,2,0,0,0,12,0
32670 DATA 0,4,0,4,0,10,0,0
32671 DATA 0,8,0,0,9,0,0,0
32672 DATA 0,16,0,22,0,20,0,0
32673 DATA 0,32,34,0,4,0,34,0
32674 DATA 0,64,4,0,0,8,0,65
32675 DATA 160,0,168,0,16,0,130,0
32676 DATA 0,64,16,112,0,64,2,0
32677 DATA 0,96,64,32,32,4,0,0
32678 DATA 160,0,160,112,68,80,0,0
32679 DATA 16,72,80,56,2,168,0,0
32680 DATA 0,44,148,72,4,2,184,0
32681 DATA 0,102,102,0,0,66,60,0
32682 DATA 0,102,66,0,0,126,0,0
32683 DATA 0,36,66,0,0,60,66,0
32684 DATA 0,0,66,0,0,52,66,0
32685 DATA 0,36,66,0,0,36,66,0
32686 DATA 0,66,36,0,0,36,66,0
32687 DATA 0,66,0,24,24,0,66,0
32688 DATA 0,0,36,0,0,36,0,0
32689 DATA 0,0,0,24,24,0,0,0
32690 DATA 0,0,24,36,36,24,0,0
32691 DATA 0,24,24,102,102,24,24,0
32692 DATA 24,24,0,195,195,0,24,24
32693 DATA 36,24,129,66,66,129,24,36
32694 DATA 36,36,219,36,36,219,36,36
32695 DATA 66,165,90,36,36,90,165,66
32696 DATA 32,102,91,36,36,218,102,4
32697 DATA 32,82,61,38,100,188,74,4
32698 DATA 48,48,27,39,228,216,12,12
32699 DATA 24,24,24,231,231,24,24,24
32700 DATA 24,36,90,165,165,90,36,24
```

Diese DATA-Zeilen wurden mit dem Programm TYPGENE.BI2 erzeugt. Mit dem folgenden kurzen Programm lernen die Trickfilm-Bilder laufen:

```
0 REM TRICKPRT.BI2
10 GRAPHICS 18:POKE 712,0:POKE 756,MND
20 FOR C=32 TO 95
30 PLOT 10,5:COLOR C
40 FOR W=0 TO 400:NEXT W
50 NEXT C
60 GOTO 20
```

10: ruft den Grafik-Modus auf und restauriert den Vektor in Adresse 756.

20: geht die ATASCII-Werte 32 bis 95 durch, das sind alle Werte, die in GRAPHICS 2 aufgerufen werden können und ihre Farbe aus Register 708 beziehen.

30: schreibt das aufgerufene Zeichen an die Bildschirmposition 10,5 und

40: legt eine Pause ein, damit unser Auge mitkommt.

Wenn Sie nachzählen wollen, es stehen DATA für 64 neu gestaltete Zeichen zur Verfügung. Das Programm TYPWEX64.BI2 lädt aber, wie besprochen, nur 63 davon, denn das nullte Zeichen bleibt als Leerzeichen erhalten. Wenn Sie die FOR-NEXT-Schleife auf 0 TO 63 ändern, dann können Sie sehen, was es bedeutet, kein Leerzeichen zu haben. Der Trickfilm ist dann zwar um ein Bild länger, aber der ganze übrige Bildschirm ist voll von gestalteten "Leerzeichen".

3.2.3 Character Dump

Wenn Aufbau und Lage des Zeichensatz-Speichers bekannt sind, dann ist es kein Problem mehr, die Daten eines bestimmten Zeichens zu lesen und in irgendeiner Weise weiter zu verarbeiten. So ist es z.B. möglich, die ATARI-Standard-Zeichen in ihren original Bit-Mustern mit dem Nadeldrucker auf Papier zu bringen. Schwierigkeiten bereitet höchstens die Kommunikation zwischen beiden Geräten. Im Kapitel 4 wird darüber ausführlicher zu reden sein.

Das markanteste Problem bei der Koordination von Rechner und Drucker ist die Tatsache, daß der Computer die Bit-Muster der Zeichen Zeilenweise zu Dezimalwerten aufrechnet, der Drucker aber spaltenweise arbeitet.

Es gibt verschiedene Wege, dieses Problem zu lösen. Am simpelsten ist es, dem Drucker die Daten aus dem Computer so zu senden, wie sie dort liegen. Die Zeichen werden dann um 90° gedreht auf das Papier gedruckt und wenn sie nicht spiegelverkehrt erscheinen sollen, müssen die Daten rückwärts verarbeitet werden. Wie ein solches Programm im Prinzip aufgebaut ist, hat Ihnen DEFCHRPR.BI2 bereits gezeigt. Sollen die Zeichen aber in richtiger Orientierung auf dem Blatt enden, dann ist ein etwas aufwendigeres Programm notwendig, das die zeilenweisen Daten-Byte aus dem Zeichensatz-Speicher des Rechners in Bits zerlegt und spaltenweise neu zu Daten-Bytes zusammenrechnet, die dem Drucker passen:

```
0 REM CHRPRT.BI2
100 LPRINT CHR$(27);CHR$(68);CHR$(4);CHR$(7);CHR$(10)
;CHR$(13);CHR$(16);CHR$(19);CHR$(22);CHR$(25);CHR$(0)
;
110 Q=6
200 DIM X(7,7)
210 FOR INV=0 TO 1
220 FOR ATA=0 TO 127
230 INC=ATA
240 IF ATA<32 THEN INC=ATA+64:GOTO 260
250 IF ATA<96 THEN INC=ATA-32:GOTO 260
260 FOR M=0 TO 7:FOR N=0 TO 7:X(M,N)=0:NEXT N:NEXT M
290 FOR I=0 TO 7
300 A=PEEK(57344+I+INC*8)
310 IF A>127 THEN A=A-128:X(0,I)=1
320 IF A>63 THEN A=A-64:X(1,I)=1
330 IF A>31 THEN A=A-32:X(2,I)=1
340 IF A>15 THEN A=A-16:X(3,I)=1
350 IF A>7 THEN A=A-8:X(4,I)=1
360 IF A>3 THEN A=A-4:X(5,I)=1
370 IF A>1 THEN A=A-2:X(6,I)=1
380 IF A>0 THEN X(7,I)=1
390 NEXT I
400 A=X(0,0)*128+X(0,1)*64+X(0,2)*32+X(0,3)*16+X(0,4)
*8+X(0,5)*4+X(0,6)*2+X(0,7)
410 A=ABS(INV*255-A)
450 B=X(1,0)*128+X(1,1)*64+X(1,2)*32+X(1,3)*16+X(1,4)
*8+X(1,5)*4+X(1,6)*2+X(1,7)
460 B=ABS(INV*255-B)
500 C=X(2,0)*128+X(2,1)*64+X(2,2)*32+X(2,3)*16+X(2,4)
*8+X(2,5)*4+X(2,6)*2+X(2,7)
510 C=ABS(INV*255-C)
550 D=X(3,0)*128+X(3,1)*64+X(3,2)*32+X(3,3)*16+X(3,4)
*8+X(3,5)*4+X(3,6)*2+X(3,7)
560 D=ABS(INV*255-D)
600 E=X(4,0)*128+X(4,1)*64+X(4,2)*32+X(4,3)*16+X(4,4)
*8+X(4,5)*4+X(4,6)*2+X(4,7)
610 E=ABS(INV*255-E)
650 F=X(5,0)*128+X(5,1)*64+X(5,2)*32+X(5,3)*16+X(5,4)
*8+X(5,5)*4+X(5,6)*2+X(5,7)
660 F=ABS(INV*255-F)
700 G=X(6,0)*128+X(6,1)*64+X(6,2)*32+X(6,3)*16+X(6,4)
*8+X(6,5)*4+X(6,6)*2+X(6,7)
710 G=ABS(INV*255-G)
750 H=X(7,0)*128+X(7,1)*64+X(7,2)*32+X(7,3)*16+X(7,4)
*8+X(7,5)*4+X(7,6)*2+X(7,7)
760 H=ABS(INV*255-H)
```

```
800 OPEN #1,8,0,"P:"
810 Q=Q+1:IF Q=7 THEN PUT #1,10:Q=-1:GOTO 830
820 FOR T=0 TO Q:PUT #1,9:NEXT T
830 PUT #1,27
840 PUT #1,75
850 PUT #1,8
860 PUT #1,0
870 PUT #1,A
880 PUT #1,B
890 PUT #1,C
900 PUT #1,D
910 PUT #1,E
920 PUT #1,F
930 PUT #1,G
940 PUT #1,H
950 CLOSE #1
960 NEXT ATA
970 NEXT INV
```

100: setzt die Tabulator-Marken des Druckers. Das Format dieser Anweisung ist auf den radix-10 von star abgestellt. Falls Sie einen anderen Drucker haben sollten, müssen Sie diese Zeile entsprechend ändern. (Es werden übrigens nur sechs TAB-Marken benötigt.)

110: Die Variable Q bekommt einen Anfangswert, der dafür sorgt, daß vor dem Ausdruck des ersten Zeichens in Zeile 810 bereits ein Zeilenvorschub an den Drucker gesendet wird. Der automatische Zeilenvorschub bei Empfang des Signals CR (carriage return = RETURN-Taste) muß unterdrückt sein (star: DIP-switch C-4: OFF).

210: Das Programm muß zweimal abgearbeitet werden, um einen normalen und einen inversen Zeichensatz auszudrucken.

220: Die folgende Schleife arbeitet 128 Werte ab.

240 und 250: verwandeln die ATASCII-Werte in internen Code.

290 bis 390: Die acht Daten-Byte des bearbeiteten Zeichens werden gelesen und in ihr Bit-Muster aufgespalten. Die doppelt-indizierte Variable X(n,n) nimmt die acht mal acht Bit-Werte auf.

400 bis 760: Die in X(n,n) erfaßten Bit-Muster werden hier spaltenweise zu Dezimalwerten aufgerechnet. Die Variable INV bewirkt, daß die Zeichen normal oder invers gedruckt werden. Ist INV=0, dann bleibt der Multiplikator 255 in der Klammer ohne Wirkung und das Minuszeichen wird durch die ABS-Funktion unschädlich gemacht. Ist aber INV=1, dann

wird von 255 der Dezimalwert abgezogen und das bewirkt ein Invertieren des Bit-Musters.

800: Der Schreib-Kanal zum Drucker wird eröffnet.

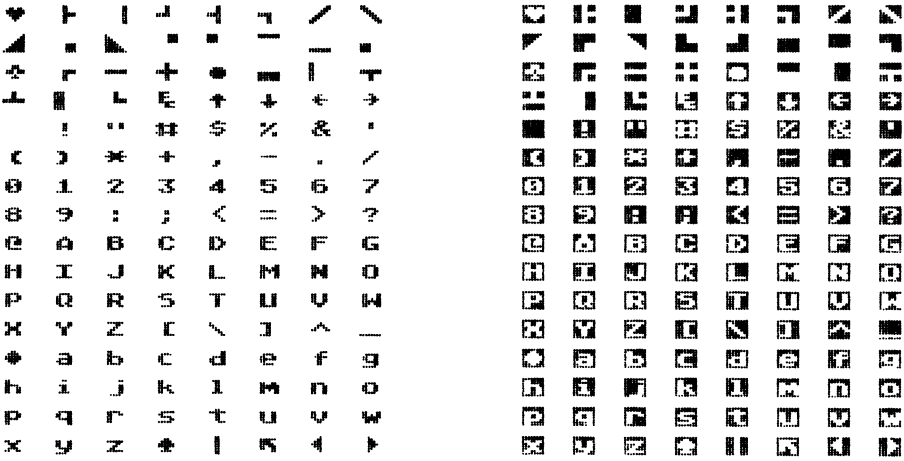
810: Da jeweils acht Zeichen nebeneinander gedruckt werden sollen, wird nur nach jedem achten Zeichen ein LF (line feed, Zeilenvorschub, ASCII-Wert 10) an den Drucker geschickt.

820: besorgt die Tabulator-Sprünge.

830 bis 860: bereiten den Drucker auf Dot-Grafik-Daten vor. Diese Zeile müssen Sie wieder entsprechend der Betriebsanleitung Ihres Druckers umschreiben.

870 bis 940: senden acht Bit-Muster an den Drucker.

Und so sieht das Ergebnis aus:



3.3 Bildschirmspeicher

Der Bildschirmspeicher enthält die Bild-Daten, mit denen die Bild-Struktur der Display-Liste gefüllt wird. Das Betriebssystem richtet im unteren RAM-Bereich den Bildschirmspeicher so ein, daß er oberhalb der Adresse 40960 endet. Ergeht durch den Anwender ein GRAPHICS-Befehl, wird entsprechend dem Speicherbedarf der aufgerufenen Betriebsart Platz reserviert. Da immer ein Grafik-Modus eingeschaltet ist, ohne besonderen Befehl ist das GR.0, ist auch immer ein Bildschirmspeicher angelegt. Der Bildschirmspeicher reduziert also die etwa 32 kByte RAM, die dem Anwender zur Verfügung stehen.

Der Vektor in den Adresse 88 (LO) und 89 (HI) zeigt auf die Start-Adresse des Bildschirmspeichers. In GR.0 ist das 40000. Der Bildschirminhalt von GR.0 belegt 960 Byte, so daß der Bildschirmspeicher in 40959 (unterhalb von 40960) endet.

Wird ein Grafik-Modus mit Textfenster aufgerufen, so wird ein Speicherbereich reserviert, der ausreicht, um den gesamten Bildschirm in der angesprochenen Grafik-Auflösung zu bedienen. Dahinter wird zusätzlich Speicherplatz für die vier Zeilen GR.0 des Textfensters belegt. Deshalb ist es möglich, in ein Grafikfenster das Textfenster ein- und wieder auszublenden. D.h. der Bildschirmspeicher für das Textfenster beginnt immer in Adresse 40800, gleichgültig in welcher Auflösung der übrige Bildschirm betrieben wird.

Unmittelbar vor dem Bildschirmspeicher ist die Display-Liste abgelegt.

3.3.1 Aufbau der SM

Im Bildschirmspeicher (SM = screen memory) sind die Daten aufgereiht, die den Bildschirminhalt darstellen. Die Start-Adresse der SM enthält die Daten für die linke obere Ecke des Bildschirms. Es folgen die Bild-Daten zeilenweise. In der End-Adresse der SM befinden sich die Informationen für die rechte untere Ecke des Bildschirms.

Der Bildschirmspeicher wird durch den Befehl PLOT (oder PRINT) beschrieben. Der COLOR-Wert (oder der String-Inhalt) gibt an, was in die durch PLOT gefundene Speicherzelle geschrieben werden soll. Im Textbetrieb werden in den Speicherzellen die Daten-Byte des internen Codes der jeweiligen Zeichen abgelegt. In GR.0 entsprechen diese Werte den 256 möglichen Zeichen. In GR.1 und GR.2 ist jeweils nur ein

Viertel des Zeichenvorrates verfügbar, dafür aber in vier verschiedenen Farben, was die möglichen Dezimalwerte wieder auf 256 bringt.

Es ist vielleicht gut, noch einmal darauf hinzuweisen, daß jede Speicherzelle acht Bit zusammenfaßt und jedes dieser acht Bit nur den Zustand 0 oder 1 kennt. In den Speicherzellen befinden sich also nicht wirklich Dezimalwerte, vielmehr gibt das Betriebssystem den Inhalt von Adressen als Dezimalwerte aus, ein Entgegenkommen an den menschlichen Benutzer, der noch immer nur zehn Finger hat und deshalb vom sperrigen Dezimalsystem nicht loskommt. Der tatsächliche Inhalt einer Speicherzelle ist aber als achtstellige Binärzahl vorzustellen.

Dieses Bewußtsein ist hilfreich, um die Erfassung von Grafikpunkten in der SM zu verstehen. Es sind drei Gruppen von Grafik-Modi zu unterscheiden, die zweifarbigen, die vierfarbigen und die mehrfarbigen.

In einer zweifarbigen Betriebsart erfaßt jedes Bit einen Grafikpunkt. Dabei ist es völlig gleichgültig, wie groß dieser Grafikpunkt später auf der Bildfläche erscheint. Diese Informationen enthält die Display-Liste. In der SM wird nur der COLOR-Wert des zugehörigen Grafikpunktes gespeichert. Bei einer zweifarbigen Betriebsart gibt es den COLOR-Wert 0 (Hintergrund) und COLOR 1 (Grafikpunkt). Bits werden je nach Bildstruktur gesetzt oder nicht gesetzt und von dem Byte, in dem sie zusammenkommen, zu einem Dezimalwert (0 bis 255) zusammengefaßt, ganau wie es für den Zeichensatz schon erläutert wurde.

In der Betriebsart GR.4 liegen 80 Bildpunkte auf dem Schirm nebeneinander. Die Information von acht Punkten hat in einem Byte Platz. Also benötigt eine GRAPHICS-4-Zeile 10 (80/8) Byte Speicherplatz.

In einem vierfarbigen Modus werden für den COLOR-Wert eines Grafikpunktes zwei Bit benötigt. In diese beiden Bit wird der COLOR-Wert in binärer Form geschrieben. Also 0 0 für COLOR 0, 0 1 für COLOR 1, 1 0 für COLOR 2 und 1 1 für COLOR 3.

Bit	7	6	5	4	3	2	1	0
27	0	0	0	1	1	0	1	1
Dezimalwert	128	64	32	16	8	4	2	1
	CO.0		CO.1		CO.2		CO.3	

In jedem Speicher-Byte haben also die Informationen für vier Grafikpunkte Platz. Die Abbildung zeigt, wie vier in einer Zeile liegende Grafikpunkte mit den Farben 0,1,2,3 in einer Zelle der SM den Dezimalwert 27 verursachen. Würde in diesen Speicherplatz der Wert 25 geschrieben, würde der vierte Grafikpunkt die Farbe 1 annehmen.

Die Grafikpunkte auf der Mattscheibe schöpfen also aus zwei Quellen. Der Bildschirmspeicher gibt an, welcher Punkt in welchem COLOR-Wert dargestellt werden soll. Welcher Farbton damit gemeint sein soll, bestimmen die Farbregister. Im folgenden Programm können Sie die Farbtöne für die Farbregister bestimmen. Dann wird in jede Speicherzelle der SM der Wert 27 geschrieben. Er bewirkt, wie oben gezeigt wurde, daß vier Grafikpunkte nebeneinander mit den COLOR-Werten 0, 1, 2 und 3 auf dem Bildschirm erscheinen:

```
10 ? CHR$(125);" BITTE GEBEN SIE VIER NUMERISCHE"
20 ? :? " FARBWERTE VON 0 BIS 254 EIN"
30 ? :? " NUR GERADE ZAHLEN!"
40 ? :? " ERSTER WERT,  GRAFIKPUNKT 1:  ";
50 INPUT G1
60 ? :? " ZWEITER WERT, GRAFIKPUNKT 2:  ";
70 INPUT G2
80 ? :? " DRITTER WERT, GRAFIKPUNKT 3:  ";
90 INPUT G3
100 ? :? " VIERTER WERT, HINTERGRUND 0:  ";
110 INPUT H
200 GRAPHICS 19:POKE 708,G1:POKE 709,G2:POKE 710,G3:
POKE 712,H
210 SM=PEEK(88)+PEEK(89)*256
220 FOR J=0 TO 239
230 POKE SM+J,27
240 NEXT J
250 GOTO 250
```

10 bis 110: schreiben Werte in die Farbregister, auf die sich COLOR 0 bis 3 beziehen. Hier können Sie also beliebige Farbtöne auswählen. Mit dem Bildschirmspeicher hat das nichts zu tun.

200: Der Grafik-Modus wird eingeschaltet und die vom Benutzer eingegebenen Farbwerte werden in die entsprechenden Register geschrieben.

210: berechnet die Start-Adresse des Bildschirmspeichers.

220 bis 240: beschreiben die 240 Speicher-Bytes des Bildschirms mit 27. Ein Streifenmuster entsteht durch die ständige Wiederholung der gleichen Folge von COLOR-Werten.

Bei diesem Programm können Sie die Folge der vier COLOR-Werte, die zu einem Byte zusammengefaßt werden, selbst bestimmen:

```
0 REM SMPOKEGR.BI2
10 ? CHR$(125);" BITTE GEBEN SIE VIER COLOR-WERTE"
20 ? :? " VON 0 BIS 3 EIN"
30 ? :? "  ERSTER GRAFIKPUNKT:  ";
40 INPUT G0
50 ? :? " ZWEITER GRAFIKPUNKT:  ";
60 INPUT G1
70 ? :? " DRITTER GRAFIKPUNKT:  ";
80 INPUT G2
90 ? :? " VIERTER GRAFIKPUNKT:  ";
100 INPUT G3
200 GRAPHICS 19
210 SM=PEEK(88)+PEEK(89)*256
220 CC=G0*64+G1*16+G2*4+G3
230 FOR J=0 TO 239
240 POKE SM+J,CC
250 NEXT J
260 GOTO 260
```

10 bis 100: nehmen die COLOR-Werte für eine Folge von vier Grafikpunkten an.

210: berechnet die Start-Adresse des Bildschirmspeichers.

220: errechnet aus den vier COLOR-Werten den Dezimalwert, der in die Speicherzellen geschrieben werden muß, um die gewünschte Farbfolge von Grafikpunkten auf der Bildfläche zu bewirken. Der erste Grafikpunkt liegt in der Ecke oben links. Im ersten Daten-Byte belegt er die Bits 7 (dezimal 128) und Bit 6 (dezimal 64). Der COLOR-Wert des ersten Grafikpunktes muß also mit 64 multipliziert werden. Der zweite Grafikpunkt belegt Bits 5 (32) und 4 (16). Sein COLOR-Wert ist demnach mit 16 zu multiplizieren. Der dritte Punkt beschreibt Bits 3 (8) und 2 (4) und sein COLOR-Wert wird mit 4 malgenommen. Der COLOR-Wert des vierten Grafikpunktes wird in Bits 1 und 0 gespeichert. So errechnet sich aus den COLOR-Werten von vier Grafikpunkten ein Dezimalwert, der in eine Zelle des Bildschirmspeichers geschrieben, das gewünschte Bit-Muster erzeugt.

230 bis 250: schreiben mit diesem einen Wert wieder den gesamten Bildschirmspeicher voll.

Natürlich kann für jeden einzelnen Grafikpunkt ein beliebiger COLOR-Wert gewählt werden. Dann bekommt jede Zelle der SM einen individuellen Dezimalwert. Das folgende Programm er-

zeugt zufällige Dezimalwerte, die in die SM geschrieben zufällige Verteilungen von COLOR-Werten verursachen:

```
200 GRAPHICS 19:POKE 708,24:POKE 709,222:POKE 710,150
:POKE 712,0
210 SM=PEEK(88)+PEEK(89)*256
220 FOR J=0 TO 239
230 POKE SM+J,INT(RND(0)*256)
240 NEXT J
250 GOTO 250
```

Schließlich verfügt ATARI noch über drei Grafik-Modi, die mehr als vier verschiedene Farbtöne gleichzeitig verwalten können. Es handelt sich um GRAPHICS 9 bis 11. Hier stellt jeder Grafikpunkt vier Bit Information dar. Je zwei Grafikpunkte werden in einem Byte zusammengefaßt. Um den damit verbundenen extrem hohen Speicherbedarf zu verringern, ist ein Grafikpunkt auf der Mattscheibe vier Bildpunkte breit (aber nur eine Fernsehzeile hoch). Dadurch haben diese drei Betriebsarten einen gleichhohen Speicherbedarf wie GRAPHICS 8, wo ein Pixel nur einen Bildpunkt breit ist, dafür aber nur eine Farbe zur Verfügung steht. (Sie erinnern sich aus Abschnitt 1.1, daß die Modi 9, 10 und 11 die gleiche Display-List verwenden wie GR.8?).

Bit	7	6	5	4	3	2	1	0
27	0	0	0	1	1	0	1	1
Dezimalwert	128	64	32	16	8	4	2	1
	COLOR 1				COLOR 11			

Der COLOR-Wert des ersten Grafikpunktes wird in binärer Form in die oberen vier Bit geschrieben, der COLOR-Wert für das folgende Pixel in die unteren vier Bit. Daraus ergibt sich ein Dezimalwert als Speicherinhalt, der zwei verschiedene Grafikpunkte erzeugt. In der obigen Abbildung hat der erste Grafikpunkt den COLOR-Wert 1 (binär 0001), der zweite den COLOR-Wert 11 (binär 1011). Als Dezimalwert ergibt sich daraus 0+0+0+16+8+0+2+1=27.

In GR.9 ist jedem der möglichen sechzehn COLOR-Werte (0 bis 15) eine Helligkeitsstufe des für alle gleichen Farbtons zugeordnet:

```
0 REM GR9COLR.BI2
10 GRAPHICS 9
```

```
20 SM=PEEK(88)+PEEK(89)*256
30 FOR C=0 TO 15
40 FOR J=0 TO 191
50 POKE SM+J*40+C,C
60 NEXT J
70 NEXT C
80 FOR M=0 TO 15
90 POKE 712,M*16
100 FOR W=0 TO 200:NEXT W
110 NEXT M
120 GOTO 120
```

Das Programm schreibt sechzehn senkrechte Farbbalken auf den Bildschirm. Der erste entspricht der Hintergrund- und Randfarbe und hebt sich deshalb nicht ab. Es folgen Streifen mit wachsender Helligkeit. Da in Zeile 50 nur Dezimalwerte von 0 bis 15 gePOKEt werden, sind die oberen vier Bit jeder Speicherzelle auf 0 gesetzt, d.h. der erste Grafikpunkt hat immer den COLOR-Wert 0, der folgende den gePOKEten Wert. Deswegen ist jeder zweite Streifen schwarz (Farbe von COLOR 0). Wenn Sie in Zeile 50 statt C den Wert C*16 POKEn, werden die oberen vier Bit beschrieben. Mit C*16+C können Sie auch beiden Grafikpunkten einen wechselnden COLOR-Wert geben.

In den Zeilen 80 bis 110 wird der Wert im Farbregister gewechselt. Er bestimmt die Farbe für alle Grafikpunkte. Die Daten der SM bestimmen nur den Helligkeitswert.

In den Farbregistern (704 bis 712) bestimmen die oberen vier Bit den Farbton (0 bis 15) und die folgenden drei Bit die Helligkeit. Bit 0 wird nicht genutzt, weil nur acht Helligkeitsstufen verarbeitet werden (die geraden Werte von 0 bis 14). Um den Farbton zu wechseln, muß in Register 712 ein Vielfaches von 16 geschrieben werden.

Bei GR.11 sieht es ganz ähnlich aus. Nur wird hier der Helligkeitswert in Register 712 definiert und von COLOR werden die sechzehn verschiedenen Farbtöne aufgerufen (die dann alle die gleiche Helligkeit haben):

```
0 REM GR11COLR.BI2
10 GRAPHICS 11
20 SM=PEEK(88)+PEEK(89)*256
30 FOR C=0 TO 15
40 FOR J=0 TO 191
50 POKE SM+J*40+C,C
60 NEXT J
70 NEXT C
```

```
80 FOR M=0 TO 15
90 POKE 712,M
100 FOR W=0 TO 200:NEXT W
110 NEXT M
120 GOTO 120
```

Das Programm schreibt sechzehn senkrechte Streifen verschiedener Farbtöne auf den Bildschirm und wechselt dann die Helligkeitsstufen.

Etwas anders ist es bei GR.10. Hier können nur neun Farbtöne (incl. Hintergrund) aufgerufen werden. Einfach deshalb, weil alle diese Farbtöne beliebig definiert werden können, dafür aber nur neun Farbregister (704 bis 712) zur Verfügung stehen. Auch hier nimmt jede Speicherzelle die COLOR-Werte von zwei Grafikpunkten auf:

```
0 REM GR10COLR.BI2
10 GRAPHICS 10:POKE 704,4
20 SM=PEEK(88)+PEEK(89)*256
30 FOR C=0 TO 8
40 FOR J=0 TO 191
50 POKE SM+J*40+C,C
60 NEXT J
70 NEXT C
80 GOTO 80
```

Die Farbregister 704 bis 707, die außer für GR.10 nur bei der Player-Missile-Grafik verwendet werden, haben den Default-Wert 0. Deshalb kommen vier schwarze Streifen auf den Bildschirm. Die Register 708 bis 712 haben die Default-Werte 40 (Orange), 208 (Hellgrün), 48 (Mittelblau), 70 (Flieder) und 0 (Schwarz).

Den ganzen Farbreichtum des ATARI zeigt Ihnen diese kleine Spielerei:

```
0 REM MULTICOL.BI2
10 GRAPHICS 11
20 SM=PEEK(88)+PEEK(89)*256
30 FOR C=0 TO 239
40 FOR J=0 TO 40
50 POKE SM+C*40+J,C+16
60 NEXT J
70 NEXT C
80 GOTO 80
```

3.3.2 Schrift und Grafik mischen

Bei manchen Anwendungen mag es nicht befriedigen, im oberen Teil des Bildschirms Grafik zu zeigen und in dem Vier-Zeilen-Textfenster darunter schriftliche Informationen. Bei Spielen und besonders bei Schaubildern, möchte man oft Schriftzeichen innerhalb der Grafik zeigen.

Theoretisch ist das überhaupt kein Problem, denn aus den Punkten des Grafik-Modus kann man natürlich Schrift- und andere Zeichen zusammensetzen. Nur ist es ein abschreckend mühsamer Vorgang, jeden Buchstaben aus Dutzenden einzelner Punkte zusammenzuPLOTten.

Nun wurde aber im vorigen Abschnitt gezeigt, daß sich die Daten-Byte in den zweifarbigen Grafik-Modi genauso zusammensetzen wie die umgerechneten Bit-Muster aus dem Zeichensatz-Speicher. Wird also ein Dezimalwert aus dem Standard-Zeichensatz gelesen (PEEK) und in eine Speicherzelle der SM geschrieben (POKE), so entsteht das Bit-Muster des Zeichens auf dem Bildschirm in der Auflösung des gewählten Grafik-Modus. Da ein Schriftzeichen aus acht Daten-Byte besteht, müssen acht fortlaufende Zellen im Zeichensatz-Speicher gelesen werden.

Die im Zeichensatz hintereinander liegenden acht Byte sollen aber auf der Bildfläche untereinander liegen. Sie gehören also im Bildschirmspeicher nicht in fortlaufende Zellen, sondern müssen jeweils eine Zeile tiefer abgelegt werden. Es ist also nötig, auszurechnen, wie viele Byte eine Zeile des Bildschirms belegt, um dann die acht Byte des Zeichens in jeweils diesem Abstand hintereinander in der SM abzulegen.

Da die Grafik in einem Grafik-Modus sinnvollerweise durch die Befehle PLOT und DRAWTO aufgebaut wird, die leichter zu handhaben sind, als eine direkte Adressierung der SM, die Schriftzeichen aber nur direkt in den Bildschirmspeicher gePOKEt werden können, sei hier noch auf eine Feinheit hingewiesen.

Durch den Befehl PLOT (und DRAWTO) wird in einer zweifarbigen Betriebsart ein bestimmtes Bit in einem bestimmten Speicher-Byte gesetzt (im vierfarbigen Gang zwei). Dadurch erhöht sich der Gesamtwert des Byte um die Wertigkeit des Bit. Durch den Befehl PLOT werden also schon im Daten-Byte vorhandene Werte nicht gelöscht, PLOT verändert nur den Zustand einzelner Bits. Wird aber ein Dezimalwert in ein Speicher-Byte gePOKEt, so affektiert das natürlich alle Grafikpunkte, deren Information in diesem Byte versammelt ist.

Das bedeutet, wird durch PLOT ein Punkt auf dem Bildschirm erzeugt und ein Bit in einer Speicherzelle gesetzt und dann ein Dezimalwert in diese Speicherzelle geschrieben, um das Bit-Muster eines Zeichens auf die Mattscheibe zu bringen, dann wird das gesetzte Bit dadurch überschrieben und der Punkt auf dem Bildschirm verschwindet. Wird aber erst der Dezimalwert in die Speicherzelle gePOKEt und dann ein Punkt gePLOTtet, so erscheint er auch innerhalb des Bereichs von acht mal acht Punkten, der von den Schriftzeichen abgedeckt wird. Diesen Vorgang des Überschreibens können Sie beim folgenden Programm beobachten:

```
0 REM GR8CHRPR.BI2
10 DIM Z$(1)
20 GRAPHICS 8
30 ? "BITTE DAS ZEICHEN EINGEBEN, DAS AUF"
40 ? "DEN BILDSCHIRM GESETZT WERDEN SOLL"
50 ? "BITTE NUR ZIFFERN UND MAJUSKELN"
60 INPUT Z$
70 IF Z$="L" THEN COLOR 1:PLOT 0,0:DRAWTO 319,159:GO
TO 30
80 ? :? "BITTE GEBEN SIE DIE SPALTE (0 BIS 39)"
90 ? "EIN, IN DER DAS ZEICHEN STEHEN SOLL"
100 INPUT X
110 ? :? "BITTE GEBEN SIE DIE ZEILE (0 BIS 152)"
120 ? "EIN, IN DER DAS ZEICHEN STEHEN SOLL"
130 INPUT Y
200 SM=PEEK(88)+PEEK(89)*256
210 OFF=(ASC(Z$)-32)*8
220 FOR J=0 TO 7
230 POKE SM+X+Y*40+J*40,PEEK(57344+OFF+J)
240 NEXT J
250 GOTO 30
```

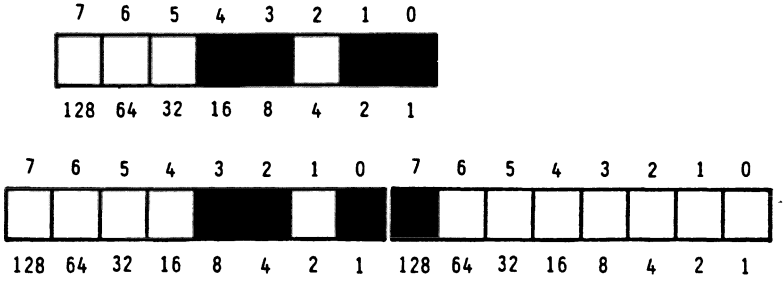
30 bis 60: Das Zeichen, das auf dem Bildschirm erscheinen soll, wird als String eingegeben. So braucht kein Datenkanal eröffnet zu werden, um die Tastatur abzufragen.

70: Wenn ein 'L' eingetastet wird, PLOTtet das Programm eine Linie von oben links nach unten rechts durch das Grafikfenster.

80 bis 100: Acht Pixel von GRAPHICS 8 sind in einem Byte zusammengefaßt. Die Daten einer Zeile verteilen sich also auf 40 Byte. In diese 40 Speicherzellen kann das Bit-Muster des aufgerufenen Zeichens geschrieben werden.

Natürlich ist es denkbar, die (Schrift-) Zeichen in der Waagerechten auch feiner zu positionieren, dann müßte aber

ein Daten-Byte des Zeichens auf zwei (nebeneinander liegende) Speicherplätze der SM verteilt werden. Dabei verändern sich die Wertigkeiten der einzelnen Bit. Bei der Verschiebung um einen einzelnen Bildpunkt nach rechts wird aus der oberen Speicherzelle das Bit 0 in die folgende Adresse verschoben und landet dort als Bit 7. Ein solches Fein-Verschieben in horizontaler Richtung wäre also mit sehr viel Rechenaufwand verbunden.



Das ursprüngliche Bit-Muster hat den Dezimalwert 27. Sollen die bewirkten Bildpunkte um eins nach rechts verschoben werden, erhält die erste Speicherzelle den Dezimalwert 13, die folgende, eine Stelle tiefer liegende, den Wert 128. Dabei wäre aber auch noch zu berücksichtigen, daß in der zweiten Adresse bereits ein Wert vorhanden ist, der natürlich durch diese Operation nicht gelöscht werden darf, während in die obere Zelle möglicherweise auch ein Bit von der noch davor liegenden Adresse hineingeschoben wird.

110 bis 130: In vertikaler Richtung kann das Zeichen beliebig positioniert werden. 159 Zeilen bilden das Grafik-Fenster von GRAPHICS 8. Die hier erfragte Position bezieht sich auf die Speicherzelle, in der das erste Byte des Zeichens abgelegt werden soll. Die übrigen sieben Byte liegen in den Zeilen darunter. Deshalb können Zeilen-Werte von 0 bis 152 (152+7=159) eingegeben werden.

200: Die SM-Start-Adresse.

210: errechnet den Offset. Da nur Großbuchstaben eingegeben werden sollen, genügt diese eine Formel. Sie ermittelt den ATASCII-Wert des Zeichens, das aus dem String entnommen wurde (ASC(Z$)), findet den internen Code durch Subtraktion von 32 und multipliziert diesen Wert mit 8.

220 bis 240: schreiben die acht Daten-Byte des aufgerufenen Zeichens in die entsprechenden Zellen des Bildschirmspeichers.

SM ist die Start-Adresse des Bildschirmspeichers, X die Spaltenposition. Die Zeilenposition Y muß mit 40 multipliziert werden, da die Speicherzellen für untereinander liegende Bildschirmpositionen 40 Byte auseinander liegen, oder anders gesagt, eine Bildschirmzeile belegt 40 Byte. Das gleiche gilt für J, da die acht Byte des Zeichens untereinander auf dem Bildschirm landen sollen. 57344 ist die Start-Adresse des Zeichensatzes, OFF der Offspring eines Zeichens und J liest die acht Byte des Bit-Musters.

Sie können mit diesem Programm beliebig viele Zeichen in das Grafik-Fenster von GR.8 schreiben. Beachten Sie, wie jedes Zeichen eine Fläche von acht mal acht Punkten belegt und dadurch andere Zeichen ganz oder teilweise löscht. Beachten Sie auch, wie die Diagonale beim Überschreiben mit Zeichen gelöscht wird, wie die Linie selbst aber in schon bestehende Zeichen hineingeschrieben wird.

Nach diesem Prinzip der direkten Adressierung werden auch die Schriftzeichen im folgenden Testbild eingefügt:

```
0 REM TESTBILD.BI2
10 DEG :DIM C$(40):POKE 82,0:? CHR$(125):? :? "    BI
TTE 40 TEST-ZEICHEN EINGEBEN:"
20 ? :? :? "(NUR MAJUSKELN + ZIFFERN /ATASCII 32-95)"
30 ? "_____________________________________":? :IN
PUT C$
40 GRAPHICS 24:COLOR 1:I0=PEEK(88)+PEEK(89)*256:Z=573
44
50 T=Z+52*8:E=Z+37*8:S=Z+51*8:B=Z+34*8:I=Z+41*8:L=Z+4
4*8:D=Z+36*8:KA=Z+8*8:C=Z+99*8:KZ=Z+9*8
60 Z1=Z+17*8:Z9=Z+25*8:Z8=Z+24*8:Z4=Z+20*8:K=Z+43*8:H
=Z+40*8:ZK=Z+12*8:CC=Z+35*8:ZZ=Z+58*8
100 FOR J=0 TO 7
110 POKE I0+816+J*40,PEEK(T+J)
120 POKE I0+817+J*40,PEEK(E+J)
130 POKE I0+818+J*40,PEEK(S+J)
140 POKE I0+819+J*40,PEEK(T+J)
150 POKE I0+820+J*40,PEEK(B+J)
160 POKE I0+821+J*40,PEEK(I+J)
170 POKE I0+822+J*40,PEEK(L+J)
180 POKE I0+823+J*40,PEEK(D+J)
200 POKE I0+2616+J*40,PEEK(KA+J)
210 POKE I0+2497+J*40,PEEK(C+J)
220 POKE I0+2458+J*40,PEEK(KZ+J)
230 POKE I0+2419+J*40,PEEK(Z+J)
240 POKE I0+2420+J*40,PEEK(Z1+J)
250 POKE I0+2461+J*40,PEEK(Z9+J)
```

```
260 POKE I0+2502+J*40,PEEK(Z8+J)
270 POKE I0+2623+J*40,PEEK(Z4+J)
300 POKE I0+4856+J*40,PEEK(K+J)
310 POKE I0+4977+J*40,PEEK(H+J)
320 POKE I0+5018+J*40,PEEK(K+J)
330 POKE I0+5059+J*40,PEEK(ZK+J)
340 POKE I0+5060+J*40,PEEK(Z+J)
350 POKE I0+5021+J*40,PEEK(CC+J)
360 POKE I0+4982+J*40,PEEK(L+J)
370 POKE I0+4863+J*40,PEEK(ZZ+J)
400 POKE I0+3687+J*40,PEEK(Z+128+J)
410 POKE I0+3680+J*40,PEEK(Z+128+J)
420 POKE I0+3719+J*40,PEEK(Z+200+J)
430 POKE I0+3712+J*40,PEEK(Z+200+J)
450 POKE I0+3646+J*40,PEEK(Z+136+J)
460 POKE I0+3721+J*40,PEEK(Z+136+J)
470 POKE I0+3678+J*40,PEEK(Z+192+J)
480 POKE I0+3753+J*40,PEEK(Z+192+J)
500 POKE I0+3605+J*40,PEEK(Z+144+J)
510 POKE I0+3762+J*40,PEEK(Z+144+J)
520 POKE I0+3637+J*40,PEEK(Z+184+J)
530 POKE I0+3794+J*40,PEEK(Z+184+J)
550 POKE I0+3564+J*40,PEEK(Z+152+J)
560 POKE I0+3803+J*40,PEEK(Z+152+J)
570 POKE I0+3596+J*40,PEEK(Z+176+J)
580 POKE I0+3835+J*40,PEEK(Z+176+J)
600 POKE I0+3523+J*40,PEEK(Z+160+J)
610 POKE I0+3844+J*40,PEEK(Z+160+J)
620 POKE I0+3555+J*40,PEEK(Z+168+J)
630 POKE I0+3876+J*40,PEEK(Z+168+J)
650 POKE I0+3482+J*40,PEEK(Z+168+J)
660 POKE I0+3885+J*40,PEEK(Z+168+J)
670 POKE I0+3514+J*40,PEEK(Z+160+J)
680 POKE I0+3917+J*40,PEEK(Z+160+J)
690 NEXT J
700 PLOT 0,0:DRAWTO 319,0:DRAWTO 319,191:DRAWTO 0,191
:DRAWTO 0,0:PLOT 0,0:DRAWTO 319,191:PLOT 319,0:DRAWTO
 0,191
710 FOR W=0 TO 360
720 X1=COS(W)*95+159:Y1=SIN(W)*95+95
730 X2=COS(W)*95+159:Y2=SIN(W)*55+95
740 X3=COS(W)*55+159:Y3=SIN(W)*55+95
750 X4=COS(W)*15+159:Y4=SIN(W)*15+95
760 PLOT X1,Y1:PLOT X2,Y2:PLOT X3,Y3:PLOT X4,Y4
770 NEXT W
780 FOR L=8 TO 64 STEP 8:PLOT L,0:DRAWTO L,191:NEXT L
790 FOR L=255 TO 311 STEP 8:PLOT L,0:DRAWTO L,191:NEX
```

```
T L
800 FOR J=0 TO 7:READ Q:POKE I0+8+J*40,Q:POKE I0+31+J
*40,Q:POKE I0+7368+J*40,Q:POKE I0+7391+J*40,Q:NEXT J
810 FOR J=0 TO 7:READ Q:POKE I0+329+J*40,Q:POKE I0+35
0+J*40,Q:POKE I0+7049+J*40,Q:POKE I0+7070+J*40,Q:NEXT
 J
820 FOR J=0 TO 7:READ Q:POKE I0+650+J*40,Q:POKE I0+66
9+J*40,Q:POKE I0+6730+J*40,Q:POKE I0+6749+J*40,Q:NEXT
 J
900 TRAP 990
910 FOR U=1 TO 40:FOR J=0 TO 7:POKE I0+6399+U+J*40,PE
EK(Z+(ASC(C$(U,U))-32)*8+J):NEXT J:NEXT U
990 DATA 85,170,85,170,85,170,85,170,204,204,51,51,20
4,204,51,51,240,240,240,240,15,15,15,15
```

30: Der Benutzer kann bis zu 40 Zeichen eingeben, die im unteren Teil in das Testbild eingefügt werden.

50 und 60: berechnen die Start-Adressen der im Testbild verwendeten Zeichen und Buchstaben.

100 bis 690: Die Zeichen werden in den Bildschirmspeicher gePOKEt.

700 bis 790: PLOTten den grafischen Teil des Testbildes.

800 bis 820: fügen frei gestaltete Grafikzeichen ein, deren DATA in Zeile 990 abgelegt sind.

900: verhindert den Programmabbruch mit ERROR-, denn in Zeile

910: werden die 40 vom Benutzer eingegebenen Zeichen verarbeitet. Dazu ist die Schleife auf 40 Durchläufe eingestellt. Wenn aber eingangs weniger als 40 Zeichen eingegeben wurden, würde das Programm abgebrochen, wenn nicht in 900 die TRAP-Falle aufgestellt wäre. U zählt die 40 Spaltenpositionen, J die acht Daten-Byte eines Zeichens. 6399 ist die Adresse im Bildschirmspeicher, in die das oberste Bit-Muster des ersten Zeichens geschrieben werden muß. C$(U,U) nimmt das U-te Zeichen aus C$ heraus und ASC wandelt den String-Ausdruck in einen ATASCII-Wert um. Durch -32 wird der ATASCII-Wert in internen Code umgewandelt.

Z hat in Zeile 40 den Wert 57344 bekommen, das ist die Start-Adresse des Zeichensatzes. Es ist sinnvoll, Zahlenwerte in einer Variablen zu erfassen, wenn sie mehrmals in einem Programm benötigt werden. Es erleichtert nicht nur das Schreiben des Programms sondern spart auch Speicherplatz ein, weil das Programm dadurch kürzer wird und eine einzelne Variable weniger Speicher belegt als ein Zahlenwert.

3.3.3 Bild-Daten speichern

Wenn Sie das Programm TESTBILD.BI2 laufen lassen, dauert es 4'45", bis die gesamte Grafik aufgebaut ist, was hauptsächlich durch die aufwendigen Berechnungen für die Ellipse und die Kreise verursacht wird. Es ist schade, ein Bild zu löschen, was mit einem solchen Rechenaufwand erzeugt worden ist. Wie der Bildschirminhalt einer hochauflöschenden (HIRES) Grafik als Hardcopy (oder screen dump) auf dem Matrix-Drucker der Nachwelt erhalten werden kann, lesen Sie im Kapitel 4.

Außer in Form kleiner schwarzer Punkte auf weißem Papier kann das Computer-Kunstwerk aber auch 'soft' auf einer Diskette gespeichert werden. Das folgende Programm bewahrt die kostbaren Bild-Daten in einem File mit dem Namen BILD8SM.DAT auf:

```
0 REM BILD8SML.BI2
1000 SM=PEEK(88)+PEEK(89)*256
1010 OPEN #1,8,0,"D1:BILD8SM.DAT"
1020 FOR J=0 TO 7679:PUT #1,PEEK(SM+J):NEXT J
1030 CLOSE #1
```

1000: berechnet die Start-Adresse des Bildschirmspeichers.

1010: eröffnet einen Schreib-Kanal zur Diskettenstation 1 für ein File mit dem Namen BILD8SM.DAT

1020: und diese FOR-NEXT-Schleife liest die 7680 Daten-Byte aus dem Bildschirmspeicher (PEEK(SM+J)) und schreibt sie auf die Diskette.

1030: schließt den Datenkanal.

Das File belegt übrigens 62 Sektoren auf der Diskette, also etwa ein Achtel der gesamten Diskette. Achten Sie also darauf, daß noch genügend Platz auf der verwendeten Diskette vorhanden ist, bevor Sie dieses Programm laufen lassen.

Nachdem die 7,5 kByte (7680 Byte) Daten in dem File abgelegt sind, können sie auf umgekehrten Wege genauso leicht und schnell wieder in den Bildschirmspeicher hineingePOKEt werden:

```
0 REM BILD8SME.BI2
1100 OPEN #1,4,0,"D1:BILD8SM.DAT"
1110 GRAPHICS 24:SM=PEEK(88)+PEEK(89)*256
1120 FOR J=0 TO 7679:GET #1,D:POKE SM+J,D:NEXT J
1130 CLOSE #1
```

```
1150 GOTO 1150
```

1100: öffnet einen Lese-Kanal zur Diskettenstation 1 und dem File BILD8SM.DAT, das die Bild-Daten enthält.

1110: natürlich muß GR.8 (ohne Textfenster) eingeschaltet und die Start-Adresse des Bildschirmspeichers errechnet werden.

1120: Diesmal liest die FOR-NEXT-Schleife die Daten-Byte von der Diskette und schreibt sie in die Speicherzellen der SM hinein.

1130: Datenkanal zu und

1150: eine Zeile, die das Ende des Programms verhindert. Die Grafik (GR.24) würde sonst gelöscht, um das READY (GR.0) zu zeigen.

Die 7680 Speicherzellen zu füllen, dauert mit diesem Programm 2'30". Das Testbild ist also fast doppelt so schnell aufgebaut, wie durch das Programm TESTBILD.BI2.

Wie lange es dauert, mit PLOT und DRAWTO eine Grafik aufzubauen, ist abhängig davon, wie komplex das Bild gegliedert ist und ob umfangreiche Berechnungen durchgeführt werden müssen, um die PLOTs zu ermitteln, wie z.B. bei Kreisen und Graphen aller Art.

Einen in Daten-Byte abgespeicherten Bildinhalt in den Bildschirmspeicher zu laden, dauert immer genau gleich lange, weil unabhängig von der Gestalt der Grafik immer genau 7680 Daten übertragen werden müssen.

3.4 Bunte Zeichen

Mit dem behandelten Wissen über die Struktur des Zeichensatzes und das Format, in dem Farb-Daten und Bit-Muster vom Rechner verarbeitet werden, sind auch die Text-Modi GRAPHICS 12 und 13 leicht zu verstehen.

In GR.0 werden in den Zellen des Bildschirmspeichers Dezimalwerte der an der entsprechenden Stelle des Bildschirms darzustellenden Zeichen in Werten des internen Codes abgelegt. Der Rechner holt sich also aus der Bildschirm-Zelle den internen Code des Zeichens. Mit diesem Wert findet er im Zeichensatz-Speicher die Daten-Byte, die die Bit-Muster des Zeichens enthalten, und diese Bit-Muster werden auf die Bildfläche geschrieben. Bei dieser Darstellung der Zeichen hat jeder der acht mal acht Punkte, die eine Schreibstelle ausfüllen, ein Bit Information: Bit gesetzt - Pixel in COLOR 1, Bit nicht gesetzt - Pixel in COLOR 0.

Bei den vierfarbigen Grafik-Modi werden jeweils zwei Bit für die Farbinformation benötigt. Ein Daten-Byte nimmt die Information für vier Pixel auf.

In GRAPHICS 12 und 13 können ebenfalls vier Farbregister angesprochen werden, also jeweils zwei Bit bestimmen den COLOR-Wert für ein Pixel. Dadurch verdoppelt sich der Informationsgehalt. Um das auszugleichen, sind die Zeichen in diesen beiden Betriebsarten nur vier Pixel breit, jeder Bildpunkt des Zeichens ist aber auf dem Bildschirm zwei Bildpunkte breit, so daß die bunten Zeichen genauso breit auf der Mattscheibe erscheinen, wie der Standard-Zeichensatz. Die Auflösung ist gröber, dafür stehen verschiedene Farben zur Verfügung.

Während die Zeichen in GR.0 die Auflösung und Farbmöglichkeiten von GR.8 haben, beziehen sich die Zeichen in GR.12 auf die Möglichkeiten von GR.15.

Wenn Sie GRAPHICS 12 einschalten, dann zeigt der Vektor in Adresse 756 auf die Start-Adresse des Standard-Zeichensatzes und die aufgerufenen Zeichen beziehen ihre Daten von hier. Durch die gerade beschriebene Farbeigenschaft von GR.12 werden die Daten-Byte aber nicht als Bit-Muster interpretiert, sondern als Farbinformation umgesetzt. Aus diesem Grund erscheinen in GRAPHICS 12 und 13 kunterbunte Zeichen die nur noch mit Mühe zu entziffern sind.

Die folgenden Abbildungen zeigen noch einmal den Unterschied in der Interpretation eines Daten-Byte in GR.0 als Bit-Muster und in GR.12 und 13 als Farbinformation.

Das Daten-Byte mit dem Dezimalwert 228

Bit	7	6	5	4	3	2	1	0
228	1	1	1	0	0	1	0	0
Dezimalwert	128	64	32	16	8	4	2	1

wird also nicht als Bit-Muster wie in GR.0 interpretiert,

Bit	7	6	5	4	3	2	1	0
228	■	■	■			■		
Dezimalwert	128	64	32	16	8	4	2	1

sondern als Farbinformation,

Bit	7	6	5	4	3	2	1	0
228	1	1	1	0	0	1	0	0
Dezimalwert	128	64	32	16	8	4	2	1
	CO.3		CO.2		CO.1		CO.0	

die vier Pixel verschiedener Farben bewirkt.

Bit	7	6	5	4	3	2	1	0
228	■	■	▒	▒	▒	▒		
Dezimalwert	128	64	32	16	8	4	2	1
	CO.3		CO.2		CO.1		CO.0	

Da die Pixel in GR.12 und 13 doppelt so breit sind wie in GR.0 wird der gleiche Speicherumfang benötigt. Ein Zeichen ist ein Byte breit, 40 Zeichen nimmt der Bildschirm nebeneinander auf, also benötigt eine Zeile 40 Byte Speicher.

Um diese beiden Betriebsarten sinnvoll zu nutzen, ist es also nötig, einen speziellen Zeichensatz zu definieren, der die Farbinformationen berücksichtigt. Wie ein Zeichensatz neu entworfen werden kann, ist bereits in Abschnitt 1.2.2 eingehend beschrieben worden.

Auflösung und Farbigkeit dieser Modi entspricht dem Grafikgang 15. Der große Vorteil liegt darin, daß jeweils acht Byte durch einen einfachen PRINT-Befehl aufgerufen und an

eine beliebige Bildschirmstelle gebracht werden können. Das ist bei feiner Farbverteilung wesentlich handlicher und viel schneller, als jedes einzelne Farbpünktchen zu PLOTten.

```
0 REM CHRGR12.BI2
1 GOSUB 32631
2 GRAPHICS 12:POKE 756,MND
3 FOR X=0 TO 19:? #6;"!!!!!!!!!!!##########!!!!!!!!!!#
##########";:NEXT X
4 END
32631 DIM MSP$(80):POKE 756,224:MND=PEEK(106)-4:CST=M
ND*256:POKE 106,MND:GRAPHICS 0
32632 FOR X=1 TO 40:READ ACHR:MSP$(X,X)=CHR$(ACHR):NE
XT X
32633 DATA 104,104,133,213,104,133,212,104,133,215,10
4,133,214,152,72,138,72,162,4,160,0,177,212,145,214
32634 DATA 200,208,249,230,213,230,215,202,208,240,10
4,170,104,168,96
32635 I=USR(ADR(MSP$),224*256,CST)
32636 FOR X=1 TO 3:FOR I=0 TO 7:READ TK:POKE CST+X*8+
I,TK:NEXT I:NEXT X
32700 DATA 85,85,125,125,125,125,85,85
32701 DATA 0,0,0,0,0,0,0,0
32702 DATA 255,255,195,195,195,195,255,255
32767 POKE 756,MND:RETURN
```

1: springt erst einmal in das bekannte Unterprogramm zum Laden eines frei definierten Zeichensatzes.

32636: verändert die Daten des ersten bis dritten Zeichens. Die DATA dafür finden sich in den Zeilen

32700 bis 32702: Verwendet werden nur das erste und dritte Zeichen. Das zweite Zeichen ist die Anführung ("). Dieses Zeichen kann in einem String nicht enthalten sein, weil das Betriebssystem Anführungsstriche als Zeichen für Anfang oder Ende des String selbst ansieht. Sollen trotzdem Anführungsstriche gePRINTet werden, so geht das nur über die CHR$-Anweisung (? CHR$(34)).

2: Nach Aufruf eines GRAPHICS muß der Vektor in 756 restauriert werden.

3: Die PRINT-Anweisung bringt die beiden neu gestalteten Zeichen in das Grafik-Fenster. Durch die FOR-NEXT-Schleife wird der gesamte Bildschirm gefüllt.

Das Ausrufezeichen ist das erste Zeichen im Charakter-Set (nach dem Leerzeichen, das ist das nullte). Die Raute ist das dritte Zeichen. Das '!!' bringt also die DATA aus Zeile

32700 auf den Bildschirm, die Raute die Daten-Byte aus Zeile 32702.

Es muß noch erwähnt werden, daß in GRAPHICS 12 und 13 eine fünfte Farbe aufgerufen werden kann. Wird nämlich ein Zeichen gePRINTet, dessen interner Code größer als 127 ist, dann wird das Zeichen in GR.0 invers dargestellt. Bei bunten Zeichen ist eine inverse Darstellung nicht sehr sinnvoll. Deshalb bewirkt das siebente Bit im internen Code (Dezimalwert 128), daß sich der COLOR-Wert 3 (binär 11) nicht auf Farbregister 710 bezieht, sondern 711 anspricht.

Es besteht folgende Beziehung zwischen COLOR-Werten und Farbregistern:

binär 0 0	COLOR 0	Farbregister 712 (Hintergrund)
binär 0 1	COLOR 1	Farbregister 708 (Grafikpunkt 1)
binär 1 0	COLOR 2	Farbregister 709 (Grafikpunkt 2)
binär 1 1	COLOR 3	Farbregister 710 (Grafikpunkt 3)
		Farbregister 711 (Grafikpunkt 3)

Ändern Sie in Zeile 3 des Listings einige Zeichen des Strings in Inverse um, und beobachten Sie, welche Farbanteile der Zeichen den Farbton wechseln.

4 Drucker

Was man schwarz auf weiß besitzt, kann man bekanntlich getrost nach Hause tragen. Doch dem Computer das Schreiben beizubringen, geht nicht ohne Druck. Als Computer-Neuling begnügt man sich gerne damit, auf der Mattscheibe des häuslichen Fernsehapparates zu lesen, was der Rechner zu sagen hat. Mit der Zeit entsteht aber ein immer größeres Bedürfnis, handfeste Unterlagen zu bekommen.

Es fängt damit an, daß man in längeren Programmen nach Fehlern suchen muß und auf dem Bildschirm immer nur einige Zeilen überprüfen kann. Da ist nur schwer Überblick zu bewahren. Bei einem Ausdruck hat man nicht nur das gesamte Programm vor Augen, man kann schriftliche Korrekturen anbringen.

Unersetzlich wird der Drucker bei der Erstellung von Dokumenten. Die schönste Computer-Grafik verschwindet von der Bildfläche, wenn der Strom abgeschaltet wird. Einen Ausdruck kann sich wer will, sogar an die Wand hängen. Und bei Anwendungen wie Textverarbeitung, Adressenverwaltung oder Buchführung (buhah!) ist der Drucker sogar Voraussetzung.

Doch mit einem weiteren Peripherie-Gerät kommen auch weitere Probleme ins Haus. Schließlich ist jeder Computer-Typ ein Individualist, jeder Drucker-Typ aber nicht weniger. Und zwei Individualisten unter einen Hut zu bringen, ist immer wieder ein Kunststück für sich.

Überhaupt keine Probleme haben Sie, wenn Sie sich zu Ihrem ATARI-Computer einen ATARI-Drucker kaufen. Geräte der gleichen Marke sind natürlich aufeinander abgestimmt. Sie werden einfach verkabelt und ab geht die Post. Leider bietet ATARI keinen wirklich leistungsfähigen Drucker an. Wer also etwas professionellen Komfort sucht, muß sich anderswo umsehen (und sehen, wie er zurecht kommt).

4.1 Steuerung

Um Computer und Drucker (verschiedener Fabrikate) miteinander zu verbinden, reicht ein einfaches Kabel nicht aus. In der schönen neuen Computer-Welt ist eben nichts einfach, was auch kompliziert zu lösen ist (wenn es um Marktanteile geht). Jeder Hersteller braut sein eigenes Hardware-Süppchen, das die Käufer auslöffeln dürfen.

Um Rechner und Drucker auf einander einzustimmen braucht es ein Interface, das genau auf beide Geräte abgestellt ist. (Lassen Sie sich bloß keinen Drucker andrehen, ohne ein passendes Interface!) Für sehr viele Drucker geeignet ist das MPP-1150 Parallel Printer Interface for ATARI Computers von Microbits. Für sehr viele, nicht für alle.

Aber selbst mit dem teuersten Drucker und einem funktionierenden Interface können Sie nicht alles auf Papier bringen, was der Bildschirm zu zeigen hat. Normale Interfaces arbeiten im sog. Transparent Mode, d.h. sie nehmen die Daten-Byte, die der Rechner ihnen sendet lediglich auf und geben sie unverändert an den Drucker weiter. Ein solches Interface ermöglicht also nur, daß überhaupt Daten fließen. Und zwar ASCII-Codes.

Der Computer sendet eine 65 (ATASCII 65 = "A"), der Drucker empfängt 65 und druckt ein A. Der Drucker übernimmt nicht das Bitmuster des ATARI "A". Er verfügt über einen eigenen Zeichensatz, der bestimmt, welche Gestalt das "A" bekommt. Inverse Zeichen mag kein Drucker von sich geben. Und den speziellen Pseudografik-Zeichensatz versteht ein gewöhnlicher Drucker auch nicht. Er kann nur drucken, was in seinem Speicher an Zeichen definiert ist. Anspruchsvollere Geräte bieten dafür die Möglichkeit, daß der Anwender einen eigenen Character-Set definiert (download character set), über den beliebige Zeichen eingegeben werden können.

Einig sind sich (international) alle Geräte nur über den ASCII-Code. Allerdings wurde im Kapitel 1.2 schon darauf hingewiesen, daß ATARI die ASCII-Codes 0 bis 31 mit Grafik-Zeichen belegt hat, während der Drucker diese Codes für Steuerbefehle wie Zeilenvorschub (LF), Formularvorschub (FF) und ähnliches nutzt. Wenn Sie einem Drucker mit Ihrem ATARI den Auftrag erteilen: Drucke Grafik-Zeichen diagonale Linie (LPRINT"(CONTROL) G"), das ist (AT)ASCII 7, dann wird der Drucker entsprechend seiner Code-Liste sein Klingelsignal erklingen lassen.

Die Ausgabe von Daten an den Drucker erfolgt also durch

das Kommando LPRINT (LP.). Im übrigen ist das Format dieses Befehls aufgebaut wie die PRINT-Anweisung an den Bildschirm. Sie können also eingeben:

```
LPRINT "A"          oder: LPRINT CHR$(65)
```

Beide Anweisungen veranlassen den Drucker, ein großes "A" auf das Papier zu hämmern.

In der gleichen Form werden auch die Steuerzeichen an den Drucker geschickt:

```
LPRINT CHR$(7)
```

läßt den Signalton des Druckers erklingen,

```
LPRINT CHR$(10)
```

veranlaßt den Drucker, die Walze um eine Zeile vorzubewegen.

Nun kommt heute keine Drucker mehr mit den 32 Steuerzeichen aus. Viele zusätzliche Funktionen haben weitere Steuer-Sequenzen nötig gemacht und die sind natürlich bei (fast) jedem Drucker anders. Im folgenden sollen Sie Drucker vier verschiedener Firmen kennnenlernen und erfahren, wie sie die Aufgabe bewältigen, das Testbild aus Kapitel 3.3.2 auszudrucken.

4.2 Hardcopy

Solange es darum geht, einfachen Text zu drucken, unterscheiden sich Drucker nur in der Geschwindigkeit, im Bedienungskomfort und im Schriftbild. Die meisten Nadeldrucker bieten aber heute die Option, jede Drucknadel einzeln anzusprechen und durch den Druck einzelner Punkte eine beliebige Grafik zu erzeugen, in sich in diesem Punktraster darstellen läßt (dot matrix graphic).

Das Prinzip ähnelt der Darstellung und Datenerfassung bei der Bildschirm-Grafik. Jeder einzelne Druckpunkt wird einem Bit zugeordnet, acht Bit werden zu einem Daten-Byte zusammengerechnet. Dieser Dezimalwert repräsentiert ein bestimmtes Bit-Muster und veranlaßt die entsprechenden Drucknadeln zu drucken oder nicht zu drucken.

Der einzige Unterschied besteht darin, daß beim Bildschirm die Bit zeilenweise zu Daten-Byte zusammengefaßt werden. Das ist sinnvoll, weil die Bildröhre zeilenweise arbeitet. Der Druckkopf arbeitet jedoch spaltenweise. Er verfügt über acht Drucknadeln, die untereinander angeordnet sind. Eine neunte Nadel dient zum Unterstreichen. In jedem Arbeitsschritt

erzeugt der Drucker eine Spalte von acht (neun) Punkten und aus mehreren solcher Punktspalten setzt sich ein Schriftzeichen zusammen.

Bei der Dot-Grafik wird das Bit-Muster einer Punktspalte in einen Dezimalwert umgerechnet. Jeder Druckdraht ist einem der acht Bits zugeordnet. Üblicherweise wird der oberste Druckdraht vom MSB (Bit 7, dezimal 128) angesprochen, der unterste vom LSB (Bit 0, dezimal 1).

Wurde der Drucker durch eine entsprechende Steuer-Sequenz auf Dot-Grafik eingestellt, dann interpretiert er die empfangenen Daten-Byte nicht als ASCII-Codes, die er als Schriftzeichen ausdrucken würde, sondern als Bit-Muster, die er durch Einzeldraht-Steuerung umsetzt.

Theoretisch ist es nun kein Problem mehr, eine Grafik, die auf dem Monitor zu sehen ist, auf dem Drucker auszugeben. Wie die Daten der Bildschirm-Grafik im Speicher des Computers zu finden sind, ist bereits bekannt. Sie müssen von dort also nur gelesen und als Dot-Grafik-Signale an den Drucker ausgegeben werden.

Das wesentliche Probelm besteht darin, daß die Bit-Muster des Bildschirms horizontal zu Daten-Byte zusammengerechnet sind, der Drucker aber vertikal zusammengefaßte Bit-Muster erwartet. Für dieses Problem gibt es zwei grundsätzliche Lösungsmöglichkeiten.

Man kann die Daten-Byte aus dem Bildschirmspeicher des Computers herauslesen, sie belassen, wie sie sind und als Dot-Grafik-Daten an den Drucker abgeben. Die Grafik vom Bildschirm entsteht dann auf dem Drucker um 90° gedreht. Das werde ich im folgenden als Druck in vertikaler oder Y-Richtung bezeichnen.

Wenn in Y-Richtung gedruckt wird, muß das Hardcopy-Programm lediglich die Reihenfolge der Daten-Byte umstellen. würden die Informationen aus dem Bildschirmspeicher in der Reihenfolge (vertikal) gedruckt, wie sie dort abgelegt sind, würde ein spiegelverkehrter Druck entstehen. In Y-Richtung müssen die Bildschirm-Daten also in irgendeiner Weise rückwärts verarbeitet werden. Entweder die (Bildschirm-) Spalten werden fortlaufend gedruckt, in jeder Spalte aber die Daten rückwärts verarbeitet, oder die Spalten werden rückwärts abgefragt, innerhalb der Spalten die Speicherzellen aber fortlaufend gelesen. Die Druckergebnisse sind jeweils identisch, aber um 180° auf dem Blatt gedreht. Da die Daten-Byte selbst unverändert bleiben, ist eine Hardcopy von GRAPHICS 8 (bzw. 24) auf diese Weise in wenigen Minu-

ten fertig.

Die zweite Methode ist wesentlich aufwendiger. Das Programm holt sich jeweils acht auf dem Bildschirm untereinander liegende Byte aus dem Speicher, spaltet diese acht Byte in ihre Bit-Muster auf und faßt sie spaltenweise zu druckergerechten Daten-Byte zusammen. Auf diese Weise errechnet, benötigt eine Hardcopy etwa fünfzehn Minuten. Der entscheidende Vorteil liegt nur darin, daß in X-Richtung gedruckt wird.

Warum ist das ein Vorteil?

Lassen Sie TESTBILD.BI2 laufen. Wenn Sie die Grafik auf der Mattscheibe haben, nehmen Sie einen Zirkel und messen den großen Kreis nach. Sie werden (erstaunt?) feststellen, daß es gar kein Kreis ist. Die Bildpunkte des Monitors haben in der Zeile einen größeren Abstand als die Zeilen voneinander. Diese Überdehnung durch die Raster-Auflösung heißt auf Neudeutsch Overscan. Ein durchschnittlicher Wert für Bildschirme liegt bei 10% horizontal.

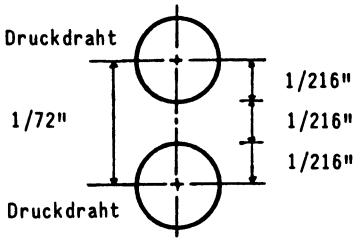
Mit bloßem Auge nehmen wir diesen Overscan kaum wahr, obwohl 10% eigentlich eine ganze Menge sind. Wir nehmen diese Abweichung auch nur deshalb nicht wahr, weil das Bild horizontal überdehnt ist. In vertikaler Richtung würde schon eine geringere Abweichung unangenehm auffallen. Unsere Wahrnehmung ist an eine waagerechte Verzerrung gewöhnt. Alles was wir täglich sehen, betrachten wir unter einem schrägen Blickwinkel. Die kreisrunde Kante einer Tasse oder Büchse sehen wir fast immer schräg von oben. Der optische Sachverhalt unserer Wahrnehmung ist also eine mehr oder weniger flache Ellipse. Unser Intellekt überlagert jedoch diese objektive Wahrnehmung. Wir wissen eben aus Erfahrung, daß ein Zylinder kreisförmig ist. Deswegen sehen wir das Oval als "Kreis" und dieser Vorgang wird uns nicht einmal mehr bewußt. Die Verzerrung durch schräge Draufsicht wird unbewußt ausgeglichen. Deshalb nehmen wir eine Überdehnung in horizontaler Richtung kaum wahr. Wir müssen dafür extra "hinsehen".

Nun haben aber auch (fast) alle Drucker einen beachtlichen Overscan, wenn sie im Einzelnadelbetrieb arbeiten. Die acht Druckdrähte stehen dichter beisammen als die Punkte nebeneinander gedruckt werden. Wie verschieden die Druckbilder unterschiedlicher Drucker aussehen, zeigen Ihnen die folgenden Seiten.

Ein für Nadeldrucker durchaus üblicher Overscan im Grafik-Modus beträgt ca. 20%. Eine solche Verzerrung ist sogar in

X-Richtung störend. In Y-Richtung ist eine solche Verzerrung unerträglich. (Vielleicht nicht für das Auge eines Technikers.) Der Vorteil des langsamen Hardcopy-Programms, das in X-Richtung arbeitet, liegt also im ästhetischen Bereich.

Um einen Hardcopy-Ausdruck zu erhalten, muß als erstes der Zeilenvorschub des Druckers so eingestellt werden, daß zwischen den einzelnen Druckzeilen kein Freiraum bleibt, sondern auch in der Vertikalen die Punkte dicht an dicht stehen. Je nach Drucker-Typ kann der Zeilenzwischenraum auf n/72", n/144 und/oder n/216" eingestellt werden.



Der Radius der Druckdrähte und der Abstand zwischen den Drähten betragen 1/216". Der Abstand der Druckdrähte von Mittelpunkt zu Mittelpunkt gemessen beträgt demnach 1/72", der halbe Abstand 1/144".

Matrix-Drucker drucken normalerweise bi-direktional, also die erste Zeile von links nach recht, die zweite von rechts nach links. Dadurch wird die Zeit eingespart, die der Druckkopf benötigt, um vom Zeilenende zum linken Rand zurückzufahren. Allerdings führt diese Arbeitsweise auch zu minimalen Abweichungen im Druckbild. Wenn Sie sich die Listings in diesem Buch ansehen, können Sie feststellen, daß die Druckzeilen ein wenig gegeneinander versetzt sind. Wenn Sie z.B. die erste Ziffer der Zeilennummern betrachten, können sie diese Abweichung deutlich sehen.

In einem Text ist diese kleine Verschiebung akzeptabel. Wenn Sie sich aber in einer Dot-Grafik eine senkrechte Linie vorstellen und in jeder Zeile, also alle acht Punkte, eine kleiner Bruch entstünde, wäre das sehr störend und unbefriedigend. Einige Matrix-Drucker schalten deshalb im Grafik-Mode automatisch auf uni-direktionalen Druck um. Bei anderen Druckern muß dieses Kommando gesondert ergehen.

Um den Drucker auf Dot-Grafik umzuschalten, braucht es eine Sequenz von Kommandos, die bei den verschiedenen Druckern sehr unterschiedlich ausfallen können. Grundsätzlich aber besteht dieses Kommando aus zwei Teilen. Einmal muß der Grafik-Mode selbst eingeschaltet werden, dann muß

festgelegt werden, wie viele Daten-Byte als Bit-Muster für Dot-Grafik interpretiert werden sollen. Diese Anzahl wird in der Reihenfolge LO (low Byte) HI (high Byte) an den Drucker gesendet. Die Summe der Grafik-Daten ist also LO+HI*256 oder umgekehrt, Grafik-Daten durch 256 teilen, der ganzzahlige Teiler ist HI, der Rest LO.

Das gesammte Kommando für Dot-Grafik könnte lauten:

LPRINT CHR$(27);CHR$(75);CHR$(LO);CHR$(HI)

wobei für LO und HI die entsprechenden Werte einzusetzen sind.

Nun gibt es aber eine Schwierigkeit. Wenn das obige LPRINT-Kommando als BASIC-Zeile in einem Programm steht, dann endet diese Programmzeile mit einem RETURN (das auf dem Bildschirm nicht als Zeichen erscheint, im Speicher des Computers aber vermerkt ist. Wenn dieses Kommando an den Drucker ergeht, wird das RETURN vom Drucker als CR (carriage return) empfangen. CR hat den ASCII-Wert 13. Da der Drucker aber schon auf Grafik-Mode umgeschaltet ist, interpretiert er das Daten-Byte 13 als Bit-Muster und druckt es aus. Deshalb kann die LPRINT-Anweisung hier nicht verwendet werden.

Es gibt aber eine zweite Form, Daten an ein Peripherie-Gerät, auch an einen Drucker, zu senden. Es wird ein Daten-Kanal eröffnet, in diesem Fall ein Schreib-Kanal an den Zeilendrucker und mit dem PUT-Befehl werden die einzelnen Daten-Byte abgesetzt.

Das komplette Hardcopy-Programm (Druck in Y-Richtung) sieht dann so aus:

```
0 REM HARDCOPY.STA
1 DIM D(191)
30000 LPRINT CHR$(27);CHR$(51);CHR$(16);CHR$(27);CHR$
(85);CHR$(1)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR B=0 TO 39
30040 FOR Z=0 TO 191
30050 D(Z)=PEEK(SM+Z*40+B)
30060 NEXT Z
30070 OPEN #1,8,0,"P:"
30080 PUT #1,27:PUT #1,75:PUT #1,192:PUT #1,0
30090 FOR P=191 TO 0 STEP -1:PUT #1,D(P):NEXT P
30100 CLOSE #1
30110 NEXT B
```

1: Die indizierte Variable D nimmt 192 Daten-Byte auf.

Vor dem eigentlichen Hardcopy-Programm muß natürlich irgendein anderes Programm liegen, daß in GRAPHICS 24 eine Grafik auf die Mattscheibe und in den Bildschirmspeicher zaubern.

Das vorliegende Hardcopy-Programm kann nur GRAPHICS 24 ausdrucken. Für GR.8 müßte die Anzahl der gelesenen Zeilen von 192 auf 159 geändert werden. Bei den übrigen Grafik-Modi müßten weitergehende Überlegungen zur Umsetzung der Farben und der Größe der einzelnen Grafikpunkte angestellt werden.

30000: In dieser Zeile wird der Zeilenabstand auf 16/144" eingestellt und uni-direktionaler Druck angeordnet. Diese Steuerkommandos gelten für den radix-10 (star). Bei anderen Modellen können andere Kommandos notwendig sein. Das Umschalten auf uni-direktionalen Druck ist bei diesem Drucker im Grafik-Mode nicht erforderlich, es wird automatisch veranlaßt.

30020: berechnet die Start-Adresse des Bildschirmspeichers.

30030: Die wichtigste Überlegung ist, in welcher Reihenfolge die Daten-Byte aus dem Bildschirmspeicher gelesen und an den Drucker abgegeben werden. Dieses Programm druckt in Y-Richtung. Es druckt also in jeder Druckzeile eine Spalte des Bildschirms (192 Byte) aus. Auf einer Bildschirmzeile liegen 40 Daten-Byte nebeneinander. Der Drucker muß also 40 Zeilen drucken. B zählt diese Zeilen.

30040: Z zählt die 192 Daten-Byte, die ein Spalte auf dem Bildschirm füllen und zu einer Zeile im Drucker zusammengefaßt werden.

30050: Die indizierte Variable D nimmt die 192 Daten-Byte auf. Der Ausdruck in der Klammer ermittel aus Z und B die jeweiligen Adressen im Bildschirmspeicher.

30060: D wird mit 192 Daten-Byte gefüllt.

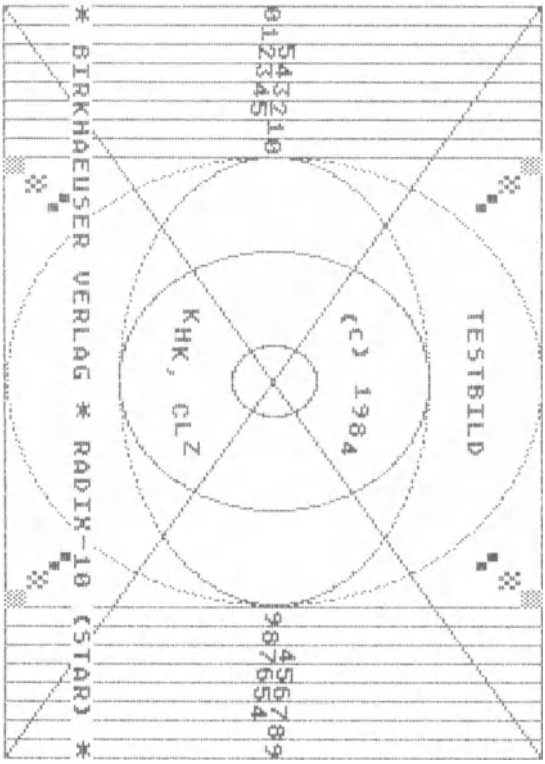
30070: eröffnet einen Schreib-Kanal an den Zeilendrucker (line-printer).

30080: ordnet Dot-Grafik an mit den Daten-Byte 27 und 75. 192 (LO) und 0 (HI) bestimmen die folgenden 192 Daten-Byte als Bit-Muster.

30090: Die 192 Daten-Byte für eine Druckzeile werden rückwärts (von 191 bis 0) an den Drucker abgegeben, damit der Druck nicht spiegelverkehrt entsteht.

30100: schließt den Datenkanal.

Und so sieht das Testbild in Y-Richtung gedruckt (auf radix-10 von star) aus:



Wenn Sie das Buch um 90° drehen, damit Sie das Bild richtig sehen, fällt der Overscan ausgesprochen unangenehm auf. Deshalb nun ein Hardcopy-Programm, das in X-Richtung den Bildschirminhalt aufs Papier bringt:

```
0 REM HARDCOPX.STA
1 DIM D(319),C(7,7)
30000 LPRINT CHR$(27);CHR$(51);CHR$(16)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR ZZ=0 TO 23
30040 FOR P=0 TO 39
30050 FOR X=0 TO 7:FOR Y=0 TO 7:C(X,Y)=0:NEXT Y:NEXT
```

```
X
30060 FOR Z=0 TO 7
30070 A=PEEK(SM+P+Z*40+ZZ*320)
30080 IF A>127 THEN C(7,Z)=1:A=A-128
30090 IF A>63 THEN C(6,Z)=1:A=A-64
30100 IF A>31 THEN C(5,Z)=1:A=A-32
30110 IF A>15 THEN C(4,Z)=1:A=A-16
30120 IF A>7 THEN C(3,Z)=1:A=A-8
30130 IF A>3 THEN C(2,Z)=1:A=A-4
30140 IF A>1 THEN C(1,Z)=1:A=A-2
30150 IF A>0 THEN C(0,Z)=1
30160 NEXT Z
30210 FOR Y=0 TO 7
30220 D(P*8+7-Y)=C(Y,0)*128+C(Y,1)*64+C(Y,2)*32+C(Y,3
)*16+C(Y,4)*8+C(Y,5)*4+C(Y,6)*2+C(Y,7)
30230 NEXT Y
30250 NEXT P
30300 OPEN #1,8,0,"P:"
30310 PUT #1,27:PUT #1,75:PUT #1,64:PUT #1,1
30320 FOR J=0 TO 319:PUT #1,D(J):NEXT J
30330 REM HIER PUT#1,10 EINFÜGEN, WENN LF UNTERDRÜCKT
30340 CLOSE #1
30350 NEXT ZZ
30360 LPRINT CHR$(27);CHR$(64)
```

1: Die indizierte Variable D muß jetzt 320 Daten-Byte aufnehmen können, weil auf dem Bildschirm 320 Punkte in einer Zeile liegen. Die doppelt-indizierte Variable C wird acht mal acht Bit aufnehmen.

Hier steht das Grafik-Programm (z.B. TESTBILD.BI2)

30000: stellt den Zeilenvorschub ein. Uni-direktionaler Druck wird nicht verordnet, weil der verwendete Drucker das automatisch einstellt.

30030: Der Drucker druckt den gesamten Bildschirminhalt in 24 Zeilen (zu je acht Punkten = 192 Punkte) aus.

30040: Eine Bildschirmzeile belegt 40 Speicherzellen.

30050: setzt die doppelt-indizierte Variable auf 0.

30060 bis 30160: holen acht Daten-Byte aus dem Speicher, die auf dem Bildschirm untereinander liegen, spalten jedes einzelne Daten-Byte in seine Bits auf und speichern diese Informationen in der doppelt-indizierten Variablen C.

30210 bis 30230: rechnen die acht mal acht Bit spaltenweise zu acht Daten-Byte zusammen.

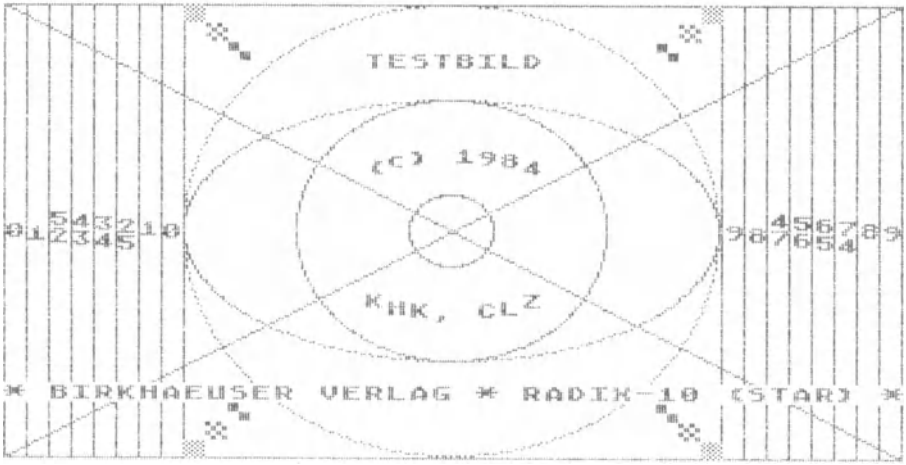
30310: Der Drucker muß hier auf 320 Grafik-Byte eingestellt werden (LO=64, HI=1).

30320: Hier wird eine Zeile gedruckt, die 320 Punkte breit und acht Punkte hoch ist.

30330: Drucker können so eingestellt werden, daß sie bei Empfang eines CR automatisch ein LF auslösen. Ist das automatische LF unterdrückt, muß hier ein Zeilenvorschub extra angeordnet werden. Statt PUT kann auch LPRINT CHR$ (10) geschrieben werden. (Wie Sie in dieser Zeile auch sehen, gibt es immer Aerger mit den deutschen Umlauten. Wir sollten unsere Sprache noch staerker fuer Computer kompatibel machen, oder?)

30360: initialisiert den Drucker, d.h. alle Einstellungen für Zeilenabstand etc. werden so eingestellt, wie sie beim Einschalten des Druckers festgelegt werden.

Und so sieht das gleiche Testbild aus, wenn es mit dem vorstehenden Programm auf dem genannten Drucker in X-Richtung umgesetzt wird:



Der Overscan, der ja rein numerisch genauso stark ist wie im vorigen Druckergebnis, ist zwar wahrzunehmen, wenn man darauf achtet, fällt aber nicht so peinlich auf.

4.3 Fabrikate

In diesem Abschnitt werden vier weitere Drucker-Modelle vorgestellt, die in ihrer Verschiedenheit das gesamte Spektrum der Besonderheiten und Druckqualitäten ausleuchten. Die Auswahl der hier vorgestellten Typen stellt in keiner Weise ein Qualitätsurteil dar, sondern erfolgte lediglich unter dem Gesichtspunkt der Verschiedenheit in Steuerung und Druckergebnis bei Einzelpunkt-Grafik.

Drei der vorgestellten Geräte lassen sich problemlos mit dem Interface MMP-1150 von Microbits mit dem ATARI verbinden. Die Firma star liefert für Ihre Drucker ein eigenes Interface.

4.3.1 M-8510 von C.ITOH

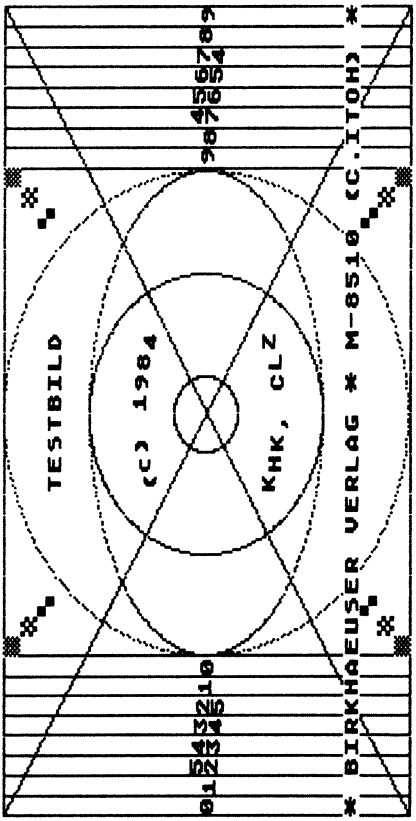
Der M-8510 ist ein Drucker der mittleren Preisklasse, der über alle üblichen Kommandos verfügt. In den Zeilen 30000 und 30310 müssen die abweichenden Kommandos eingefügt werden. Eine Besonderheit bei diesem Gerät besteht darin, daß die Menge der Daten-Byte, die als Bit-Muster interpretiert werden sollen, nicht im Zwei-Byte-Format (LO, HI) einzugeben sind, sondern als vier einzelne Ziffern, die als ASCII-Wert gesendet werden müssen. Sollen also 320 Daten-Byte angekündigt werden, so sind die Ziffern 0, 3, 2, 0 als ASCII-Werte 48, 51, 50, 48 zu senden.

Dieser Drucker wartet aber noch mit einer ganz besonderen Überraschung auf. Im Grafik-Modus sind die Druckdrähte den einzelnen Bit des Daten-Byte anders zugeordnet, als es allgemein üblich ist. Der oberste Draht wird vom LSB (Bit 0) angesprochen, der unterste von MSB (Bit 7).

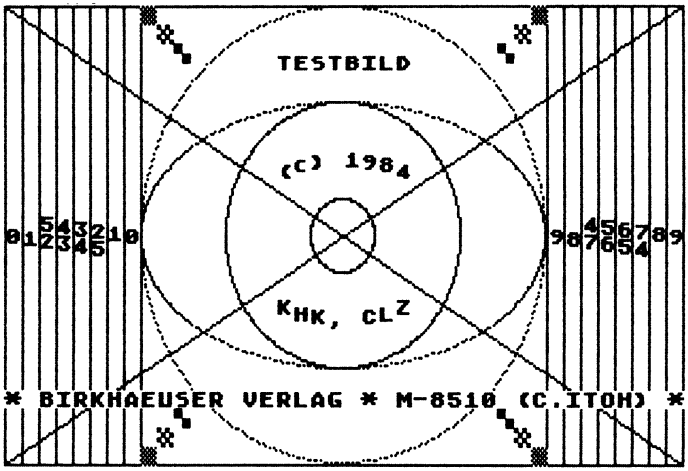
Diese Besonderheit wird im Programm HARDCOPY.CIT dadurch berücksichtigt, daß die Daten-Byte aus dem Bildschirmspeicher in einer anderen Reihenfolge gelesen werden. In Zeile 30030 werden die 40 Daten-Byte einer Bildschirmzeile rückwärts verarbeitet, es werden also erst die 192 Daten der letzten Bildschirmspalte gelesen und in der indizierten Variablen abgelegt. In Zeile 30090 können sie dafür dann in der normalen Reihenfolge, also nicht rückwärts, an den Drucker abgegeben werden.

Im Programm HARDCOPX.CIT liegt der wesentliche Unterschied in Zeile 30210. Hier werden die aufgespaltenen Bit in umgekehrter Reihenfolge mit Zweierpotenzen bewertet, so daß das MSB unten und das LSB oben angeordnet ist.

Die Druckergebnisse beider Programme unterscheiden sich




```
0 REM HARDCOPY.CIT
30000 LPRINT CHR$(27);CHR$(84);CHR$(49);CHR$(55);CHR$
(27);CHR$(62)
30010 DIM D(191):SM=PEEK(88)+PEEK(89)*256
30030 FOR B=39 TO 0 STEP -1
30040 FOR Z=0 TO 191
30050 D(Z)=PEEK(SM+Z*40+S)
30060 NEXT Z
30070 OPEN #1,8,0,"P:"
30080 PUT #1,27:PUT #1,83:PUT #1,48:PUT #1,49:PUT #1,
57:PUT #1,50
30090 FOR P=0 TO 191:PUT #1,D(P):NEXT P
30100 CLOSE #1
30110 NEXT B
```




```
0 REM HARDCOPX.CIT
1 DIM D(319),C(7,7)
30000 LPRINT CHR$(27);CHR$(51);CHR$(16)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR ZZ=0 TO 23
30040 FOR P=0 TO 39
30050 FOR X=0 TO 7:FOR Y=0 TO 7:C(X,Y)=0:NEXT Y:NEXT
X
30060 FOR Z=0 TO 7
30070 A=PEEK(SM+P+Z*40+ZZ*320)
30080 IF A>127 THEN C(7,Z)=1:A=A-128
30090 IF A>63 THEN C(6,Z)=1:A=A-64
30100 IF A>31 THEN C(5,Z)=1:A=A-32
30110 IF A>15 THEN C(4,Z)=1:A=A-16
30120 IF A>7 THEN C(3,Z)=1:A=A-8
30130 IF A>3 THEN C(2,Z)=1:A=A-4
30140 IF A>1 THEN C(1,Z)=1:A=A-2
30150 IF A>0 THEN C(0,Z)=1
30160 NEXT Z
30210 FOR Y=0 TO 7:D(P*8+7-Y)=C(Y,7)*128+C(Y,6)*64+C(
Y,5)*32+C(Y,4)*16+C(Y,3)*8+C(Y,2)*4+C(Y,1)*2+C(Y,0):N
EXT Y
30250 NEXT P
30310 OPEN #1,8,0,"P:":PUT #1,27:PUT #1,83:PUT #1,48:
PUT #1,51:PUT #1,50:PUT #1,48
30320 FOR J=0 TO 319:PUT #1,D(J):NEXT J
30340 CLOSE #1
30350 NEXT ZZ
```

nicht wesentlich von denen des radix-10. Die Grafikpunkte stehen etwas dichter zueinander. Das Druckbild ist etwa 10% kleiner und dichter.

Durch das Umdrehen der Bit-Wertigkeit erzeugt das in X-Richtung druckende Hardcopy-Programm ein Bild, dessen Overscan in vertikaler Richtung liegt, das in Y-Richtung druckende Programm überdehnt in waagerechter Richtung. Im Endergebnis ist das unerheblich, da der Overscan auch hier ca. 20% beträgt.

4.3.2 RX-80 von EPSON

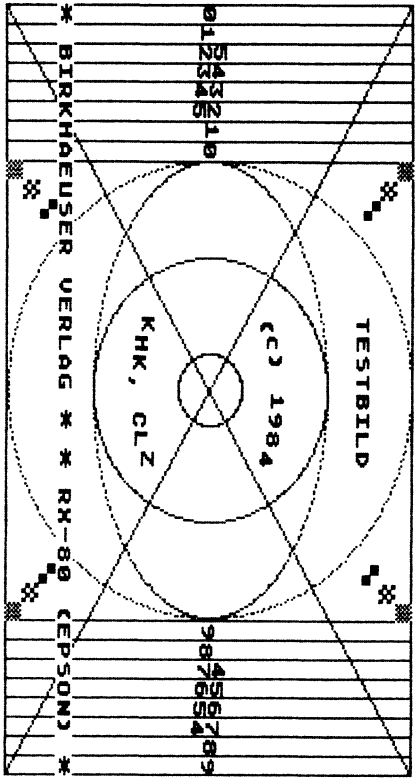
Auch der RX-80 ist in die mittlere Preisgruppe zu rechnen. Er kann nur mit Traktor-Papier betrieben werden, es gibt aber ein sonst identisches Modell, das auch Einzelbatt bedruckt.

Der EPSON verfügt über fünf verschiedene Grafik-Betriebsarten. Die erste ist Dot-Grafik mit sog. normaler Dichte. In einer Druckzeile stehen 480 Punkte nebeneinander. Ein zweiter Modus erlaubt den Druck in doppelter Dichte. 960 Punkte füllen die Zeile und überdecken sich teilweise. Dadurch wird ein dichteres und schärferes Druckbild ermöglicht. Sogar vierfache Dichte (1920 Punkte pro Zeile) steht zur Verfügung. Aber Drucker, die Einzelnadel-Grafik beherrschen, bieten meist diese verschiedenen Dichten an.

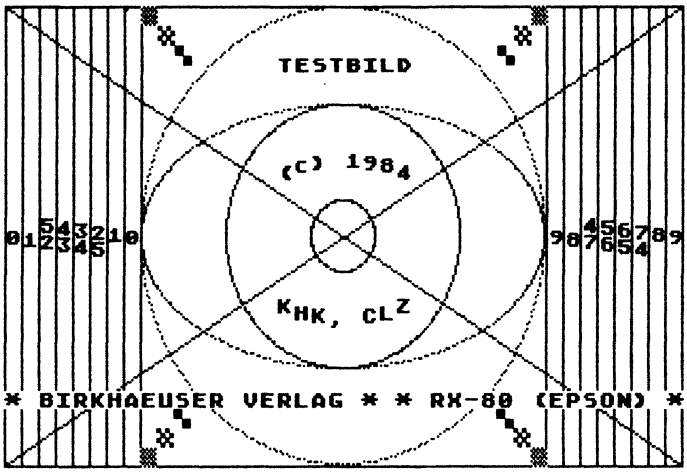
Als Besonderheit kann die sog. CRT-Grafik aufgerufen werden, die laut Anleitungsbuch besonders für Hardcopy geeignet ist. Die folgenden Hardcopy-Programme für den RX-80 laufen deshalb natürlich in diesem Grafik-Modus.

Bei der CRT-Grafik stehen 640 Punkte auf einer Druckzeile, das sind 30% mehr als im normalen Grafik-Betrieb. Daraus läßt sich schon vorab errechnen, daß statt eines horizontalen Overscan von 20%, ein vertikaler von 10% zu erwarten ist. D.h. auch bei diesem speziellen Hardcopy-Grafik-Modus bleibt ein Overscan von 10%, der sich optisch noch sehr deutlich bemerkbar macht. Allerdings liegt er senkrecht, so daß mit dem schnellen Y-Hardcopy-Programm gearbeitet werden kann. Der Overscan, den das Druckergebnis schließlich aufweist, entspricht übrigens dem, den TV-Schirme üblicherweise zeigen.

Die fünfte Grafik-Betriebsart ist CRT-Grafik II, bei der 720 Spalten auf einer Zeile liegen. Und der Vollständigkeit halber sei noch erwähnt, daß es einen Grafikgang mit doppelter Dichte und doppelter Druckgeschwindigkeit gibt.




```
0 REM HARDCOPY.EPS
1 DIM D(191)
30000 LPRINT CHR$(27);CHR$(65);CHR$(8);CHR$(27);CHR$(
60)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR B=0 TO 39
30040 FOR Z=0 TO 191
30050 D(Z)=PEEK(SM+Z*40+B)
30060 NEXT Z
30070 OPEN #1,8,0,"P:"
30080 PUT #1,27:PUT #1,42:PUT #1,4:PUT #1,192:PUT #1,
0
30090 FOR P=191 TO 0 STEP -1:PUT #1,D(P):NEXT P
30100 CLOSE #1
30110 NEXT B
```




```
0 REM HARDCOPX.EPS
1 DIM D(319),C(7,7)
30000 LPRINT CHR$(27);CHR$(65);CHR$(8);CHR$(27);CHR$(
60)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR ZZ=0 TO 23:FOR P=0 TO 39
30050 FOR X=0 TO 7:FOR Y=0 TO 7:C(X,Y)=0:NEXT Y:NEXT
X
30060 FOR Z=0 TO 7
30070 A=PEEK(SM+P+Z*40+ZZ*320)
30080 IF A>127 THEN C(7,Z)=1:A=A-128
30090 IF A>63 THEN C(6,Z)=1:A=A-64
30100 IF A>31 THEN C(5,Z)=1:A=A-32
30110 IF A>15 THEN C(4,Z)=1:A=A-16
30120 IF A>7 THEN C(3,Z)=1:A=A-8
30130 IF A>3 THEN C(2,Z)=1:A=A-4
30140 IF A>1 THEN C(1,Z)=1:A=A-2
30150 IF A>0 THEN C(0,Z)=1
30160 NEXT Z
30210 FOR Y=0 TO 7:D(P*8+7-Y)=C(Y,0)*128+C(Y,1)*64+C(
Y,2)*32+C(Y,3)*16+C(Y,4)*8+C(Y,5)*4+C(Y,6)*2+C(Y,7):N
EXT Y
30250 NEXT P
30300 OPEN #1,8,0,"P:"
30310 PUT #1,27:PUT #1,42:PUT #1,4:PUT #1,64:PUT #1,1
30320 FOR J=0 TO 319:PUT #1,D(J):NEXT J
30340 CLOSE #1
30350 NEXT ZZ
```

4.3.3 MT-80 von Mannesmann Tally

Dieser Drucker gehört in die untere Preiskategorie, wird aber allen Anforderungen an einen Nadeldrucker gerecht. Der geringere Preis muß natürlich mit gewissen Nachteilen im Bedienungskomfort und Abstrichen bei den verfügbaren Kommandos erkauft werden.

Auffällig ist, daß das Druckbild im ganzen etwas dichter ist, die Schriftzeichen fallen deutlich kleiner aus, als bei anderen Nadeldruckern. Dieses Druckbild und auch die Art der Bedienung ist ähnlich wie bei einigen Billigdruckern aus Fernost, die im Preis deutlich unter diesem Modell liegen.

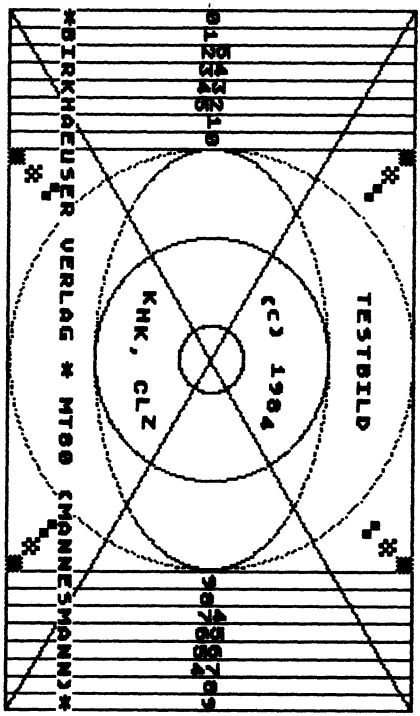
Die Kommandos des Mannesmann-Druckers MT-80 sind weitgehend identisch mit denen des radix-10. Einzige Merkwürdigkeit, für den Zeilenabstand muß (warum auch immer) der Wert 22 eingesetzt werden. Das müßte eigentlich einem Zwischenraum von 22/72" entsprechen, aber wie auch immer, mit diesem Wert wird ein dem Augenschein nach gutes Ergebnis erzielt.

Interessant ist, daß ausgerechnet dieses Gerät, das unter den vorgestellten das schlichteste und billigste ist, im Grafik-Modus eine Hardcopy liefert, die fast keine Überdehnung aufweist. Legt man einen in Y-Richtung gedruckten Screen-dump und einen in X-Richtung gedruckten übereinander, zeigt sich nur eine ganz minimale Abweichung.

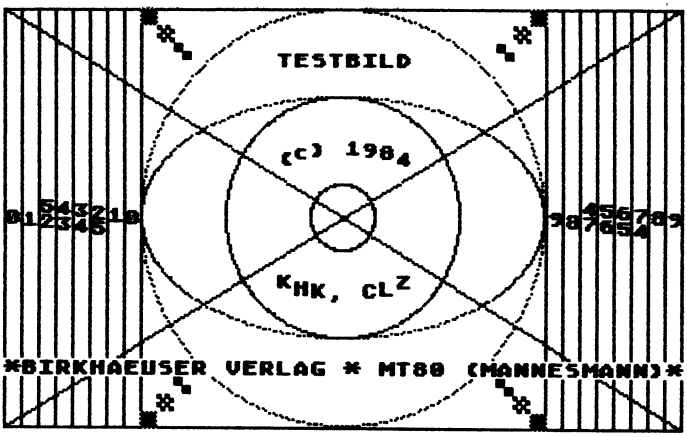
Da die Druckpunkte sehr dicht zusammenstehen, ergibt sich jedoch ein relativ kleines Bild, das etwas matschig wirkt. Hier müssen Sie als Käufer abwägen, was Sie von Ihrem Drucker erwarten. Suchen Sie hohen Bedienungskomfort und eine breite Palette von Drucksteuerungen mit vielen Schrifttypen, Ausdruck in Korrespondenz-Qualität (CQ oder NLQ = near letter quality) und besonders die Möglichkeit, eingene Schriftzeichen zu definieren und in den Drucker zu laden, müssen Sie ein anderes Gerät wählen, als wenn Sie lediglich ein Listing lesbar ausdrucken wollen. Wenn Sie großen Wert auf hohe Grafikqualität legen, sollte Ihre Wahl wiederum anders ausfallen.

4.3.4 radix-15i von star

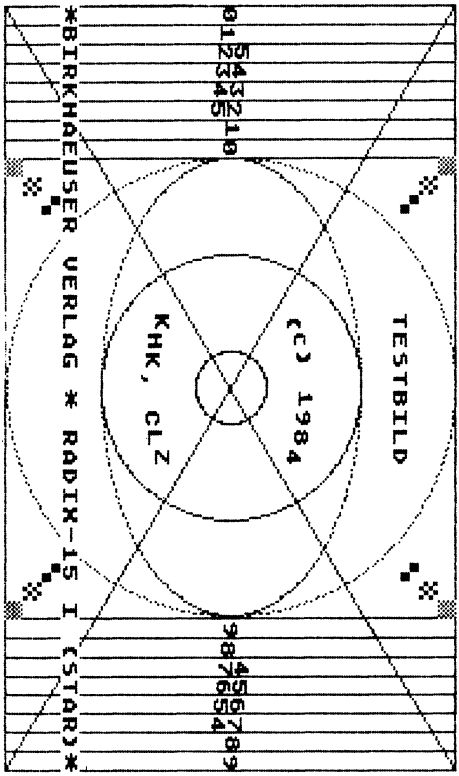
Ein Drucker, der alle Anwendungen zur größten Zufriedenheit erledigt, ist der radix-15i. Dieses Gerät ist aber auch bei weitem das teuerste unter den vorgestellten. Deutlich preiswerter ist der radix-10i, der sich nur in der Breite des



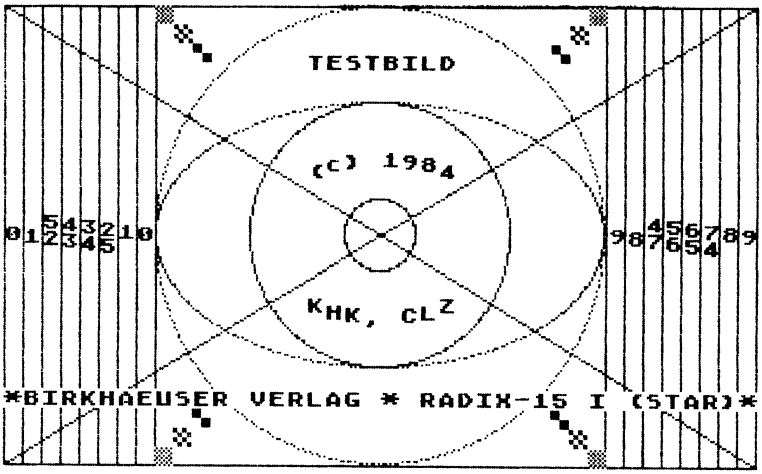

```
0 REM HARDCOPY.MAN
1 DIM D(191)
30000 LPRINT CHR$(27);CHR$(51);CHR$(22);CHR$(27);CHR$
(85)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR B=0 TO 39
30040 FOR Z=0 TO 191
30050 D(Z)=PEEK(SM+Z*40+B)
30060 NEXT Z
30070 OPEN #1,8,0,"P:"
30080 PUT #1,27:PUT #1,75:PUT #1,192:PUT #1,0
30090 FOR P=191 TO 0 STEP -1:PUT #1,D(P):NEXT P
30100 CLOSE #1
30110 NEXT B
```




```
0 REM HARDCOPX.MAN
1 DIM D(319),C(7,7)
30000 LPRINT CHR$(27);CHR$(51);CHR$(22);CHR$(27);CHR$
(85):SM=PEEK(88)+PEEK(89)*256
30030 FOR ZZ=0 TO 23
30040 FOR P=0 TO 39
30050 FOR X=0 TO 7:FOR Y=0 TO 7:C(X,Y)=0:NEXT Y:NEXT
X
30060 FOR Z=0 TO 7
30070 A=PEEK(SM+P+Z*40+ZZ*320)
30080 IF A>127 THEN C(7,Z)=1:A=A-128
30090 IF A>63 THEN C(6,Z)=1:A=A-64
30100 IF A>31 THEN C(5,Z)=1:A=A-32
30110 IF A>15 THEN C(4,Z)=1:A=A-16
30120 IF A>7 THEN C(3,Z)=1:A=A-8
30130 IF A>3 THEN C(2,Z)=1:A=A-4
30140 IF A>1 THEN C(1,Z)=1:A=A-2
30150 IF A>0 THEN C(0,Z)=1
30160 NEXT Z
30200 FOR Y=0 TO 7
30210 D(P*8+7-Y)=C(Y,0)*128+C(Y,1)*64+C(Y,2)*32+C(Y,3
)*16+C(Y,4)*8+C(Y,5)*4+C(Y,6)*2+C(Y,7)
30220 NEXT Y
30230 NEXT P
30300 OPEN #1,8,0,"P:"
30310 PUT #1,27:PUT #1,75:PUT #1,64:PUT #1,1
30320 FOR J=0 TO 319:PUT #1,D(J):NEXT J
30330 CLOSE #1
30340 NEXT ZZ
```




```
0 REM HARDCOPY.STI
1 DIM D(191)
30000 LPRINT CHR$(27);CHR$(51);CHR$(24);CHR$(27)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR B=0 TO 39
30040 FOR Z=0 TO 191
30050 D(Z)=PEEK(SM+Z*40+B)
30060 NEXT Z
30070 OPEN #1,8,0,"P:"
30080 PUT #1,27:PUT #1,42:PUT #1,5:PUT #1,192:PUT #1
,0
30090 FOR P=191 TO 0 STEP -1:PUT #1,D(P):NEXT P
30100 CLOSE #1
30110 NEXT B
```




```
0 REM HARDCOPX.STI
1 DIM D(319),C(7,7)
30000 LPRINT CHR$(27);CHR$(51);CHR$(24)
30020 SM=PEEK(88)+PEEK(89)*256
30030 FOR ZZ=0 TO 23
30040 FOR P=0 TO 39
30050 FOR X=0 TO 7:FOR Y=0 TO 7:C(X,Y)=0:NEXT Y:NEXT
X
30060 FOR Z=0 TO 7
30070 A=PEEK(SM+P+Z*40+ZZ*320)
30080 IF A>127 THEN C(7,Z)=1:A=A-128
30090 IF A>63 THEN C(6,Z)=1:A=A-64
30100 IF A>31 THEN C(5,Z)=1:A=A-32
30110 IF A>15 THEN C(4,Z)=1:A=A-16
30120 IF A>7 THEN C(3,Z)=1:A=A-8
30130 IF A>3 THEN C(2,Z)=1:A=A-4
30140 IF A>1 THEN C(1,Z)=1:A=A-2
30150 IF A>0 THEN C(0,Z)=1
30160 NEXT Z
30170 FOR Y=0 TO 7:D(P*8+7-Y)=C(Y,0)*128+C(Y,1)*64+C(
Y,2)*32+C(Y,3)*16+C(Y,4)*8+C(Y,5)*4+C(Y,6)*2+C(Y,7):N
EXT Y
30180 NEXT P
30200 OPEN #1,8,0,"P:"
30210 PUT #1,27:PUT #1,42:PUT #1,5:PUT #1,64:PUT #1,1
30220 FOR J=0 TO 319:PUT #1,D(J):NEXT J
30230 CLOSE #1
30240 NEXT ZZ
```

ausdruckbaren Formates (hier 10", dort 15") unterscheidet. Beide Geräte sind IBM kompatibel und verfügen über einen zweiten Zeichensatz, der den wichtigsten Teil des umfangreichen IBM-Zeichensatzes enthält.

Der Bedienungskomfort läßt keine Wünsche übrig. Für Einzelblatt gibt es einen automatischen Papiereinzug. Natürlich kann auch Endlospapier mit Traktorführung bedruckt werden. Besonderen Komfort aber bietet der 16-kByte-Druckpuffer.

Die Datenübertragung vom Computer zum Drucker ist naturgemäß viel schneller als der Druckvorgang selbst. Dieser Drucker nimmt bis zu 16.000 Zeichen in einen Puffer genannten Zwischenspeicher auf und gibt den Computer schon frei, während der Druckvorgang noch lange fortgeht. 16 kByte fassen immerhin gut die Hälfte des Anwender-RAM vom ATARI! Wenn Sie den Drucker also beispielsweise ein Listing drucken lassen, gehen nach dem Befehl LIST"P:" die Daten in kurzer Zeit an den Drucker ab und werden dort im Puffer zwischengespeichert. Nach NEW können Sie schon ein neues Programm schreiben oder laden, während der Drucker noch einige Zeit am Ausdruck des Listings arbeitet.

Im Rahmen dieses Druckervergleichs ist aber besonders interessant, daß der radix-10i (oder radix-15i) über einen weiteren Grafik-Modus verfügt, der als Plotter-Grafik bezeichnet wird. Diese Betriebsart füllt eine 10"-Zeile mit 576 Punkten und schafft damit ein absolut verzerrungsfreis Bild. Die in X-Richtung und in Y-Richtung gedruckten Hardcopies sind völlig deckungsgleich.

Die Steuerbefehle dieses star-Druckers weichen an vielen Stellen von denen des bereits vorgestellten radix-10 des gleichen Herstellers ab, sind aber, wo gleiche Funktionen vorliegen, identisch mit den EPSON-Kommandos. Wenn zwei Hersteller von solcher Marktbedeutung zu einheitlichen Steuerbefehlen gefunden haben, dann kann man hoffen, daß sich hier nach und nach ein durchgehender Standard etabliert. Wer schon mit fertiger Software gearbeitet hat, die auch den Drucker anspricht, z.B. eine Textverarbeitung oder ein Buchführungsprogramm, der weiß, wie lästig es ist, das teuer gekaufte Programm mühsam an den Drucker anzupassen. Wären die Steuerkommandos für alle Drucker einheitlich vereinbart, wäre diese Anpassung überflüssig.

5 Diskette

Das DOS (Disk Operating System) ist ein Disketten-Betriebs-Programm, das den Datenfluß zwischen Computer und Diskettenstation überwacht und steuert. Mit DOS wird der Betrieb der Diskettenstation erst möglich.

DOS ist nicht zu verwechseln mit dem DOS-Menü, über das verschiedene Hilfsprogramme aufgerufen werden können (z.B. Kopieren von Programmen, Einrichten eines MEM.SAV-Files, mit dem BASIC-Programme zwischengespeichert werden können). Das eigentliche Disk Operating System wird hingegen automatisch geladen, sobald in die betriebsbereite Diskettenstation eine Diskette eingelegt und der Computer eingeschaltet wird.

Von ATARI werden drei verschiedene DOS-Versionen angeboten. Davon ist DOS 1 das älteste und inzwischen am wenigsten benutzte Disketten-Betriebs-System. DOS 2 bietet demgegenüber größeren Betriebs-Komfort und ist das derzeit am weitesten verbreitete System. Aus diesem Grund sind die Programm-Beispiele im Buch auch auf DOS 2 abgestimmt.

Allerdings können auch die Benutzer des neueren DOS 3 die hier beschriebenen Disketten-Operationen anwenden, da sich die beiden DOS-Versionen nur dadurch unterscheiden, daß DOS 3 eine größere Speicherkapazität der Diskette ermöglicht und ein deutschsprachiges DOS-Menü anbietet.

In den nachfolgenden Abschnitten wird nicht auf die einfachen Disketten-Operationen wie SAVE "D:NAME" oder das Erstellen eines Überschreib-Schutzes eingegangen. Solche Funktionen sind in der Betriebsanleitung ausreichend beschrieben und dürften jedem Diskettenstation-Besitzer geläufig sein. Stattdessen wird der effektive Umgang mit den sog. "wild cards" vermittelt sowie Disketten-Operationen ohne Aufruf des DOS-Menüs. Später finden Sie dann fortgeschrittene Anwendungen wie Datenverwaltung, Errichten von Textfiles und die AUTO-RUN-Funktion, mit der bestimmte Programme automatisch gestartet werden können. Besonders nützlich sind die Programmierhilfen zur Erzeugung von DATA-Zeilen und vollständigen BASIC-Programmen mit Hilfe der Diskettenstation. Mit diesen Hilfen können Sie sich z.B. ein bescheidenes, aber bequemes Textverarbeitungs-System programmieren oder Daten zum Betrieb anderer Programme abspeichern.

5.1 Arbeiten im DOS-Menü

Neben den ganz normalen Operationen mit den DOS-Hilfsprogrammen (Formatieren einer Diskette, Kopieren des DOS-Files oder Erstellen eines Überschreib-Schutzes gespeicherter Programme) gibt es einige komfortable Funktionen, mit denen sich Disketten-Operationen erleichtern lassen.

Zu diesen Erleichterungen zählen die sog. "wild cards"; das sind die Zeichen ? und *. Das Fragezeichen ersetzt ein einzelnes Zeichen; das Sternchen einen ganzen Filenamen oder Extender (Namens-Erweiterung).

Wenn Sie sich im DOS-Menü befinden und das Inhalts-Verzeichnis der Diskette (Directory) auslisten lassen wollen, müssen Sie normalerweise eingeben:

```
A                (RETURN) (RETURN)
```

Nun wird die Directory auf dem Bildschirm ausgegeben. Falls sich jedoch mehr als 24 Programme auf der Diskette befinden (insgesamt kann die Directory 64 Namen erfassen), erleben Sie das sog. Scrolling; d.h. das Inhalts-Verzeichnis rollt relativ schnell über den Bildschirm, so daß Sie die oberen Namen nicht mehr lesen können. Mit einem der "wild cards" können Sie erreichen, daß nur der Teil der Directory ausgelistet wird, der Sie interessiert (z.B. alle Programme mit dem Extender LST). In diesem Fall müssen Sie eingeben:

```
A                (RETURN)
*.LST            (RETURN)
```

Nun könnte es sein, daß Sie noch weiter spezifizieren wollen (z.B. wenn Sie eine Programmserie mit PROGR1.LST, PROGR2.LST, PROGR3.LST usw. begonnen haben und wissen wollen, welche Nummer als nächstes folgen müßte). In diesem Fall sind alle anderen Programme mit dem Extender LST uninteressant. Sie können sich den betreffenden Teil der Directory auslisten lassen, indem Sie eingeben:

```
A                (RETURN)
PROGR?.LST       (RETURN)
```

Auch die Kombination beider "wild cards" ist möglich; z.B.:

```
A                (RETURN)
PROGR?.*         (RETURN)
```

In diesem Fall werden alle Filenamen aufgelistet, die mit PROGR beginnen, unabhängig davon, welchen Extender sie besitzen. Falls Sie die Directory nicht auf dem Bildschirm, sondern gedruckt sehen wollen, geben Sie ein:

```
A                (RETURN)
D:,P:            (RETURN)
```

Das P steht als Abkürzung für Printer (Drucker). Mit diesem Parameter wird automatisch ein Datenkanal zum Drucker eröffnet. Demzufolge müßte bei Ausgabe auf dem Bildschirm ebenfalls ein Datenkanal eröffnet werden; z.B. durch die Angabe S (Screen = Bildschirm) oder E (Bildschirm-Editor). Da diese Ausgabeform jedoch die häufigste ist, nimmt ATARI diese Arbeit ab und eröffnet automatisch einen Datenkanal zum Bildschirm, wenn keine andere Eingabe vorgenommen wird.

Sie können die oben stehende Anweisung an den Drucker auch abkürzen:

```
A                (RETURN)
,P:              (RETURN)
```

Häufig kommt es vor, daß man beim Anblick der Directory gar nicht mehr weiß, worum es sich bei einzelnen Programmen handelt. Normalerweise müssen Sie das DOS verlassen, das Programm laden und LISTen. Diesen Umstand können Sie sich vereinfachen, indem Sie die Copy-Funktion des DOS nutzen.

Anstatt ein Programm auf eine andere (oder die gleiche) Diskette zu kopieren, müssen Sie es lediglich auf den Bildschirm kopieren:

```
C                (RETURN)
D:NAME.EXT,E:    (RETURN)
```

Durch ,E: wird der Datenkanal zum Bildschirm-Editor eröffnet und das betreffende Listing erscheint, obwohl Sie sich im DOS befinden. Natürlich können Sie auch hier "wild cards" verwenden; z.B.:

```
C                (RETURN)
D:*.LST,E:       (RETURN)
```

Nun erscheint das Listing des ersten Programms mit dem Extender LST auf dem Bildschirm. Die Voraussetzung für diese Pseudo-Copy ist allerdings, daß die betreffenden Programme auf der Diskette geLISTet sind (nicht geSAVEd). Bei geSAVEten Programmen erfolgt zwar auch eine Copy auf dem Bildschirm, jedoch ist diese im sog. Token-Format gehalten. Das sind die internen Kürzel für alle BASIC-Anweisungen. Für den unkundigen User stellen sie nur ein Wirrwar von Zeichen und Zahlen dar.

GeLISTete Programme hingegen werden in normalen ASCII-Werten abgespeichert, die bei der Copy im CHR$-Format (und damit

in lesbaren Zeichen) ausgegeben werden. Sie können sich zum besseren Verständnis einmal ein geSAVEtes und ein geLISTetes Programm auf den Bildschirm kopieren.

Die Copy-Funktion kann auch genutzt werden, um sämtliche Listings einer Diskette auf dem Drucker auszugeben. Dies ist eine äußerst angenehme Arbeitserleichterung. Denn normalerweise müßten Sie jedes einzelne Programm in den Speicher laden und mit der Anweisung LIST "P: an den Drucker abgeben. Bei Anwendung der Copy-Funktion im DOS brauchen Sie nur anstelle des Parameters E ein P (für Printer) einzugeben; also:

```
C                (RETURN)
D:NAME.EXT,P:    (RETURN)
```

Oder, falls Sie sämtliche Programme nacheinander mit Drucker geLISTet haben wollen, mit "wild cards":

```
C                (RETURN)
D:*.*,P:         (RETURN)
```

Genau wie bei der Copy auf dem Bildschirm müssen hier alle Programme auf Diskette geLISTet sein (also nicht geSAVEd).

Mit der Funktion D (Delete = Löschen) können Sie, während Sie sich im DOS-Menü befinden, ein Datenfile bzw. ein Programm löschen. Dazu geben Sie normalerweise ein:

```
D                (RETURN)
D:NAME.EXT       (RETURN)
Y                (RETURN)
```

Das Y (für YES) wird vom Computer erfragt. Es soll eine zusätzliche Sicherung sein, falls Sie nur aus Versehen die Delete-Funktion aufgerufen haben und möglicherweise ein wertvolles Programm verloren geht. Das ist auch sehr höflich vom Computer. Aber in der Regel sind Sie ganz sicher, daß Sie ein bestimmtes Programm löschen wollen. Und in diesem Fall ist das Y lästig. Mit einem kleinen Zusatz (den die deutsche Bedienungs-Anleitung leider verschweigt) können Sie das Y unterdrücken. Dies ist die Anweisung /N hinter dem Programmnamen; z.B.:

```
D                (RETURN)
D:NAME.EXT/N     (RETURN)
```

Der Einsatz von /N wird besonders attraktiv, wenn Sie mehrere Programme hintereinander löschen wollen; z.B. die Serie PROGR1.LST, PROGR2.LST, PROGR3.LST usw. In diesem Fall können Sie wieder mit den "wild cards" arbeiten und folgendes eingeben:

```
D                 (RETURN)
D:PROGR?.LST/N    (RETURN)
```

Falls sich auf der Diskette keine Programme mit dem Namen PROGRn und dem Extender LST befinden, können Sie bei oben gezeigten Delete-Anweisung auch folgendes eingeben:

```
D                 (RETURN)
D:PROGR?.*/N      (RETURN)
```

Nun werden sämtliche Programme, die mit PROGR beginnen gelöscht, gleichgültig, welchen Extender sie besitzen.

"wild cards" können auch eingesetzt werden, um sämtliche Programme gegen Überschreiben zu schützen. Das kann nützlich sein, wenn nur noch wenige Sektoren auf der Diskette frei sind und ein längeres Datenfile gespeichert werden soll. Hierbei können unter Umständen Sektoren anderer Programme zerstört werden. Bei gesicherten Programmen wird dies in jedem Fall vermieden.

Die normale Anweisung zur Errichtung eines Überschreib-Schutzes lautet:

```
F                 (RETURN)
NAME.EXT          (RETURN)
```

Mit "wild cards" können Sie sämtliche Programme gleichzeitig gegen Überschreiben schützen (oder eine bestimmte Auswahl, wenn Sie dies wünschen); z.B.:

```
F                 (RETURN)
*.*               (RETURN)
```

oder:

```
F                 (RETURN)
PROGR?.*          (RETURN)
```

Im zweiten Fall werden alle Programme, die mit dem Namen PROGR beginnen, gegen Überschreiben geschützt.

Falls Sie sämtliche Programme einer Diskette auf diese Weise schützen, so werden auch DOS.SYS, DUP.SYS (DOS-Hilfsprogramm) und MEM.SAV geschützt. Auf DOS.SYS und DUP.SYS hat dies keine nachteiligen Auswirkungen. MEM.SAV verliert jedoch seine Funktionsfähigkeit, da in diesem File ein im Speicher befindliches Programm zwischengelagert wird, sobald Sie DOS aufrufen. Sie müssen also MEM.SAV vom Überschreib-Schutz befreien:

```
G                 (RETURN)
MEM.SAV           (RETURN)
```

Selbstverständlich können auch hier "wild cards" eingesetzt werden, um mehrere oder alle Programme einer Diskette vom Überschreibschutz zu befreien; z.B.:

```
G                 (RETURN)
*.*               (RETURN)
```

```
DISK OPERATING SYSTEM II VERSION 2.0S
COPYRIGHT 1980 ATARI

A. DISK DIRECTORY          I. FORMAT DISK
B. RUN CARTRIDGE           J. DUBLICATE DISK
C. COPY FILE               K. BINARY SAVE
D. DELETE FILE(S)          L. BINARY LOAD
E. RENAME FILE             M. RUN AT ADDRESS
F. LOCK FILE               N. CREATE MEM.SAV
G. UNLOCK FILE             O. DUPLICATE FILE
H. WRITE DOS FILE


SELECT OR RETURN FOR MENU
```

Die Abbildung zeigt das gesamte DOS-Menü, so wie es sich auf dem Bildschirm darstellt. Mit I können Sie eine Diskette neu formatieren, mit H ein DOS-Menü auf die formatierte Diskette schreiben.

Durch die Eingabe von N wird das bereits erwähnte MEM.SAV erzeugt. K,L und M stehen für Binärdaten und Programme in Maschinensprache zur Verfügung. Darauf geht dieses Buch nicht ein. (Es handelt sich ja schließlich um eine BASIC-Trick-Kiste.)

Wie man Disketten-Operationen auch ohne DOS-Menü durchführen kann, erfahren Sie im folgenden Abschnitt.

5.2 DOS-Operationen mit XIO

Das Arbeiten im DOS-Menü ist zwar komfortabel; aber es hat einen Nachteil: Beim Aufruf von DOS wird das im Speicher befindliche Programm gelöscht. Wenn sich ein MEM.SAV-File auf der Diskette befindet, speichert der Rechner ein aktuelles Programm im MEM.SAV ab; doch der Ladevorgang des DOS-Menüs dauert dreimal länger als normal. Außerdem verbraucht ein MEM.SAV-File 45 Sektoren der Diskette.

Interessant sind also Möglichkeiten, DOS-Operationen ohne DOS-Menü durchzuführen. Das erspart Zeit und verbraucht keinen Speicherplatz auf der Diskette. Mit dem XIO-Befehl können einige (leider nicht alle) DOS-Aufgaben gelöst werden. Die Angaben in einem XIO-Befehl sind:

XIO Befehls-Code,Kanal-Nr.,Funktion 1,Funktion 2,Gerät

Es gibt eine ganze Reihe von Befehls-Code-Zahlen, mit denen außer DOS-Operationen noch andere BASIC-Befehle ausgeführt werden können (z.B. 3 für OPEN, 5 für INPUT u.a.). In den anderen Kapiteln dieses Buches wird im Zusammenhang mit dem behandelten Thema auf einige dieser Funktionen eingegangen. Hier interessieren nur die Code-Zahlen für DOS-Operationen, die nachfolgend erklärt werden.

Da mit XIO immer Dateneingabe oder -ausgabe gesteuert werden, muß hierfür ein entsprechender Daten-Kanal zur Verfügung stehen (bei DOS-Operationen in der Regel #1). Die Angabe für die Funktion 1 hängt von der Aufgabe ab, die gelöst werden soll; z.B. dient bei XIO-Anwendung als OPEN-Befehl eine 4 zum "Lesen vom angesprochenen Gerät" und eine 8 zum "Schreiben auf das angesprochene Gerät". Bei DOS-Operationen kann die Angabe für die Funktion 1 immer eine 0 sein.

Die Funktion 2 wird nur bei XIO-Anwendung als OPEN-Befehl (zur Festlegung der Grafik-Betriebsart, der Programm-Recorder- und Drucker-Steuerung) und bei der Programmierung der RS 232 Schnittstelle benötigt. Bei allen anderen Operationen ist der Wert von Funktion 2 eine 0.

Die Angabe für das Gerät muß bei Disketten-Operationen immer "D: oder "D:NAME.EXT sein. Soll mit XIO eine Diskette neu formatiert werden, lautet der Befehl:

```
XIO 254,#1,0,0,"D:
```

Diesen Formatierungs-Befehl können Sie auch wunderbar zum Schutz Ihrer Programme anwenden (z.B. wenn jemand gegen Ihren Wunsch versucht, ein Programm zu LISTen, lassen Sie die Diskette einfach neu formatieren. Das ist zwar "böse",

aber sehr wirksam. Näheres erfahren Sie in dem Kapitel über Programm-Schutz).

Wenn Sie einen Filenamen umbenennen wollen, können Sie dies ebenfalls ohne DOS-Menü mit XIO durchführen:

```
XIO 32,#1,0,0,"D:ALTNAME.EXT,NEUNAME.EXT
```

Übrigens kann XIO mit X. abgekürzt werden. Wenn Sie ein File von der Diskette löschen wollen, geben Sie ein:

```
XIO 33,#1,0,0,"D:NAME.EXT
```

Hierbei tritt ein Fehler auf, wenn das zu löschende Programm vor Überschreiben geschützt ist. Diesen Schutz können Sie ebenfalls mit XIO aufheben:

```
XIO 36,#1,0,0,D:NAME.EXT
```

Wenn Sie hingegen einen Überschreib-Schutz erstellen wollen, müssen Sie eingeben:

```
XIO 35,#1,0,0,D:NAME.EXT
```

Genau wie bei den Beispielen für DOS-Operationen im vorhergehenden Abschnitt können Sie beim XIO-Befehl "wild cards" einsetzen. Wenn Sie z.B. sämtliche Programme einer Diskette hintereinander löschen wollen, geben Sie ein:

```
XIO 33,#1,0,0,D:*.*/N
```

Bedenken Sie aber, daß hierbei auch DOS.SYS und DUP.SYS gelöscht werden. Besser ist es daher, nacheinander einzelne Programm-Gruppen zu löschen; z.B. mit Angaben wie:

```
XIO 33,#1,0,0,"D:PROGR?.*/N
```

Dadurch bleiben DOS.SYS und DUP.SYS erhalten.

Damit sind die Möglichkeiten des XIO-Befehls zur Bewältigung von Disketten-Operationen erschöpft. Der große Vorteil dieser Anwendung besteht darin, daß Sie Ihr augenblickliches BASIC-Programm nicht verlassen müssen. Außerdem können Sie die XIO-Befehle innerhalb von Programmen einsetzen, was z.B. bei Textverarbeitung oder Datenverwaltung sehr nützlich ist.

So ist es möglich, durch Tastatur-Eingabe einzelne Datenfiles mittels XIO zu löschen, vor Überschreiben zu schützen oder umzubenennen. Dazu müssen Sie freilich erst einmal wissen, wie man solche Files erstellt. Das erfahren Sie im folgenden Abschnitt.

5.3 Disketten-Operationen mit PRINT # und INPUT

Bislang haben Sie nur Disketten-Hilfsprogramme kennengelernt. Darüber hinaus lassen sich aber auch umfangreiche Aufgaben durch den Datenaustausch zwischen Rechner und Disketten-Station lösen; z.B. Adressen- und Datenverwaltung oder Textverarbeitung. Außerdem lassen sich auch BASIC-Programmzeilen (z.B. DATA-Zeilen) mit fortgeschrittenen Disketten-Operationen generieren.

Voraussetzung ist, daß bestimmte INPUT/OUTPUT-Kanäle geöffnet werden. Der Befehl dazu lautet:

OPEN#Kanal,Betriebsart,Funktion,Gerät

Normalerweise werden die Kanäle 1 bis 5 benutzt. Die Kanäle 6 und 7 stehen für andere Zwecke zur Verfügung (Kanal 6 zum Ausführen von Grafikbefehlen in den Betriebsarten 1 und 2; Kanal 7 zur Ausführung von CLOAD-, CSAVE- und LPRINT-Anweisungen).

Es dürfen mehrere Kanäle gleichzeitig mit verschiedenen Betriebsarten und Funktionen geöffnet sein. Allerdings kommt es zur Fehlermeldung, wenn Sie versuchen, zweimal hintereinander den gleichen Kanal zu öffnen. In der Praxis erweist es sich daher als sinnvoll, zum Programmbeginn die Datenkanäle zu schließen, die man benutzen will. Der Befehl dazu lautet:

CLOSE#Kanal

Die nachfolgenden Tabellen schlüsseln die verschiedenen Angaben im OPEN-Befehl auf.

Ansprechbare Geräte beim OPEN-Befehl*	
Angabe im OPEN-Befehl	Gerät
"C: oder "C:NAME	Programm-Recorder
"D: oder "D:NAME.EXT	Disketten-Station
"E:	Bildschirm-Editor
"K:	Tastatur (Keyboard)
"P:	Drucker (Printer)
"R:	RS 232 Schnittstelle
"S:	Bildschirm (Screen)
* Über den OPEN-Befehl lassen sich auch die RS 232 Schnittstelle und einige Bildschirm-Funktionen programmieren (z.B. Wahl der Grafik-Betriebsart). Bei Disketten-Operationen sind diese Möglichkeiten jedoch nicht interessant.	

Erster Parameter des OPEN-Befehls		
Betriebsart	Bedeutung	Gerät
4 8	Lesen (vom Gerät) Schreiben (zum Gerät)	C: C:
4 8 9 12	Lesen (von Diskette) Schreiben (auf Diskette; z.B. Datenfile) Anhängen (an bestehendes Datenfile) Lesen und Schreiben (z.B. teilweises Verändern eines bestehenden Datenfiles)	D: D: D: D:
8 12 13	Ausgabe (auf dem Bildschirm) Eingabe von Tastatur, Ausgabe auf dem Bildschirm Lesen vom Bildschirm, Ausgabe auf Bildschirm	E: E: E:
4	Tastatur-Abfrage	K:
8	Ausgabe (an Drucker)	P:

Zweiter Parameter des OPEN-Befehls		
Funktion	Bedeutung	Gerät
0 128	Normale Pausen zwischen aufgezeichneten Blöcken Kurze Pausen zwischen aufgezeichneten Blöcken	C: C:
0	Ohne Wirkung	D:
0	Ohne Wirkung	E:
0	Ohne Wirkung	K:
0 83	Normale Ausgabe der Zeichen Liegende Zeichen (ATARI-Drucker 820)	P: P:

5.3.1 Datenverwaltung

Die einfachste Form, ein Datenfile auf Diskette zu schreiben, zeigt das folgende Beispiel:

```
10 OPEN #1,8,0,"D:TEXTFILE"
20 PRINT #1;"Dies ist ein Text"
30 CLOSE #1
```

Nun befindet sich "Dies ist ein Text" in Form von ASCII-Werten unter dem Dateinamen TEXTFILE auf der Diskette. Wenn Sie ihn in den Rechner einlesen wollen, müssen Sie folgendes Programm schreiben:

```
10 DIM A$(20)
20 OPEN #1,4,0,"D:TEXTFILE"
30 INPUT #1;A$
40 PRINT A$:CLOSE #1
```

Der OPEN-Befehl in den beiden Programmen unterscheidet sich nur in der Angabe des ersten Parameters. Im ersten Listing wird mit der Betriebsart 8 das Schreiben auf Diskette ermöglicht; im zweiten mit der Betriebsart 4 das Lesen von der Diskette vorbereitet.

Das Schreiben selbst wird mit PRINT# durchgeführt; das Lesen mit INPUT# (der Datenaustausch mittels PUT# und GET# wird in den Abschnitten unter 5.4 behandelt).

Mit dem Beispiel auf der vorhergehenden Seite läßt sich das Prinzip einer Adressen-Verwaltung veranschaulichen. Versuchen Sie folgendes Programm:

```
10 PRINT CHR$(125)
20 DIM A$(50),B$(50),C$(50),D$(50)
30 OPEN #1,8,0,"D:ADRESSEN"
40 PRINT "Name, Vorname:":INPUT A$
50 PRINT "Strasse, Postfach:":INPUT B$
60 PRINT "Postleitzahl, Ort:":INPUT C$
70 PRINT "Zusatz?":INPUT D$
80 PRINT #1;A$;B$;C$;D$
90 CLOSE #1
```

Nach RUN können Sie Ihre Eingaben machen. Falls Sie keine Eingabe wünschen (z.B. bei "Zusatz"), drücken Sie einfach RETURN. Jeder eingegebene String darf maximal 50 Zeichen lang sein. Zum Abschluß werden die Texte unter dem Dateinamen ADRESSEN auf Diskette geschrieben (am besten benutzen Sie eine Experimentier-Diskette, da die hier gezeigten Programme noch keine professionellen Aufgaben bewältigen, sondern nur zum besseren Verständnis der Funktionsweise der PRINT# und INPUT#-Befehle dienen).

Nachdem Sie nun Ihre Adresse abgespeichert haben, können wir es Ihnen verraten: Das Programm ist der reinste Blödsinn. Die Adresse befindet sich zwar auf Diskette; aber nicht hübsch säuberlich nach A$,B$,C$ und D$ getrennt, sondern als ununterbrochene Zeichenkette. Der Grund dafür: Nur der Rechner unterscheidet die einzelnen String-Variablen; auf der Diskette werden ausschließlich die ASCII-Werte der einzelnen Buchstaben abgelegt.

Sie können sich das selbst beweisen, indem Sie die Adresse mit folgendem Programm wieder in den Rechner einlesen:

```
10 PRINT CHR$(125)
20 DIM A$(100)
30 OPEN #1,4,0,"D:ADRESSEN"
40 INPUT #1;A$
50 PRINT A$
60 CLOSE #1
```

Auf dem Bildschirm sehen Sie nun die Zeichenketten, die Sie zuvor einzeln eingegeben hatten, ohne Zwischenräume und Zeilenvorschub; z.B.:

```
Johannes MeierBahnhofstr. 62000 Hambur
g-Nordersteadt

READY
```

Und wie macht man's nun richtig? Der Trick ist ganz einfach: Sie müssen nach jeder Einzeleingabe ein "End of Line" (EOL) setzen. Beim normalen Arbeiten mit dem Computer wird ein EOL automatisch beim Drücken von RETURN gesetzt. Dieses Kommando (ASCII-Code = 155) ist jedoch nicht Bestandteil des eingegebenen Strings (in diesem Beispiel A$, B$ usw.), so nur die ASCII-Werte der Textzeichen auf Diskette geschrieben werden.

Sie müssen also eine spezielle EOL-Anweisung geben, damit die Adressen in gewohnter Weise auf dem Bildschirm erscheinen. Verbessern Sie das kurze Adressen-Programm folgendermaßen:

```
10 PRINT CHR$(125)
20 DIM A$(50),B$(50),C$(50),D$(50),E$(1)
25 E$=CHR$(155)
30 OPEN #1,8,0,"D:ADRESSEN"
40 PRINT "Name, Vorname:":INPUT A$
50 PRINT "Strasse, Postfach:":INPUT B$
60 PRINT "Postleitzahl, Ort:":INPUT C$
70 PRINT "Zusatz?":INPUT D$
80 PRINT #1;A$;E$;B$;E$;C$;E$;D$
90 CLOSE #1
```

Es ist nicht notwendig, die alte ADRESSEN-Datei zu löschen. Nach RUN wird sie von den neuen Eingaben automatisch überschrieben. Zum Lesen der Adresse müssen Sie das oberste Listing auf dieser Seite ebenfalls ändern:

```
10 PRINT CHR$(125)
20 DIM A$(50),B$(50),C$(50),D$(50),E$(1)
25 E$=CHR$(155)
30 OPEN #1,4,0,"D:ADRESSEN"
```

```
40 INPUT #1;A$,B$,C$,D$
50 PRINT A$;E$;B$;E$;C$;E$;D$
60 CLOSE #1
```

Mit diesen Beispielen können Sie natürlich nicht nur Adressen speichern, sondern auch andere Textblöcke oder irgendwelche Daten. Wenn Sie diese Daten an den Drucker abgeben wollen, ändern Sie die PRINT-Anweisung einfach in LPRINT. Weitere nützliche Anwendungs-Beispiele mit diesen fortgeschrittenen Disketten-Operationen finden Sie im Kapitel über String-Manipulationen.

5.3.2 DATA-Zeilen generieren

Wenn Sie in normalen Programmen DATA-Zeilen verwenden, dann liegen die einzelnen DATA-Werte durch Ihre Eingabe fest. Das ist nützlich, wenn Sie z.B. Musiknoten, Player-Missile-Daten oder Grafikzeichen definieren und während des Programmablaufs aufrufen.

Nun kommt es jedoch auch vor, daß sich die DATA-Werte für bestimmte Dinge erst während eines Programms ergeben, so wie bei dem Zeichensatz-Generator im Kapitel 3.2.2 dieses Buches. Ebenso ergeben sich die Dezimalwerte für eine Player-Missile-Grafik, wenn Sie sie mit einem Hilfsprogramm auf dem Bildschirm konstruieren, erst durch Ihre aktuellen Eingaben. Wenn Sie solche Werte in DATA-Zeilen ablegen wollen, so gelingt dies natürlich nicht innerhalb eines normalen BASIC-Programms. Mit Hilfe einer speziellen Disketten-Operation können Sie jedoch DATA-Werte mitsamt BASIC-Zeilennummern abspeichern und hinterher mit ENTER "D:NAME in den Rechner einlesen.

Sie müssen dabei die automatisch generierten Zeilennummern so wählen, daß beim ENTERn Ihr Arbeits-Programm nicht überschrieben wird. Im folgenden Demonstrations-Beispiel beginnen die DATA-Zeilen mit 1000. Sie erhöhen sich dann in 10er-Schritten, so wie es bei normaler BASIC-Programmierung praktiziert wird.

```
10 DIM A$(6):A$=" DATA "
20 OPEN #1,8,0,"D:DATAGEN"
30 ZN=1000:AD=40000
40 FOR X=1 TO 10
50 A0=PEEK(AD):A1=PEEK(AD+1):A2=PEEK(AD+2):A3=PEEK(AD
+3):A4=PEEK(AD+4)
60 PRINT #1;ZN;A$;A0;",";A1;",";A2;",";A3;",";A4;CHR$
(155)
```

```
70 ZN=ZN+10:AD=AD+5
80 NEXT X
90 CLOSE #1
```

10: A$ wird mit 6 Zeichen DIMensioniert und als " DATA " definiert.

20: Der Datenkanal zur Disketten-Station wird eröffnet. Das File heißt DATAGEN.

30: Die erste DATA-Zeile trägt die Zeilen-Nummer 1000. AD dient als Abkürzung für "Adresse". Ab der Speicherstelle 40000 beginnt in der Grafik-Betriebsart 0 der Bildschirm-Speicher; d.h. hier werden alle Zeichen registriert, die sich auf dem Bildschirm befinden. Es handelt sich hierbei jedoch nicht um ASCII-Werte, sondern um den sog. Internen Code (vergl. Tabelle im Anhang). 40000 ist die Adresse für die linke obere Ecke des Bildschirms. Sorgen Sie also dafür, daß sich irgend etwas auf dem Bildschirm befindet (z.B. das Listing dieses Programms), da sich sonst nur Null-Werte in den Speicherstellen befinden.

50: Die Variablen A0 bis A4 nehmen die Werte an, die sich aus den Speicherstellen 40000 (AD), 40001 (AD+1) usw. PEEKen lassen. Beim zweiten Durchlauf der FOR-NEXT-Schleife ist AD dann 40005 (s. Zeile 70).

60: Auf die Diskette wird die Zeilen-Nummer (beim ersten Mal 1000), A$ (DATA) und die gePEEKten Werte aus dem Bildschim-Speicher geschrieben. Genau wie beim normalen Programmieren müssen zwischen den einzelnen Werten Kommas stehen. Die DATA-Zeile wird mit CHR$(155) = EOL beendet.

70: Die BASIC-Zeilennummer soll sich bei jedem Durchlauf der FOR-NEXT-Schleife um 10 erhöhen. AD erhöht sich um 5; d.h. in jeder DATA-Zeile stehen später fünf Werte (A0 bis A4).

Nach RUN hören Sie, wie die Disketten-Station arbeitet. Wenn das vertraute READY auf dem Bildschirm erscheint, geben Sie folgendes ein:

```
NEW                  (RETURN)
ENTER "D:DATAGEN     (RETURN)

RUN                  (RETURN)
```

Auf dem Bildschirm erscheinen nun DATA-Zeilen von 1000 bis 1090 mit den Internen Code-Werten der Zeichen der Speicherstellen 40000 bis 40050. In unserem Beispiel waren das die Daten, die Sie nachfolgend sehen. Ihre eigenen können sich davon natürlich unterscheiden.

```
1000 DATA 0,0,18,16,0
1010 DATA 47,48,37,46,0
1020 DATA 3,17,12,24,12
1030 DATA 16,12,2,36,26
1040 DATA 36,33,52,33,39
1050 DATA 37,46,2,0,0
1060 DATA 0,0,0,0,0
1070 DATA 0,0,0,0,0
1080 DATA 0,0,19,16,0
1090 DATA 58,46,29,17,16
```

Mit dem folgenden Hilfsprogramm können Sie sich ausPRINTen lassen, was die DATAs zu bieten haben. Der Zusatz +32 in Zeile 30 ist notwendig, um den internen Code in ASCII-Code umzuwandeln. Im Kapitel Zeichensatz wurde beschrieben, wie der interne Code zum ASCII-Code blockweise verschoben ist.

```
10 TRAP 100
20 READ D
30 PRINT CHR$(D+32);
40 GOTO 20
100 END
```

5.3.3 Directory ohne DOS

Nicht selten kommt es vor, daß man ein Programm auf Diskette abspeichern will und nicht sicher ist, ob ein anderes Programm mit gleichem Namen bereits existiert. Ein Blick in die Directory bringt zwar Klarheit; aber mit dem Aufruf von DOS wird das im Speicher befindliche Programm gelöscht. Von MEM.SAV war bereits die Rede und auch davon, daß der Ladevorgang des DOS und die anschließende Rückbringung des zwischengespeicherten Programms relativ lange dauert.

Es gibt jedoch die Möglichkeit, die Directory ohne Aufrufen von DOS auslisten zu lassen. Dazu das folgende Programm:

```
30000 CLOSE #1:CLR :DIM XYZ$(17):OPEN #1,6,0,"D:*.*":
FOR X=0 TO 64:INPUT #1;XYZ$:PRINT XYZ$:NEXT X
30001 CLOSE #1:CLR :DIM ZYX$(17):OPEN #1,6,0,"D:*.*":
FOR X=0 TO 64:INPUT #1;ZYX$:LPRINT ZYX$:NEXT X
```

Falls Sie keinen Drucker benutzen, genügt es, nur die Zeile 30000 einzugeben. Durch CLR werden alle Variablen-Werte auf 0 gesetzt. Dies ist notwendig, falls Sie in Ihrem aktuellen Programm die gleichen Variablen-Namen benutzen wie in diesem Hilfs-Programm. Anstelle von XYZ$ bzw. ZYX$ können Sie auch andere Namen nehmen. Mit der Betriebsart 6 im ersten Parameter des OPEN-Befehls kann die Directory gelesen

werden (vergl. Tabelle Seite 138). Durch Verwendung der "wild cards" D:*.* werden alle Filenamen der Diskette gelesen. Insgesamt können sich 64 Namen auf einer Diskette befinden. Damit auch die Zahl der freien Sektoren ausgelistet wird, macht die FOR-NEXT-Schleife 65 Durchläufe. Dabei nimmt XYZ$ bzw. ZYX$ den aktuellen Filenamen an, der auf dem Bildschirm oder vom Drucker ausgegeben wird.

Sie können das Programm z.B. unter dem Namen DIRECTOR auf jede Diskette im LIST-Format abspeichern und jederzeit zu Ihrem aktuellen Programm hinzuENTERn. Wie Sie wissen, wird dadurch Ihr Programm nicht gelöscht (außer, wenn Sie die Programmzeilen 30000 und 30001 belegt haben).

Wenn Sie die Directory auf dem Bildschirm sehen wollen, geben Sie ein: GOTO 30000. Zur Ausgabe auf dem Drucker geben Sie ein: GOTO 30001. Falls sich weniger als 64 Filenamen auf der Diskette befinden, werden Sie nach dem Auslisten eine ERROR-Meldung auf dem Bildschirm lesen. Das hat allerdings keine weitere Bedeutung. Dem Rechner fehlen lediglich Namen, die durch die FOR-NEXT-Schleife aufgerufen werden.

5.4 Disketten-Operationen mit GET# und PUT#

GET# und PUT# unterscheiden sich von PRINT# und INPUT# dadurch, daß keine Datenketten, sondern immer nur ein einzelnes Byte vom angesprochenen Gerät gelesen bzw. an dieses abgegeben wird (z.B. ein einzelner ASCII-Wert). Für einige Disketten-Operationen ist dies sehr nützlich, wie die folgenden Beispiele zeigen.

5.4.1 AUTONUMBER-Funktion

Vom DATA-Zeilen-Generator bis zur automatischen Zeilennummerierung für BASIC-Programme ist es nur ein kleiner Schritt. Mit dem folgenden AUTONUMBER-Programm können Sie bequem BASIC-Programme schreiben; denn die Zeilen-Nummerierung erfolgt automatisch. Sie geben die gewünschte Anfangszeile und den Zeilenabstand ein und können das Programm jederzeit durch Drücken von HELP unterbrechen.

Im Gegensatz zu den meisten anderen AUTONUMBER-Programmen ist es hierbei möglich, nach einer Unterbrechung mit der automatischen Zeilen-Nummerierung fortzufahren. Sie müssen nur die Zeile angeben, die als nächstes erzeugt werden soll. Auf diese Weise können Sie Ihr BASIC-Programm in logische Zeilen-Blöcke unterteilen (z.B. erstes Unterprogramm ab Zeile 1000, zweites Unterprogramm ab 2000 usw.).

Selbstverständlich dürfen Sie keine Programmzeilen generieren, die das AUTONUMBER-Programm überschreiben. Um dies praktisch zu vermeiden, wurde die Utility an das Ende der erlaubten Zeilen-Nummern gelegt (32767 ist die letzte zulässige BASIC-Zeile).

```
32760 CLR :DIM A$(255),B$(17):B$="D:":? "Programm-Nam
e?":INPUT A$:B$(LEN(B$)+1)=A$:? "Neu(8)/Anhang(9)?":I
NPUT A
32761 ? "Beginn mit Zeile?":INPUT ZN:? "Zeilen-Abstan
d?":INPUT ZAB:OPEN #1,A,0,B$:POKE 732,0:? CHR$(125)
32762 OPEN #2,4,0,"K:"
32763 ? ZN;:PRINT #1;ZN;:ZN=ZN+ZAB
32764 GET #2,A:? CHR$(A);:PUT #1,A:IF A<>155 THEN 327
64
32765 IF PEEK(732)=17 THEN 32767
32766 IF A=155 THEN 32763
32767 PRINT #1;32759;" END":CLOSE #1:CLOSE #2:ENTER B
$
```

32760: CLR löscht alle Variablen-Werte. B$ ist zunächst "D:" (Anfang eines Disketten-Filenamens). Ihre Eingabe A$ wird mit B$ verkettet, so daß ein kompletter Dateiname entsteht (z.B. D:TESTPROG.LST). Danach wird gefragt, ob Sie eine neue Datei eröffnen oder Programm-Zeilen an eine bestehende anhängen wollen (Eingabe 8 oder 9).

32761: Zeilen-Nummer (ZN) und Zeilen-Abstand (ZAB) werden abgefragt. Danach wird der Datenkanal zur Disketten-Station eröffnet und der Dateiname (B$) abgelegt. POKE 732,0 löscht das Register, mit dem das Betätigen der HELP-Taste abgefragt werden kann. CHR$(125) löscht den Bildschirm.

32762: Der Datenkanal zur Tastatur wird eröffnet (vergl. Tabelle Seite 138).

32763: Auf dem Bildschirm erscheint die erste Zeilen-Nummer des BASIC-Programms, das Sie generieren möchten. Gleichzeitig wird diese Nummer in Form von ASCII-Werten auf Diskette abgelegt (besser gesagt: Es werden jeweils 128 Byte in einem Puffer gesammelt, bevor diese Werte auf Diskette gelangen). Mit ZN=ZN+ZAB erhöhen sich die Zeilen-Nummern um die von Ihnen eingegebenen Werte.

32764: GET#2 registriert Ihre Tastatur-Eingabe in Form von ASCII-Werten. Mit CHR$(A) erscheinen diese Werte als normal lesbare Zeichen auf dem Bildschirm. Hier schreiben Sie also Ihr eigentliches BASIC-Programm. Gleichzeitig wird jeder Wert auf Diskette gePUTtet. Korrekturen sind zunächst nicht mehr möglich. Fehler machen sich erst beim ENTERn bemerkbar (Zeile 32767), nachdem Sie das Programm unterbrochen oder beendet haben. Solange A nicht 155 (EOL) ist, (ASCII-Code von RETURN), bleibt der Rechner in dieser Zeile.

32765: Wenn die HELP-Taste gedrückt wird, geht der Rechner zur Zeile 32767.

32766: Wenn RETURN gedrückt wurde, geht der Rechner zur Zeile 32763, wo die nächste Zeilen-Nummer generiert wird.

32767: Nach dem Drücken der HELP-Taste wird automatisch eine Zeilen-Nummer 32759 generiert, die den BASIC-Befehl END aufnimmt. Danach werden die Datenkanäle geschlossen und das von Ihnen geschriebene BASIC-Programm mit ENTER B$ in den Rechner geladen. Sie können das Programm dann ganz normal mit RUN starten. Durch die Zeile 32759 gelangt der Rechner nie in das AUTONUMBER-Programm. Falls Sie Fehler feststellen oder Änderungen vornehmen, können Sie anschließend das veränderte Programm mit der Anweisung LIST B$ auf der Diskette abspeichern.

5.4.2 Textverarbeitung mit AUTONUMBER

Selbstverständlich ist es möglich, die AUTONUMBER-Funktion mit anderen Aufgaben zu kombinieren; z.B. als einfaches Text-System. Das folgende Listing ist freilich von "WordStar" weit entfernt; aber mit ihm können einzelne Anwendungen, wie das Schreiben eines Briefes, erleichtert werden.

```
1 DIM T$(255),N$(17):POKE 732,0:ZN=90:N$="D:":? "Prog
ramm-Name ":INPUT T$:N$(LEN(N$)+1)=T$
2 CLOSE #1:OPEN #1,8,0,N$
3 ZN=ZN+10:INPUT T$:PRINT #1;ZN;" LPRINT ";T$;CHR$(155)
4 IF ZN=500 THEN CLOSE #1:ENTER N$
5 IF PEEK(732)=17 THEN CLOSE #1:ENTER N$
6 GOTO 3
7 DIM A$(50),B$(50),C$(50),D$(50)
8 LPRINT CHR$(27);CHR$(66);CHR$(5)
9 LPRINT CHR$(27);CHR$(66);CHR$(2)
10 LPRINT CHR$(27);CHR$(97);CHR$(3)
11 LPRINT CHR$(27);CHR$(77);CHR$(4)
12 ? "Vorname, Name:      ";:INPUT A$
13 ? "Zusatz:             ";:INPUT B$
14 ? "Strasse, Postfach:  ";:INPUT C$
15 ? "Ort:                ";:INPUT D$
16 LPRINT A$:LPRINT :LPRINT B$:LPRINT :LPRINT C$:LPRI
NT :LPRINT D$
17 LPRINT CHR$(27);CHR$(97);CHR$(5)
```

1: In diesem Programm werden nicht die einzelnen Zeichen in Form von ASCII-Werten geGETtet, sondern jede Textzeile als String registriert und mit PRINT an die Diskette abgegeben. Sie können also Ihren Text beliebig ändern, solange Sie nicht RETURN drücken. T$ (Text-String) und N$ (Name des Programms) werden DIMensioniert. Das Register 732 (HELP-Taste) wird auf 0 gesetzt. ZN ist die Variable für die Zeilen-Nummer. So wie bei AUTONUMBER bereits erklärt, wird der von Ihnen eingegebene Programm-Name mit N$ verkettet.

2: Der Datenkanal zur Disketten-Station wird eröffnet und der betreffende Programm-Name registriert.

3: Nach jedem RETURN erhöht sich die Zeilen-Nummer automatisch um 10 (die Zeilen-Nummer erscheint nicht auf dem Bildschirm; Sie können sich ganz auf den Text konzentrieren). Mit INPUT T$ können Sie den gewünschten Text eingeben (in diesem Fall ein Brief). Danach wird die aktuelle Zeilen-Nummer und der eingegebene Text in Form von ASCII-Zeichen auf Diskette abgelegt. Als Zeilenabschluß steht das bereits erklärte EOL (ASCII-Code 155).

4: Wenn die Zeilen-Nummer 500 erreicht ist, wird das Programm automatisch beendet und der Text geENTERt. Mit der Zeile 500 sind insgesamt 50 Textzeilen erzeugt; d.h. ein DIN A 4-Blatt dürfte, je nach Zeilenabstand des Druckers, damit voll sein. Die Zeile 4 ist also ein Schutzroutine, auf die Sie natürlich auch verzichten können.

5: Wenn Sie HELP (RETURN) drücken, wird das Programm beendet und der Text von der Diskette in den Rechner geladen.

6: Solange keine der Bedingungen aus den Zeilen 4 und 5 erfüllt sind, kehrt der Rechner zur Zeile 3 zurück.

7: In diesen Programmteil gelangt der Rechner nicht automatisch, sondern nur durch Ihre Anweisung: Sobald das Programm beendet ist (in diesem Beispiel der Brief fertig geschrieben ist), wird das unter N$ abgespeicherte Programm mitsamt den automatisch generierten Zeilen-Nummern in den Rechner geladen. Wenn Sie nun LIST (RETURN) eingeben, sehen Sie sowohl das Text-Schreibe-Programm als auch den Text selbst in Form von LPRINT-Anweisungen (LPRINT = Ausgabe an den Drucker). In Zeile 7 werden nun die Eingaben für die Adresse des Briefempfängers DIMensioniert. Falls Sie die Utility für ganz andere Texte verwenden wollen, können die folgenden Zeilen natürlich fortgelassen werden bzw. Ihren Bedürfnissen entsprechend verändert werden.

8 bis 11: Diese Anweisungen an den Drucker bestimmen die Schrifttype, Zeilenformat, Randbegrenzung usw. Sie sind in diesem Fall auf den "star radix 10" abgestimmt und können von den Angaben für andere Modelle abweichen.

12 bis 15: Hier wird die Anschrift des Briefempfängers abgefragt.

16: Die Anschrift wird gedruckt.

17: Eine weitere Anweisung für den Drucker, mit der er fünf Zeilen vorrückt, bevor der eigentliche Brieftext beginnt.

Wenn Sie das Programm mit RUN starten, erscheint zunächst die Frage:

```
Programm-Name ?
```

Wenn Sie den gewünschten Namen eingegeben haben, öffnet der Rechner den Datenkanal zur Disketten-Station. Auf dem Bildschirm sehen Sie dann:

```
?
```

Nun müssen Sie Anführungsstriche eingeben und können dann Ihren Text schreiben. Natürlich wäre es auch möglich, die

Anführungsstriche ebenso wie die Zeilen-Nummer und LPRINT gleich mitgenerieren zu lassen. Das hat jedoch einen Nachteil: Wenn Sie statt Text Steuerfunktionen an den Drucker geben wollen (mit CHR$-Anweisungen), so geht dies nur ohne Anführungszeichen fehlerfrei.

Nachdem der gewünschte Text eingegeben und das Programm geENTERt ist, tippen Sie GOTO 7 (RETURN). Danach können Sie die Anschrift eingeben. Der Brieftext wird automatisch gedruckt.

Es kann vorkommen, daß sich während der Texteingabe Fehler einschleichen. Beim ENTERn erscheinen dann ERROR-Zeilen auf dem Bildschirm, die Sie einfach korrigieren müssen, bevor Sie GOTO 7 eingeben.

5.4.3 Disketten

Ein paar Worte zu den Disketten selbst, denen Sie Ihre wertvollen Programme oder Textfiles anvertrauen. Genau wie bei Tonband-Cassetten werden viele Disketten-Sorten von verschiedenen Herstellern angeboten.

Bei der Ausarbeitung dieses Buches wurden z.T. erhebliche Qualitäts-Unterschiede von einzelnen Disketten-Marken festgestellt. So war es z.B. notwendig, Disketten dreimal zu formatieren, bevor sie Daten bzw. Programme speicherten. Diese unangenehme Erfahrung sollten Sie sich ersparen. Folgende Disketten-Marken (in alphabetischer Reihenfolge) brachten bei unseren Tests zufriedenstellende Ergebnisse: Dysan, Inmac, magna, maxell, memorex, Xidex.

5.5 AUTORUN

Das Disk Operating System bietet neben den bisher gezeigten Disketten-Tricks noch einen besonderen Leckerbissen: Wenn sich ein Binärfile mit dem Namen AUTORUN.SYS auf der Diskette befindet, wird der in ihm enthaltete Befehl beim Einschalten des Computers automatisch durchgeführt werden.

Auf diese Weise kann z.B. ein bestimmtes Programm gleich nach dem Einschalten des ATARI gestartet werden (was auch einen gewissen LIST-Schutz bietet). Ebenso ist es möglich, mit AUTORUN.SYS zum Programmier-Beginn die Bildschirm-Farbe zu wechseln, einen Sound zu erzeugen oder eine Druckersteuerung vorzunehmen. Das folgende Listing dient als Grundlage für Ihr eigenes AUTORUN.SYS-Programm:

```
100 DATA 255,255,106,6,255,6,169,80,141,0,3,169,1,141
,1,3,169,63,141,2
110 DATA 3,169,64,141,3,3,169,5,141,6,3,141,5,3,169,0
,141,4,3,141
120 DATA 9,3,141,10,3,141,11,3,169,12,141,8,3,32,89,2
28,16,1,96,162
130 DATA 11,189,0,5,157,0,3,202,16,247,32,89,228,48,6
,32,6,5,108,12
140 DATA 0,96,169,193,141,33,3,169,6,141,34,3,96,251,
243,51,246,207,6,163
150 DATA 246,51,246,60,246,76,228,243,172,255,6,240,9
,185,234,6,206,255,6,160
160 DATA 1,96,140,33,3,169,228,141,34,3,169,155,160,1
,96
170 DATA 34,48,49,55,69,75,79,80,58,68,34,32,78,85,82
,32,32,32,32,32
180 DATA 20,226,2,227,2,106,6,224,2,225,2,180,6
190 TRAP 220
200 OPEN #1,8,0,"D:AUTORUN.SYS"
210 READ A:PUT #1,A:GOTO 210
220 END
```

Die DATA-Zeilen 100 bis 160 sowie die Zeile 180 stellen ein Maschinensprache-Programm dar, das Bestandteil jedes AUTORUN.SYS-Programms ist und nicht verändert wird. Interessant ist hingegen die Zeile 170; die hier enthaltenen Werte können von Ihnen bestimmt werden. Sie stellen die ASCII-Werte des Befehls dar, der automatisch gestartet werden soll.

In diesem Beispiel handelt es sich um den Befehl RUN "D:POKE710". Sie können sich zum besseren Verständnis folgendes Hilfs-Programm schreiben, mit dem die ASCII-Werte lesbar auf dem Bildschirm erscheinen:

```
10 TRAP 30
20 READ A:? CHR$(A);:GOTO 20
30 END
170 DATA 34,48,49,55,69,75,79,80,58,68,34,32,78,85,82
```

Nach RUN sehen Sie:

```
017EKOP:D" NUR
```

Das Ganze wird für Sie deutlicher, wenn Sie den Text von hinten lesen. In dieser Form können Sie jeden beliebigen Programm-Namen eingeben (außer AUTORUN.SYS selbst) und beim Einschalten des Rechners automatisch starten lassen.

Die betreffende DATA-Zeile (in diesem Beispiel Zeile 170) muß grundsätzlich 20 Daten enthalten. AUTORUN.SYS bewirkt sonst eine Fehlfunktion, die meist zum "Absturz" des Computers führt. Werte, die nicht vom Programm-Namen beansprucht werden, können Sie als "Leerzeichen" eingeben (ASCII-Wert 32).

Natürlich muß sich das Programm, das Sie automatisch starten wollen, ebenfalls auf der Diskette befinden. In diesem Beispiel also das Programm POKE710. Es sorgt für einen Farbwechsel des Bildschirms. Hier das Listing:

```
10 REM AUTORUN.SYS - DEMO
20 REM BILDSCHIRM-FARBE WECHSELN
30 POKE 710,194
```

Das Programm muß unter dem Namen POKE710 auf Diskette geSAVEd werden (geENTERte Programme können nicht mit RUN "D:NAME gestartet werden). Es ist jedoch auch möglich, ein Programm mit AUTORUN.SYS zu ENTERn; d.h. es wird nur in den Rechner geladen, aber nicht gestartet. In diesem Fall müssen die Daten in Zeile 170 die ASCII-Werte von ENTER "D:POKE710" enthalten (ebenfalls rückwärts). Das Programm selbst muß mit LIST "D:POKE710" abgespeichert sein (geSAVEte Programme können nicht geENTERt werden). Nachfolgend finden Sie die geänderte DATA-Zeile:

```
170 DATA 34,48,49,55,69,75,79,80,58,68,34,32,82,69,84
,78,69,32,32,32
```

Falls Sie, wie hier in diesem Beispiel, nur einen einzigen Befehl ausführen wollen, können Sie auch auf den Aufruf eines Programms verzichten und den Befehl direkt in die DATA-Zeile übersetzen (ebenfalls rückwärts). 170 sieht dann folgendermaßen aus:

```
170 DATA 52,57,49,44,48,49,55,32,69,75,79,80,32,32,32
,32,32,32,32,32,32
```

6 Schutz

Seit es Menschen gibt, gibt es Diebe. Stehlen ist das älteste Gewerbe der Welt (im Gegensatz zu einer anderen weitverbreiteten Ansicht). - Kein Wunder, daß sich das räuberische Treiben auch im Computer-Zeitalter fortsetzt.

Unter den Sammelbegriffen "Software-Klau" und "Raubkopierer" versuchen arglistige Gentlemen , sich auf billige Weise an Programmen zu bereichern, die pfiffige Tüftler in wochenlangen Nachtsitzungen mit kalten und warmen Schwitzbädern zusammengebastelt haben.

Als Folge davon begann ein elektronisches Wettrüsten um die besten Schutzmaßnahmen gegen Software-Klau. Eines wurde schnell klar: Einen absoluten Schutz gibt es nicht. Man kann jedoch den Programm-Räubern die "Arbeit" reichlich schwer machen. Besonders "gemein" sind Schutzroutinen, die sich erst im Laufe der Benutzung eines raubkopierten Programms bemerkbar machen; z.B. wenn ein Textverarbeitungs-Programm erst abstürzt, nachdem der Text eingegeben wurde oder Kalkulations-Programme, die ziemlich sonderbare Prognosen ausgeben.

Von solch komplizierten Schutzmaßnahmen handelt dieses Kapitel allerdings nicht. Vielmehr geben wir Ihnen Schutz-Tips für den Hausgebrauch (Vollprofis brauchen dieses Buch ohnehin nicht).

Aber falls Sie Programme haben, die Sie z.B. Software-Firmen anbieten wollen, ist es ganz angebracht, die Listings vor fremden Blicken zu schützen, indem Sie mit AUTORUN starten und die Directory auf der Diskette verlegen, damit die Datenfiles nicht über DOS-Funktion kopiert werden können. Dazu können Sie durch Zerstören des Statement-Zeigers oder der Variablen-Tabelle einen zusätzlichen Listschutz einbauen und die BREAK-Taste außer Funktion setzen.

Eine Warnung sei gleich vorweg gegeben: Benutzen Sie zum Testen der hier gezeigten Schutzroutinen eine Experimentier-Diskette und sichern Sie Ihre zu schützenden Programme vorher noch auf einer anderen Diskette. Sie können sonst das Pech haben, daß Sie den Zugriff auf Ihre eigenen Listings unwiederbringbar verbauen.

6.1 Listschutz

Listschutz ist noch kein Kopierschutz. Trotzdem kann ein einfacher Listschutz im Zusammenhang mit den anderen hier gezeigten Schutzroutinen ein wirksames Mittel gegen unerwünschten Zugriff auf Ihre Programme sein.

Es gibt verschiedene Methoden, um einen Listschutz zu erzeugen. Wir zeigen Ihnen nachfolgend zwei Versionen: Das Zerstören des aktuelles Statement-Zeigers und das Verändern der Variablen-Tabelle.

In beiden Fällen können Programme zwar gestartet, aber nicht geLISTet werden. Beim Zerstören des Statement-Zeigers führt der LIST-Versuch zum Absturz des Computers; bei Veränderung der Variablen-Tabelle erscheint nach der LIST-Anweisung auf Wirrwar unentzifferbarer Zeichen.

6.1.1 Statement-Zeiger zerstören

So wie die Register 130 und 131 den Zeiger auf die Variablen-Tabelle enthalten (vergl. 1.5 "Die Variablen-Tabelle"), so gibt es auch einen Zeiger auf das aktuell bearbeitete Programm-Statement. Dieser Zeiger ist in den Adressen 138 und 139 enthalten.

Wenn Sie in die zweite Adresse hinter dem aktuellen Statement-Zeiger eine 0 einPOKEn und Ihr Programm mit gleicher Anweisung abspeichern, erhalten Sie einen wunderbaren Listschutz. Das folgende Listing zeigt das Grundprinzip:

```
10 REM NAME.EXT
20 PRINT "Dies ist ein Demo-Programm"
30 POKE PEEK(138)+PEEK(139)*256+2,0:SAVE "D:NAME.EXT"
:NEW
```

In Zeile 30 wird durch LO+HI*256 die Adresse für den aktuellen Statement-Zeiger ermittelt. In die zweite Adresse dahinter wird dann eine 0 eingePOKEt. Danach erfolgt die Abspeicherung des Programms auf Diskette und eine NEW-Anweisung. Sie können das Programm nun mit RUN "D:NAME.EXT" starten. Eine LOAD-Anweisung oder LIST führen zum Absturz des Rechners.

Wenn Sie diese Schutzroutine in Ihrem eigenen Programm einsetzen wollen, ist es wichtig, daß Sie dabei eine Zeilen-Nummer verwenden. Ohne Zeilen-Nummer bleibt die Anweisung wirkungslos. Schreiben Sie z.B. 30000 POKE PEEK...usw. und geben Sie danach GOTO 30000 (RETURN) ein. Sichern Sie Ihr Programm jedoch vorher ganz normal auf einer anderen Dis-

kette mit SAVE "D:NAME.EXT" oder LIST "D:NAME.EXT", damit Ihnen das Listing für Ihre eigene Arbeit erhalten bleibt.

6.1.2 Variablen-Tabelle verändern

Im Kapitel 1.5 wurde bereits ausführlich auf die Variablen-Tabelle eingegangen. Es ist jedoch nicht nur möglich, durch POKE-Anweisungen einzelne Variablen-Namen zu verändern, sondern auch einen wirksamen Listschutz zu erzeugen.

Zu diesem Zweck müssen in die einzelnen Adressen der Tabelle irgendwelche Dezimalwerte gePOKEt werden. Im untenstehenden Beispiel wurde jedes Register mit 0 belegt. Sie können jedoch auch andere Werte von 0 bis 255 nehmen.

```
1 X=PEEK(130)+PEEK(131)*256:Y=PEEK(132)+PEEK(133)*256
:FOR Z=X TO Y:POKE Z,0:NEXT Z
10 DIM A$(10),B$(10),C$(20)
20 A$="Dies ist "
30 B$="ein "
40 C$="Demo-Programm"
50 XY=1234
60 YZ=2345
70 VARIABLE=XY+YZ
80 ? A$;B$;C$
90 ? VARIABLE
```

Die Zeile 1 ist das eigentliche Listschutz-Programm. Die Register 130 und 131 enthalten die erste Adresse der Variablen-Tabelle; die Register 132 und 133 die letzte Adresse (jeweils LO-Byte und HI-Byte). Durch die FOR-NEXT-Schleife werden alle Adressen der Variablen-Tabelle mit 0 belegt. Um die Wirkung anschaulich zu demonstrieren, enthält das Beispiel-Programm eine Reihe von Phantasie-Variablen.

Bevor Sie RUN (RETURN) eingeben, müssen Sie das Programm abspeichern! Sobald die Variablen-Tabelle durch die Anweisung in Zeile 1 zerstört ist, können Sie das Programm nicht mehr sichern.

Es ist natürlich auch möglich, das Zerstören der Variablen-Tabelle an den Schluß eines Programms zu legen (z.B. in Zeile 30000 wie im vorhergehenden Beispiel-Programm).

Versuchen Sie nun, nachdem Sie Ihr Programm gesichert haben, nach RUN (RETURN) eine LIST-Anweisung zu geben. Sie werden eine Menge unsinniger Buchstaben und Grafikzeichen auf dem Bildschirm lesen. Das verlorene Listing danach zu rekonstruieren, dürfte sehr mühsam sein.

6.2 Programmschutz

Selbst wenn ein Programm nicht mehr gelistet werden kann, so ist es doch möglich, mit Hilfe der DOS-Funktion C (Copy) ein Dublikat anzufertigen.

In diesem Abschnitt finden Sie u.a. eine Schutzroutine, mit der die Directory der Diskette auf andere Sektoren verlegt wird. Dadurch kann das Disk Operating System bei Aufruf der Copy-Funktion den angesprochenen Filenamen nicht finden und demnach auch nicht kopieren.

Daneben zeigen wir noch einige andere Möglichkeiten des Programmschutzes; z.B. das Sperren der BREAK-Taste oder das erneute Booten (Laden des Disketten-Betriebs-Systems), sobald die Taste RESET gedrückt wird, die normalerweise jedes Programm unterbricht.

6.2.1 Automatisches NEW

Normalerweise kann man durch BREAK oder RESET jedes Programm unterbrechen. Man kann diese Tasten jedoch auch mit anderen Funktionen belegen oder ganz außer Betrieb setzen.

Wenn Sie in ein Programm die Anweisung:

```
POKE 580,1
```

einbauen, kann nach RUN (RETURN) das Programm mit RESET zwar unterbrochen werden; sobald jedoch ein LIST (RETURN) eingegeben wird, meldet sich nicht mehr als ein freundliches READY auf dem Bildschirm.

Dieser POKE-Befehl sorgt nämlich für ein automatisches NEW. Der Normalwert dieses Registers ist 0.

6.2.2 Automatisches Booten

Anstelle eines NEW kann das Drücken von RESET auch eine andere Funktion auslösen; und zwar das erneute Booten des DOS, das normalerweise nur nach Einschalten des Rechners (mit angeschlossener Disketten-Station) erfolgt.

Durch diese Maßnahme erreichen Sie ebenfalls einen gewissen Schutz für Ihr Programm. Die notwendige Anweisung dazu lautet:

```
POKE 202,1
```

Der Normalwert dieses Registers ist 0.

6.2.3 BREAK-Taste sperren

Die bislang unter 6.2 gezeigten kleinen Tips zum Schutz Ihrer Programme berücksichtigen nur die RESET-Taste. Da man aber auch durch Drücken von BREAK normalerweise jedes Programm unterbrechen kann, muß auch für diese Taste eine Manipulatons-Möglichkeit geschaffen werden.

Dies gelingt durch die Anweisung:

```
POKE 16,64:POKE 53774,64
```

Die BREAK-Taste wird hierdurch außer Funktion gesetzt. Sie können das mit folgendem Experimentier-Programm einmal ausprobieren:

```
10 POKE 16,64:POKE 53774,64
20 ? "BREAK-Taste gesperrt"
30 ? :? "Wenn Sie die Taste <A> druecken,"
35 ? "koennen Sie wieder mit BREAK unter-   brechen"
40 IF PEEK(764)=63 THEN GRAPHICS 0:GOTO 60
50 GOTO 40
60 ? "BREAK-Taste funktioniert wieder"
70 GOTO 70
```

Es ist wichtig, diese POKE-Anweisung nach jedem GRAPHICS-Befehl in Ihrem Programm zu wiederholen. Denn mit dem Aufruf einer anderen Grafik-Betriebsart werden die beiden Register wieder in Normalzustand versetzt (16,192 und 53774,247).

6.2.4 Directory verlegen

Im Register 4226 ist festgelegt, auf welchem Sektor der Diskette die Directory, also das Inhaltsverzeichnis der Diskette beginnt. Normalerweise steht in 4226 eine 105. Durch Ändern dieses Wertes ändert sich auch der Sektor des Beginns der Directory. Damit erreichen Sie einen wirksamen Schutz Ihres Programms.

Selbst wenn jemand den Namen des Programms kennt, wird es ihm nicht möglich sein, es zu laden oder zu kopieren. Vor Betriebsbeginn muß nämlich in 4226 der von Ihnen bestimmte Wert abgelegt werden. Das geschieht am besten durch eine AUTORUN-Routine (s. 6.3).

Wenn man ein Programm in einer verlegten Directory abspeichert, wird DUP.SYS außer Funktion gesetzt, da ein zu kopierendes File für den Rechner nicht auffindbar ist.

Sie sollten jedoch vor Anwendung dieser Schutzroutine folgende

Punkte genau beachten, da Sie sonst in die berühmte "Teufelsküche" kommen (und da kann Sie niemand mehr schützen).

1. POKE 4226,n (n = 0 bis 255, jedoch nicht 105)
2. Das zu schützende Programm schreiben
3. SAVE "D:NAME.EXT (RETURN)
4. NEW (RETURN)
5. POKE 4226,105

Wenn Sie nun DOS aufrufen und die Directory mit A (RETURN) (RETURN) auslisten wollen, werden Sie den Programm-Namen nicht finden. Sie bekommen lediglich angezeigt, wieviele Sektoren noch frei sind.

Gehen Sie nun mit B (RETRUN) zurück in den Rechner und tippen Sie POKE 4226,n (n = Ihr Wert). Versuchen Sie dann DOS aufzurufen; mehr als ein READY wird Ihr ATARI nicht zustande bringen.

Sie können Ihr Programm nur noch mit RUN "D:NAME.EXT" starten oder mit LOAD laden.

6.3 Schutzroutine mit AUTORUN

Alles was bisher gezeigt wurde, sind im Grunde nur Teil-Schutzmaßnahmen, die für sich allein noch keinen unerwünschten Zugriff auf Ihr Programm verhindern. Nachfolgend finden Sie daher eine Routine, bei der zunächst mit AUTORUN.SYS das Register 4226 verändert wird; danach wird zum Start Ihres Programms die BREAK-Taste gesperrt und RESET auf erneutes Booten programmiert. Wenn Sie zusätzlich noch einen Listschutz (s. 6.1.1) in Ihr eigentliches Programm einbauen, dürften Ihre programmierten Mühen vor allzu neugierigen Blicken gut geschützt sein (wie bereits gesagt: Einen absoluten Schutz gibt es nicht).

Die Diskette, auf der Sie die folgende Routine anwenden, sollte keine anderen Programme enthalten; Sie können dann einfach mit RUN "D:*.*" starten. Falls Sie trotzdem mehrere Programme abspeichern wollen, müssen Sie hinter dem Schutzprogramm liegen, da mit RUN "D:*.*" das erste Programm auf der Diskette gestartet wird.

```
100 DATA 255,255,106,6,255,6,169,80,141,0,3,169,1,141
,1,3,169,63,141,2
110 DATA 3,169,64,141,3,3,169,5,141,6,3,141,5,3,169,0
,141,4,3,141
120 DATA 9,3,141,10,3,141,11,3,169,12,141,8,3,32,89,2
28,16,1,96,162
130 DATA 11,189,0,5,157,0,3,202,16,247,32,89,228,48,6
,32,6,5,108,12
140 DATA 0,96,169,193,141,33,3,169,6,141,34,3,96,251,
243,51,246,207,6,163
150 DATA 246,51,246,60,246,76,228,243,172,255,6,240,9
,185,234,6,206,255,6,160
160 DATA 1,96,140,33,3,169,228,141,34,3,169,155,160,1
,96
170 DATA 53,53,50,44,54,50,50,52,32,69,75,79,80,32,32
,32,32,32,32,32
180 DATA 20,226,2,227,2,106,6,224,2,225,2,180,6
190 TRAP 220
200 OPEN #1,8,0,"D:AUTORUN.SYS"
210 READ A:PUT #1,A:GOTO 210
220 END
```

Mit dieser AUTORUN.SYS-Routine wird beim Einschalten des Rechners (bei angeschlossener Disketten-Station) der Befehl POKE 4226,255 automatisch ausgeführt.

Der Befehl ist in DATA-Zeile 170 enthalten. Falls Sie einen anderen Wert POKEn wollen, beachten Sie bitte die ausführ-

liche Beschreibung im Abschnitt 5.5, Seite 150. Geben Sie nun RUN (RETURN) ein, damit das AUTORUN.SYS-File auf Diskette geschrieben wird. Danach tippen Sie NEW (RETURN) und POKE 4226,255 (oder ggf. einen anderen Wert). Schreiben Sie dann folgenden Programm:

```
10 POKE 580,1:REM Erneutes Booten
20 POKE 16,64:POKE 53774,64:REM BREAK sperren
30 RUN "D:NAME.EXT"
```

Sofern sich noch kein anderes Programm auf Ihrer Diskette befindet, können Sie dieses Programm unter irgendeinem Namen abspeichern. Es wird dann später durch RUN "D:*.*" automatisch gestartet. In Zeile 30 müssen Sie den Namen Ihres eigentlichen BASIC-Programms schreiben.

In dieses Programm können Sie dann wiederum Schutzmaßnahmen einbauen; z.B. einen Listschutz, wie unter 6.1.1 gezeigt.

```
10 REM NAME.EXT
20 REM
30 REM
40 REM Hier folgt Ihr Programm
50 REM
60 REM
30000 POKE PEEK(138)+PEEK(139)*256+2,0:SAVE "D:NAME.E
XT":NEW
```

Speichern Sie das Programm, indem Sie GOTO 30000 (RETURN) eingeben. Schalten Sie danach den Rechner aus und wieder ein. Neben dem Boot-Vorgang wird auch AUTORUN.SYS ausgeführt. Wenn Sie RUN "D:*.*" tippen, wird zuerst das kurze Schutzprogramm und dann Ihr eigentliches BASIC-Programm gestartet. Sie können das Programm nun weder mit BREAK noch mit RESET unterbrechen. Selbst wenn Sie das DOS einer anderen Diskette laden (was zunächst nicht geht, da das Register 4226 verändert ist), ist es nicht möglich, an das Programm heranzukommen. Sie können das nach Herzenslust einmal versuchen.

Nun noch einmal die einzelnen Arbeitsschritte, die Sie genau befolgen sollten, da Sie sonst möglicherweise den Zugriff auf Ihr eigenes Programm verlieren.

1. Normales Formatieren einer Diskette (falls noch nicht gegeschehen)
2. WRITE DOS-FILE: H (RETURN) 1 (RETURN) Y (RETURN)
3. Mit B (RETURN) zurück in den Rechner
4. AUTORUN.SYS schreiben wie vorhergehende Seite (oder mit verändertem Dezimalwert)

5. RUN (RETURN)
6. NEW (RETURN)
7. Ohne Zeilen-Nummer: POKE 4226,255 (oder einem anderen, von Ihnen gewählten Dezimalwert)
8. Das kurze Schutz-Programm schreiben (s. vorhergehende Seite)
9. SAVE "D:NAME.EXT"
10. NEW (RETURN)
11. Das eigentliche BASIC-Programm schreiben, das geschützt werden soll
12. Falls Sie mit Listschutz programmieren: GOTO 30000 (oder die Zeilen-Nummer, die Sie dafür bestimmen)
13. POKE 4226,105 oder Ausschalten des Rechners
14. Nach erneutem Einschalten: RUN "D:*.*" (RETURN)

Bitte denken Sie daran, daß das Register bis zum Ausschalten des Rechners den Dezimalwert enthält, den Sie eingePOKEt haben. Bevor Sie mit neuen Programmier-Arbeiten beginnen, sollten Sie den Normalwert 105 eingeben oder ausschalten. Sie finden sonst möglicherweise Ihre wertvollen Programme auf keiner Diskette mehr wieder, weil sie beim Auslisten der Directory nicht gefunden werden können.

Dezimalwert	- ASCII-Zeichen	- ATASCII-Zeichen

0	NUL		43	+		86	V	
1	SOH		44	,		87	W	
2	STX		45	-		88	X	
3	ETX		46	.		89	Y	
4	EOT		47	/		90	Z	
5	ENQ		48	0		91	[	
6	ACK		49	1		92	\	
7	BEL		50	2		93	]	
8	HT		51	3		94	^	
9	BS		52	4		95		
10	LF		53	5		96	`	
11	VT		54	6		97	a	
12	FF		55	7		98	b	
13	CR		56	8		99	c	
14	SO		57	9		100	d	
15	SI		58	:		101	e	
16	DLE		59	;		102	f	
17	DC1		60	<		103	g	
18	DC2		61	=		104	h	
19	DC3		62	>		105	i	
20	DC4		63	?		106	j	
21	NAK		64	@		107	k	
22	SYN		65	A		108	l	
23	ETB		66	B		109	m	
24	CAN		67	C		110	n	
25	EM		68	D		111	o	
26	SUB		69	E		112	p	
27	ESC		70	F		113	q	
28	FS		71	G		114	r	
29	GS		72	H		115	s	
30	RS		73	I		116	t	
31	US		74	J		117	u	
32			75	K		118	v	
33	!		76	L		119	w	
34	"		77	M		120	x	
35	#		78	N		121	y	
36	$		79	O		122	z	
37	%		80	P		123	{	
38	&		81	Q		124	|	
39	'		82	R		125	}	
40	(		83	S		126	~	
41	)		84	T		127	DEL	
42	*		85	U				

ZEICHENSATZ

ATASCII-Code - Zeichen - interner Code

A	Z	I	A	Z	I	A	Z	I
0	♥	64	40	(	8	80	P	48
1	┣	65	41	)	9	81	Q	49
2	┃	66	42	*	10	82	R	50
3	┛	67	43	+	11	83	S	51
4	┫	68	44	,	12	84	T	52
5	┓	69	45	-	13	85	U	53
6	╱	70	46	.	14	86	V	54
7	╱	71	47	/	15	87	W	55
8	◢	72	48	0	16	88	X	56
9	▒	73	49	1	17	89	Y	57
10	◣	74	50	2	18	90	Z	58
11	▪	75	51	3	19	91	[	59
12	▪	76	52	4	20	92	\	60
13	▔	77	53	5	21	93	]	61
14	▁	78	54	6	22	94	^	62
15	▪	79	55	7	23	95	_	63
16	♣	80	56	8	24	96	♦	96
17	┌	81	57	9	25	97	a	97
18	─	82	58	:	26	98	b	98
19	┼	83	59	;	27	99	c	99
20	●	84	60	<	28	100	d	100
21	▬	85	61	=	29	101	e	101
22	▏	86	62	>	30	102	f	102
23	┬	87	63	?	31	103	g	103
24	▟	88	64	@	32	104	h	104
25	▐	89	65	A	33	105	i	105
26	▙	90	66	B	34	106	j	106
27	▖	91	67	C	35	107	k	107
28	↑	92	68	D	36	108	l	108
29	↓	93	69	E	37	109	M	109
30	←	94	70	F	38	110	n	110
31	→	95	71	G	39	111	o	111
32		0	72	H	40	112	p	112
33	!	1	73	I	41	113	q	113
34	"	2	74	J	42	114	r	114
35	#	3	75	K	43	115	s	115
36	$	4	76	L	44	116	t	116
37	%	5	77	M	45	117	u	117
38	&	6	78	N	46	118	v	118
39	'	7	79	O	47	119	w	119

120	x	120	166		134	212		180
121	y	121	167		135	213		181
122	z	122	168		136	214		182
123	+	123	169		137	215		183
124	|	124	170		138	216		184
125	~	125	171		139	217		185
126	←	126	172		140	218		186
127	→	127	173		141	219		187
128		192	174		142	220		188
129		193	175		143	221		189
130		194	176		144	222		190
131		195	177		145	223		191
132		196	178		146	224		224
133		197	179		147	225		225
134		198	180		148	226		226
135		199	181		149	227		227
136		200	182		150	228		228
137		201	183		151	229		229
138		202	184		152	230		230
139		203	185		153	231		231
140		204	186		154	232		232
141		205	187		155	233		233
142		206	188		156	234		234
143		207	189		157	235		235
144		208	190		158	236		236
145		209	191		159	237		237
146		210	192		160	238		238
147		211	193		161	239		239
148		212	194		162	240		240
149		213	195		163	241		241
150		214	196		164	242		242
151		215	197		165	243		243
152		216	198		166	244		244
153		217	199		167	245		245
154		218	200		168	246		246
155		219	201		169	247		247
156		220	202		170	248		248
157		221	203		171	249		249
158		222	204		172	250		250
159		223	205		173	251		251
160		128	206		174	252		252
161		129	207		175	253		253
162		130	208		176	254		254
163		131	209		177	255		255
164		132	210		178			
165		133	211		179			

GRAPHICS 1 und 2

COLOR-Werte

Zeichen		Farbregister				Zeichen		Farbregister			
1.	2.	708	709	710	711	1.	2.	708	709	710	711
		32	0	160	128	@		64	96	192	224
!		33	1	161	129	A	a	65	97	193	225
"		34	2	162	130	B	b	66	98	194	226
#		35	3	163	131	C	c	67	99	195	227
$		36	4	164	132	D	d	68	100	196	228
%		37	5	165	133	E	e	69	101	197	229
&		38	6	166	134	F	f	70	102	198	230
'		39	7	167	135	G	g	71	103	199	231
(		40	8	168	136	H	h	72	104	200	232
)		41	9	169	137	I	i	73	105	201	233
*		42	10	170	138	J	j	74	106	202	234
+		43	11	171	139	K	k	75	107	203	235
,		44	12	172	140	L	l	76	108	204	236
-		45	13	173	141	M	m	77	109	205	237
.		46	14	174	142	N	n	78	110	206	238
/		47	15	175	143	O	o	79	111	207	239
0		48	16	176	144	P	p	80	112	208	240
1		49	17	177	145	Q	q	81	113	209	241
2		50	18	178	146	R	r	82	114	210	242
3		51	19	179	147	S	s	83	115	211	243
4		52	20	180	148	T	t	84	116	212	244
5		53	21	181	149	U	u	85	117	213	245
6		54	22	182	150	V	v	86	118	214	246
7		55	23	183	151	W	w	87	119	215	247
8		56	24	184	152	X	x	88	120	216	248
9		57	25	185	153	Y	y	89	121	217	249
:		58	26	186	154	Z	z	90	122	218	250
;		59	27	187	155	[		91	123	219	251
<		60	28	188	156	\		92	124	220	252
=		61	29	189	157	]		93	125	221	253
>		62	30	190	158	^		94	126	222	254
?		63	31	191	159	_		95	127	223	255

Die Werte n von 0 bis 255 rufen in den Text-Betriebsarten GRAPHICS 1 und 2 das Zeichen in der entsprechenden Zeile auf, das seinen Farbwert aus dem Register in der jeweiligen Spalte bezieht. Mit den Werten 125 und 155 können keine Zeichen aufgerufen werden. Um die Zeichen der zweiten Spalte zu erreichen, muß der Zeiger in Adresse 756 von 224 auf 226 (POKE 756,226) geändert werden.

GRAPHICS 1 und 2

COLOR-Werte (interner Code)

Zeichen		Farbregister				Zeichen		Farbregister			
1.	2.	708	709	710	711	1.	2.	708	709	710	711
	♥	0	64	128	192	@	♦	32	96	160	224
!	├	1	65	129	193	A	a	33	97	161	225
"	┤	2	66	130	194	B	b	34	98	162	226
#	┴	3	67	131	195	C	c	35	99	163	227
$	┬	4	68	132	196	D	d	36	100	164	228
%	┌	5	69	133	197	E	e	37	101	165	229
&	┐	6	70	134	198	F	f	38	102	166	230
'	└	7	71	135	199	G	g	39	103	167	231
(	◢	8	72	136	200	H	h	40	104	168	232
)	▘	9	73	137	201	I	i	41	105	169	233
*	◣	10	74	138	202	J	j	42	106	170	234
+	▝	11	75	139	203	K	k	43	107	171	235
,	▗	12	76	140	204	L	l	44	108	172	236
-	▀	13	77	141	205	M	m	45	109	173	237
.	▁	14	78	142	206	N	n	46	110	174	238
/	▖	15	79	143	207	O	o	47	111	175	239
0	♣	16	80	144	208	P	p	48	112	176	240
1	┌	17	81	145	209	Q	q	49	113	177	241
2	─	18	82	146	210	R	r	50	114	178	242
3	┼	19	83	147	211	S	s	51	115	179	243
4	●	20	84	148	212	T	t	52	116	180	244
5	■	21	85	149	213	U	u	53	117	181	245
6	▕	22	86	150	214	V	v	54	118	182	246
7	┬	23	87	151	215	W	w	55	119	183	247
8	┴	24	88	152	216	X	x	56	120	184	248
9	▐	25	89	153	217	Y	y	57	121	185	249
:	▙	26	90	154	218	Z	z	58	122	186	250
;	▟	27	91	155	219	[	♠	59	123	187	251
<	↑	28	92	156	220	\	▕	60	124	188	252
=	↓	29	93	157	221	]	▚	61	125	189	253
>	←	30	94	158	222	^	▌	62	126	190	254
?	→	31	95	159	223	_	▶	63	127	191	255

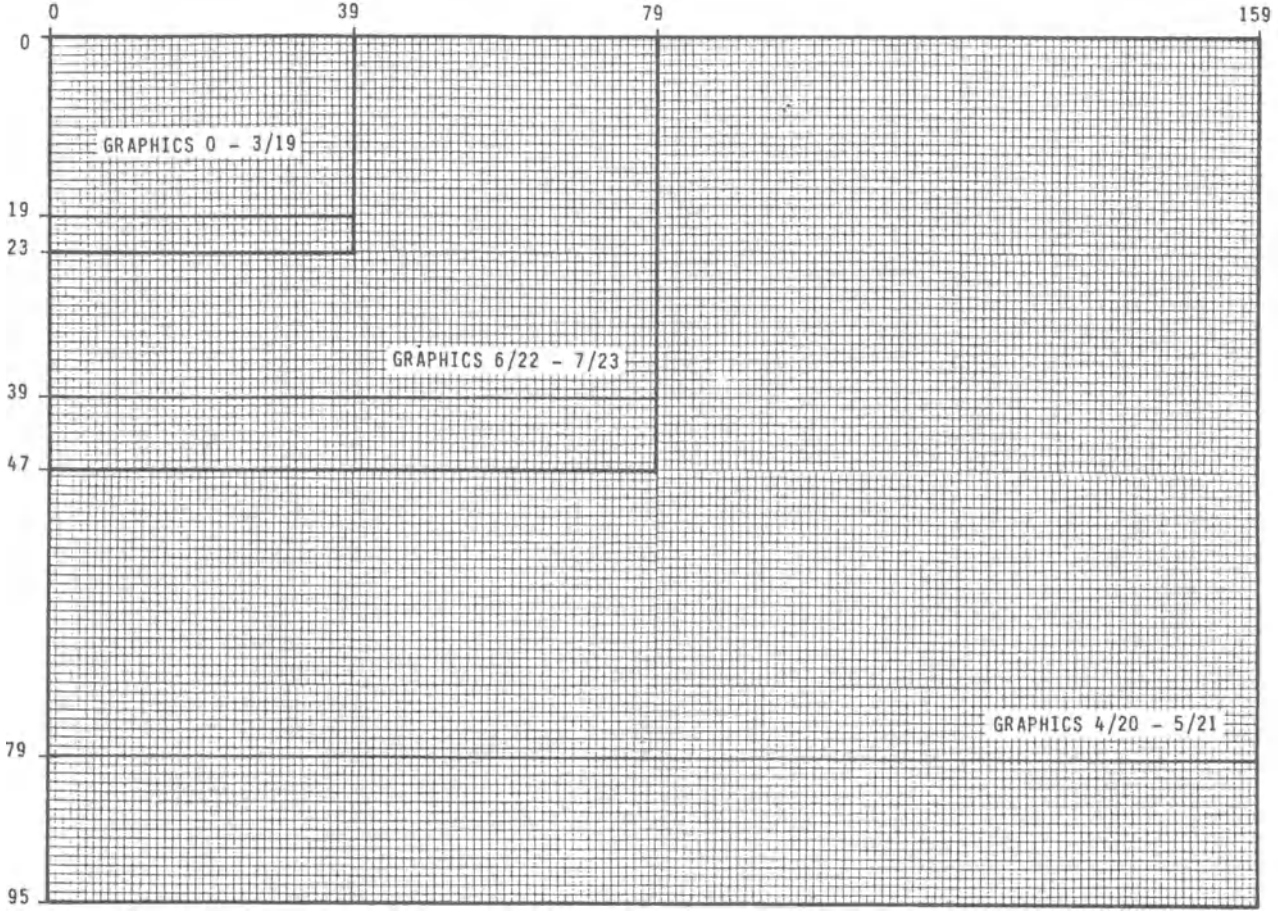
TASTATUR-CODE							
	gedrückte Taste(n)				gedrückte Taste(n)		
Code		+64	+128	Code		+64	+128
0	L	l		32	,	[	
1	J	j		33	leer		
2	;	:		34	.	]	
3				35	N	n	
4				36			
5	K	k		37	M	m	
6	+	\		38	/	?	
7	*	^		39	inv		
8	O	o		40	R	r	
9				41			
10	P	p		42	E	e	
11	U	u		43	Y	y	
12	Ret			44			
13	I	i		45	T	t	
14	-	_		46	W	w	
15	=	|		47	Q	q	
16	V	v		48	9	(	
17				49			
18	C	c		50	0	)	
19				51	7	'	
20				52	BackS		Del
21	B	b		53	8	@	
22	X	x		54	<	Clear	
23	Z	z		55	>	Ins	
24	4	$		56	F	f	
25				57	H	h	
26	3	#		58	D	d	
27	6	&		59			
28	ESC			60	Caps		
29	5	%		61	G	g	
30	2	"	BEL	62	S	s	
31	1	!		63	A	a	

In der Adresse 764 wird die zuletzt gedrückte Taste im Tastatur-Code erfaßt. Der Tastatur-Code ist eine 6-Bit-Information (Dezimalwerte von 0 bis 63). Wird zusammen mit einer Taste SHIFT gedrückt, wird Bit 6 (dezimal 64) gesetzt. Das Drücken von CONTROL setzt Bit 7 (dezimal 128).

TASTATUR-Code

♥	160	SPACE	33	@	117	♣	162
├	191	!	95	A	63	a	127
┤	149	"	94	B	21	b	85
┴	146	#	90	C	18	c	82
┬	186	$	88	D	58	d	122
┌	170	%	93	E	42	e	106
┐	184	&	91	F	56	f	120
└	189	'	115	G	61	g	125
┘	185	(	112	H	57	h	121
▝	141	)	114	I	13	i	77
▘	129	*	7	J	1	j	65
▗	133	+	6	K	5	k	69
▖	128	,	32	L	0	l	64
▀	165	-	14	M	37	m	101
▄	163	.	34	N	35	n	99
▌	136	/	38	O	8	o	72
▐	138	0	50	P	10	p	74
▛	175	1	31	Q	47	q	111
▀	168	2	30	R	40	r	104
▜	190	3	26	S	62	s	126
▙	173	4	24	T	45	t	109
▟	139	5	29	U	11	u	75
▄	144	6	27	V	16	v	80
▛	174	7	51	W	46	w	110
▙	150	8	53	X	22	x	86
▟	171	9	48	Y	43	y	107
▜	151	:	66	Z	23	z	87
▞	28	;	2	[	96	♠	130
↑	142	<	54	\	70	¦	79
↓	143	=	15	]	98	CLEAR	118
←	134	>	55	^	71	BACK	52
→	135	?	102	_	78	TAB	44

39	ATARI	119	INSERT
60	CAPS	118	CLEAR
167	CONTROL+ATARI	172	CONTROL+TAB
12	RETURN	108	SHIFT+TAB
116	DELETE	158	CONTROL+2 (BEL)


0
39
79
159
0
19
23
39
47
79
95
GRAPHICS 0 - 3/19
GRAPHICS 6/22 - 7/23
GRAPHICS 4/20 - 5/21