Hornig · Trapp
Weltner

noch mehr 64 Tips & Tricks

BAND 2

Eine Fundgrube für den
Commodore 64 Anwender

EIN DATA BECKER BUCH

Hornig · Trapp
Weltner

noch mehr 64 Tips & Tricks

BAND 2

Eine Fundgrube für den Commodore 64 Anwender

EIN DATA BECKER BUCH

ISBN 3-89011-065-7

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von dem Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

INHALT

======

1. EINLEITUNG

Es dürfte jedem Benutzer inzwischen klar sein, daß der C-64 ein Super-Computer ist. Aber auch die Leute, die diesen Computer gebaut haben, konnten nicht an alles denken. Wir wollen Ihnen hier deshalb einige Fehler zeigen, mit denen Sie Leute, die diese Fehler nicht kennen, sehr schön erschrecken können, indem Sie die Fehler auftreten lassen und sagen, der Computer sei kaputt. Auch in einem Fachgeschäft könnte das eine gewisse Wirkung zeigen.

Doch nun zu den Fehlern:

1. Fehler:
Diesen Fehler werden Sie vielleicht auch schon bemerkt haben. Wenn Sie gleichzeitig die linke Shift-Taste und die beiden Cursor-Tasten drücken, dann erscheint ein Pik-Zeichen auf dem Bildschirm.

2. Fehler:
Drücken Sie einmal gleichzeitig die Commodore-Taste, die Taste mit dem Semikolon und die Gleich-Taste. Der Cursor wird verschwinden.
Was ist passiert?
Nun drücken Sie Run-Stop/Restore und geben Sie den Befehl

POKE 53281,0

ein.
Wenn Sie jetzt noch einmal diese drei ersten Tasten gleichzeitig drücken, so sehen Sie, warum der Cursor verschwunden ist:
Die Zeichenfarbe hat von weiß zu blau gewechselt.

3. Fehler:
Hier handelt es sich eindeutig um den interessantesten Fehler.
Gehen Sie mit Ihrem Cursor auf die unterste Zeile vom Bildschirm und schreiben Sie jetzt so lange (auch Leerzeichen), bis der Cursor zum 2.Mal rechts aus dem Bildschirm gesprungen ist. Drücken Sie jetzt die INST-DEL-Taste. Sobald Sie versuchen, das 80.Zeichen zu löschen, tritt der Fehler auf:
Auf dem Bildschirm erscheint folgendes Bild:

```
LOAD

?SYNTAX ERROR
READY
RUN

READY.
```

Nun geht nichts mehr - der Computer ist im Niemandsland.
Aber das war noch nicht alles: Ein BASIC-Programm wird gestartet, bis es an eine INPUT-Anweisung kommt. Dort hängt es sich auf, da die Tastatur nicht mehr funktioniert. Sollte kein Programm vorhanden sein, so erscheint das oben beschriebene Bild. Wer jetzt aber denkt, daß es außer Reset- und Aus-Einschalten des Computers keine Möglichkeit mehr gibt, den Computer aus dem Niemandsland wieder auf sicheren Boden zurückzuführen, der hat sich geirrt. Geben Sie # (SHIFT + 3) ein. Daraufhin gibt der totgeglaubte Computer "PRESS PLAY ON TAPE" aus. Sollten Sie eine Datasette angeschlossen haben, so kommen Sie dieser Aufforderung nach. Nun geht der C-64 wieder in den normalen Lade-Modus, den Sie ganz einfach mit Run-Stop unterbrechen können.

Sonderbarerweise tritt dieser Fehler tritt nur bei einigen

Zeichenfarben auf: rot, cyan, dunkelblau, gelb (mit Variante des Fehlers), rosa, dunkelgrau, hellblau und hellgrau.

Es funktioniert demnach nur bei Farben, die auf den Tasten 3,4,7 und 8 liegen.
Übrigens: Wenn dieser Fehler bei Ihnen nicht auftritt, so seien Sie beruhigt - Ihr Computer ist nicht defekt. Sie haben nur einen Computer der neuen Serie. Bei diesen Modellen tritt dieser Fehler nicht mehr auf.
Ob Sie einen Computer neueren Datums haben, können Sie auch auf andere Art erfahren: Löschen Sie den Bildschirm und geben Sie POKE 1024,1 ein. Sollte in der linken oberen Ecke ein A erscheinen, so sind Sie glücklicher Besitzer eines neuen Modells, da bei einem älteren zusätzlich ein Wert in den Farb-RAM gePOKEt werden muß, um ein Zeichen erscheinen zu lassen.

Fragen Sie uns jedoch nicht, woher die Fehler kommen, wir wissen es nicht. Die ersten beiden scheinen mit der Tastatur-Abfrage zusammenzuhängen. Beim letzten erwies sich der Verdacht, die IRQ-Routine sei verbogen worden, als unbegründet (TI$ wird weitergesetzt, der Cursor blinkt etc.).

Das sollte nur ein kleiner Einstieg sein. Wie Sie (trotz dieser kleinen Fehler) das Beste aus Ihrem C-64 machen können, sollen Ihnen die nächsten Seiten zeigen.

2. *TIPS & TRICKS FÜR DEN HAUSGEBRAUCH*

Auf den folgenden Seiten finden Sie eine ganze Reihe nützlicher BASIC- und Maschinen-Routinen.
Es werden interessante Speicheradressen und damit verbundene Möglichkeiten aufgezeigt...
Kurz: Tips & Tricks für den Hausgebrauch !

STEUERN DER DATASETTE VON BASIC AUS

Es ist eine angenehme Eigenschaft, daß der Motor der Datasette nach Beendigung eines LOAD-, bzw. SAVE-Vorganges selbsttätig stoppt.
Erst nach Betätigung der Stoptaste am Rekorder kann der Motor wieder durch Drücken der PLAY-Taste gestartet werden.
Der Motor der Datasette kann also folglich vom Computer aus gesteuert werden.
Diese Eigenschaft läßt sich selbstverständlich auch für eingene Zwecke ausnutzen !
Wichtig bei der Software-Kassettenmotorsteuerung ist die Adresse 1, der Prozessor-Port.
Außerdem wird das Kassettenmotor-Flag (Adresse 192, $C0) benötigt.
Zur Demonstration der Steuereigenschaft drücken Sie bitte die PLAY-Taste des Rekorders : Das Band wird gespult, der Motor läuft.
Geben Sie nun folgende Zeile im Direktmode ein:

```
POKE 192,1: POKE 1,PEEK(1) OR 32
```

Der Motor der Datasette stoppt, ohne daß die STOP-Taste des Rekorders betätigt wurde. Das 5. Bit der Adresse 1 und das Kassettenmotor-Flag wurden gesetzt.
Um den Motor nun wieder softwaregesteuert zu starten, genügt die folgende Zeile:

```
POKE 1,PEEK(1) AND 39: POKE 192,0
```

Der Rekorder arbeitet nun wieder wie vor Ihrem Eingriff !
Mit der Steuerung des Motor sind die Möglichkeiten jedoch noch nicht erschöpft. Es lassen sich im Gegenteil noch Aussagen über die Betätigung der Tasten am Rekorder machen. Wieder wird Adresse 1 benutzt:

```
IF PEEK(1) =55 THEN PRINT" KEINE TASTE BETAETIGT !"
IF PEEK(1) =7  THEN PRINT" TASTE GEDRUECKT !!"
```

Dies läßt sich auch mit einem WAIT-Befehl nutzen:

```
WAIT 1,16
```

wartet solange, bis die STOP-Taste des Rekorders betätigt wird.

Hier noch einmal alle Steuereigenschaften zusammengefaßt:

```
Rekordermotor aus:.....POKE 192,1: POKE 1,PEEK(1) OR 32
Rekordermotor ein:.....POKE 1,PEEK(1) AND 39:POKE 192,0
Warten auf STOP-Taste:.....WAIT 1,16
Warten auf PLAY-Taste:.....x IF PEEK(1)=55 THEN GOTO x
```

EINE KOPIER-SCHUTZ-VARIANTE FÜR DIE DATASETTE

Im nun Folgenden soll gezeigt werden, wie auf einfache Art verhindert werden kann, daß ein unerlaubt kopierte Programm problemlos gestartet werden kann.
Man macht sich dabei folgende Eigenschaft zunutze: Wenn Sie ein Programm auf Datasette abSAVEn, haben Sie die Möglichkeit, einen bis zu 172 Zeichen langen Programmnamen anzugeben. Wird das Programm jedoch in den Computer geladen, so werden lediglich bis zu 16 Zeichen ausgegeben. Die

Übrigen 160 Zeichen des Programmnamens bleiben im Kassettenpuffer verborgen.
Das folgende kleine BASIC-Programm fragt nun das 17.Zeichen des Programmnamens ab. Es wird beim Laden nicht auf den Bildschirm ausgegeben und ist somit unbekannt.
Wird nun das Programm widerrechtlich kopiert, so wird das 17.Zeichen des Programmnamens nicht mit übernommen, da es dem Kopierer ja unbekannt ist.

```
10 IF PEEK(849) =ASC("X") THEN GOTO 30
20 PRINT"? LOAD  ERROR":NEW
30 PRINT"ORIGINALPROGRAMM !"
```

Diese drei Zeilen sollten Sie vor Ihr eigenes Programm stellen. Wenn Sie das Programm abSAVEn, so geben Sie als 17.Zeichen im Programmnamen den Buchstaben "X" ein !

SAVE "1234567890123456X" (X genau an 17. Stelle!)

Wird das Programm jetzt kopiert, erkennt dies der Computer am fehlenden 17.Zeichen des Programmnamens, gibt ein ? LOAD ERROR aus und löscht das Programm. So ist der Kopierschutz als solcher gar nicht zu erkennen!
Wollen Sie einen anderen Buchstaben als das X verwenden, so muß lediglich Zeile 10 entsprechend abgeändert werden.
Es ist darauf zu achten, daß das mit dem Kopierschutz versehene Programm lediglich mit LOAD ohne Namensbezeichnung geladen wird.
Perfekter wird der Kopierschutz, wenn die drei Kontrollzeilen zusätzlich mit einem Listschutz versehen werden (siehe auch Kapitel Softwareschutz !).

VERSCHIEBEN DES KASSETTENPUFFERS

Der Kassettenpuffer liegt normalerweise von $033C - $03FB (828 - 1019) im Speicher.

In ihm werden die Daten, die vom Rekorder kommen oder zu ihm geleitet werden sollen, zwischengespeichert und blockweise ausgegeben.
Der Kassettenpuffer wird somit nur bei LOAD-, bzw. SAVE-Anweisungen in Verbindung mit der Datasette benutzt, sonst ist dieser Speicherbereich ungenutzt.
Er wird daher sehr gern zur Speicherung kleinerer Maschinenprogramme oder als Platz für die Sprite-Blocks 13-15 genutzt. Dies geht solange gut, bis - ja, bis der Kassettenpuffer wieder einmal seiner eigentlichen Bedeutung zugeführt wird, und Programme von Datasette geladen oder auf Kassette geSAVEt werden.
In diesem Moment nämlich werden alle zuvor in diesem Speicherbereich stehenden Daten überschrieben.
Es gibt jedoch eine simple, wenn auch wenig bekannte Methode, dies zu verhindern. Man bedient sich des Kassettenpuffer-Vektors ($B2/$B3, 178/179), der normalerweise auf den Beginn des Kassettenpuffers zeigt.
Möchte man also den Inhalt des ursprünglichen Kassetten-Puffers vor Überschreiben schützen, so legt man den Vektor beispielsweise in dem ungenutzten Speicherbereich ab $C000:

```
10 A=49152: REM ANFANGSADRESSE NEUER KASSETTENPUFFER
20 HI=INT(A-256)
30 LO=A-(HI*256)
40 POKE 178,LO: POKE 179,HI
```

Sollte der C-Bereich bereits belegt sein, bietet sich auch der Bildschirmspeicher als Kassettenpuffer an. Er wird während des LADE-, bzw. SAVE-Vorganges der Datasette ohnehin nicht benutzt.
Zeile 10 muß in diesem Fall ersetzt werden durch:

```
10 A=1024
```

LADEN NUR MIT CODE

Nach der Datasette kommen wir nun zur Floppy 1541. Auch hier wird der Programm-Name bei LOAD-, bzw. SAVE-Befehlen im Speicher des C64 abgelegt. Der Kassettenpuffer bleibt jedoch von diesem Vorgang unberührt.
Vielmehr gibt der Zeiger $BB/$BC (187/188) Aufschluß über die Adresse, ab der der Name gespeichert worden ist.
Das folgende kleine BASIC-Programm gibt den zuletzt bei Disketten-Operationen benutzten Programmnamen auf dem Bildschirm aus:

```
10 AD= PEEK(187)+256*PEEK(188)
20 FOR A=AD TO 40959: B=PEEK(A)
30 NA$=CHR$(B)
40 NEXT:PRINT NA$
```

Man kann sich auch die Floppy-Programmnamen-Abspeicherung zunutze machen!
Stellen Sie einmal die folgenden Zeilen irgendeinem Ihrer Programme voran!

```
10 FOR A=40955 TO 40959
11 B=PEEK(A)
12 READ C$: C=ASC(C$)
13 IFB =C THEN 15
14 PRINT"? LOAD  ERROR";:NEW
15 NEXT
16 DATA C,O,D,E,1
```

SAVEn Sie nun das so veränderte Programm unter einem unbedingt 16 Zeichen langen Namen ab. Nun laden Sie das Programm wie gewöhnlich. Starten Sie es! Es kommt zu einem LOAD ERROR, und das Programm ist gelöscht.
Laden Sie das Programm erneut. Geben Sie diesmal den

kompletten Programmnamen ein und hängen Sie dahinter das Kennwort "CODE1". Wenn das auf diese Weise geladene Programm gestartet wird, gibt es keine Komplikationen.
Wieso? Nun, in Zeile 10 werden die letzten fünf Zeichen des gespeicherten Programmnamens abgefragt und mit dem Kennwort in der DATA-Zeile verglichen. Nur wenn sie identisch sind, kann das Programm fortgesetzt werden.
Wichtig ist auch, daß der reguläre Programmname 16 Zeichen lang ist. Sonst ist es nicht möglich, weitere Code-Zeichen einzugeben.

FEHLERKANAL AUSLESEN

Oft kommt es bei der Benutzung der Floppy-Disk 1541 zu einem Fehler, gut zu erkennen an der blinkenden, roten LED.
Manchmal jedoch ist man sich keiner Schuld bewußt und sucht oft lange nach dem vermeindlichen Fehler.
Einfacher und schneller geht es mit der folgenden kleinen Routine, die den Fehlerkanal der Floppy ausliest. Sie gestattet die bequeme Ausgabe der Fehlermeldung und erleichtert so das Aufspüren des Fehlers.
Das Maschinen-Äquivalent dazu finden Sie unter dem Kapitel Floppy-Tricks.

```
10 REM AUSLESEN DES FEHLERKANALS DER FLOPPY
20 OPEN 1,8,15: REM OEFFNEN DES KANALS
30 INPUT#1,F ,FM$,SP,SE
40 PRINT F ","FM$","SP","SE
50 CLOSE 1
```

Variablen:

F	=	Fehlernummer
FM$	=	Fehlermeldung
SP	=	Spur
SE	=	Sektor

AUS SAVE MACH LOAD !

Die folgenden Vektoren und deren Manipulation sind sowohl für die Datasette als auch für das Floppy-Disk 1541 interessant.
Beschäftigen wollen wir uns mit dem SAVE- ($0332/$0333 818/819) und dem LOAD-Vektor ($033O/$0331 816/817).
Diese beiden Zeiger deuten auf die im ROM liegenden LOAD- bzw. SAVE-Routinen.
Gemäß der Überschrift soll nun aus SAVE LOAD werden. Dies ist nicht allzu schwer (wenngleich auch kein funktioneller Austausch praktiziert wird, scheint dies doch wenigstens so !):

```
POKE 818,PEEK(816): POKE 819,PEEK(817)
```

Ab sofort ist SAVE nicht mehr SAVE, sondern LOAD (VERIFY). Nun kann man nicht mehr ohne weiteres SAVEn.
Den Normalzustand erreicht man wieder durch:

```
POKE 818,237: POKE 819,245      (Zeiger $F5ED)
```

Auch umgekehrt läßt sich LOAD ausschalten:

```
POKE 816,PEEK(818): POKE 817,PEEK(819)
```

Zurück in den Normalzustand gelangt man wieder mit:

```
POKE 816,165: POKE 817,244      (Zeiger $F4A5)
```

LOAD und SAVE vertauschen - auch das ist nicht schwer:

```
POKE 816,237:POKE817,245:POKE818,165:POKE819,244
```

Das ausschalten von LOAD und SAVE erreicht man durch:

```
POKE 818(816),26: POKE819(817),167
```

Sowohl LOAD als auch SAVE werden einfach mit einem kühlen "READY" beantwortet.

AUTOMATISCHES NACHLADEN

Nehmen wir einmal folgenden Fall an: Sie haben ein Programm geschrieben, das während des Programmablaufes weitere Teile nachlädt oder Dateien benutzt.
Dies ist kein Problem, wenn man weiß, ob mit Floppy oder Datasette gearbeitet wird.
Nun kann man selbstverständlich den Benutzer vom Programm aus fragen, welches Speichergerät er benutzt. Dies ist jedoch nicht nur umständlich, es ist auch für den Benutzer unbequem. Eleganter geht es auf folgende Art (Beispiele):

```
450 FD=PEEK(186): OPEN 1,FD,1      oder

670 FD=PEEK(186): LOAD"TEST",FD,(Sekundäradresse)
```

Selbstverständlich ist der Computer nicht allwissend. So kann er natürlich auch nicht voraussagen, welches Gerät Sie zur Speicherung verwenden. In den oben genannten Beispielen wird einfach davon ausgegangen, daß das Gerät, mit dem das Hauptprogramm geladen wurde, auch zur weiteren Speicherung benutzt wird.

Adresse 186 enthält die Nummer des zuletzt benutzten Gerätes. Sie müssen nur auf weitere Geräte, beispielsweise den Drucker aufpassen.
Es empfiehlt sich, am Programmanfang der Variable FD den Inhalt der Adresse 186 zuzuweisen. So kann dann später

dieser Wert nicht mehr durch andere Geräte verfälscht werden!

Ein Wörtchen zu LOAD ERROR...

Es kann schon einmal vorkommen, daß ein von Kassette geladenes Programm einen LOAD ERROR hervorruft.
Dieser Ladefehler muß allerdings noch nicht bedeuten, daß das geladene Programm für den Benutzer verloren ist.
Sowohl der VC-20 als auch der Commodore 64 speichern das Programm bei einem SAVE-Vorgang zweimal hintereinander auf das Band.
Beim Ladevorgang wird die erste Version in den Speicher geholt und mit der auf Band befindlichen zweiten Version verglichen. Wenn sich diese beiden Versionen voneinander unterscheiden, kommt es zum LOAD ERROR.
Möchte man das geladene Programm nun retten, muß zunächst überprüft werden, ob es sich auflisten läßt. Wenn dies der Fall ist, kann das Programm gerettet werden.
Im Kassettenpuffer ist in den Adressen 831/832 die Länge des geladenen Programmes abgespeichert. Diese Information wird jedoch erst vom Computer übernommen, wenn das Programm fehlerfrei geladen worden ist.
Tritt nun ein LOAD ERROR auf, fehlen dem Computer diese Informationen. Der Variablenstart liegt dann im BASIC-Programm und zerstört dieses.
Folgende Zeilen im Direktmode eingegeben, können das fehlerhaft geladene Programm retten (sofern es sich listen ließ):

```
POKE 46,PEEK(832): POKE 47,PEEK(831): POKE 48,PEEK(832):
POKE 49,PEEK(831): POKE 50,PEEK(832)
```

Anschließend kann das Programm mit RUN gestartet werden!

LOAD UND SAVE BEI MASCHINEN-PROGRAMMEN

Um ein Maschinenprogramm auf Band zu speichern, gibt es eine Möglichkeit, die komfortabel ist, leider aber nicht oft genug benutzt wird:
Das Maschinenprogramm direkt auf Band speichern!
Im Gegensatz zum Basic-Loader hat sie den Vorteil, daß es schneller geht, und daß der Speicher, den der Basic-Loader einnehmen würde, eingespart wird.
Auch selbstdefinierte Zeichensätze können so einfach abgespeichert werden.
Zuerst müssen Sie wissen, wo das Programm beginnt, und wo es endet. Nehmen wir an, Sie wollen den Bereich $5000-$6000 abSAVEn.
Beim SAVEn wird alles zwischen dem BASIC-Anfang (43/44) und dem Variablen-Anfang (45/46) auf Band gespeichert.
Sie müssen also $5000 in den Vektor 43/44 und $6000 in den Vektor 45/46 schreiben.

```
$5000= #20480     LB= 0     HB= 80
$6000= #24576     LB= 0     HB= 96
```

Der Bereich $5000- $6000 wird auf folgende Art abgespeichert:

```
POKE 43,0: POKE 44,80: POKE 45,0: POKE 46,96: CLR:
SAVE "(name)",1,1
```

Die erste EINS hinter dem SAVE ist die Geräte-Adresse (ACHT für Floppy).
Die zweite EINS bedeutet, daß der Bereich unmittelbar abgespeichert wird, das heißt, bei erneutem Laden wird der Bereich automatisch wieder an die gleiche Adresse geladen.
Außerdem sieht der Computer diesen Bereich nicht als

BASIC-Programm an und berechnet demnach auch keine Links.

Noch ein Trick:
Nehmen wir an, Sie haben ein Programm im Bereich $5000-$6000 abgeSAVEt, wollen es jetzt aber in den Bereich $8000-$9000 laden.
Das geht ohne große Probleme.
Geben Sie SYS 63276 ($F72C) ein.
Es erscheint die Meldung PRESS PLAY ON TAPE. Dieser Aufforderung kommen Sie nach. Es wird jedoch nur der Tape-Header geladen.

Da im Tape-Header angegeben wird, von wo bis wo das Programm steht, können Sie es einfach verschieben:

```
POKE 829,LB (Anfang)
POKE 830,HB (Anfang)
POKE 831,LB (Ende)+2
POKE 832,HB (Ende)
```

Für $8000-$9000 müssen Sie eingeben:

```
POKE 829,0: POKE 830,128: POKE 831,0: POKE 832,144
SYS 62849 ($F581)
```

Der letzte Befehl setzt den Ladevorgang fort.
Nach Beendigung geben Sie NEW ein und können nun starten.
Dieser Trick geht allerdings nur mit Datasette, da das Floppy den Kassetten-Puffer nicht benutzt. Somit können auch die einzelnen Adressen nicht auf die hier beschriebene Weise geändert werden.

UMWANDLUNG ASCII-BILDSCHIRMCODE

Sicherlich werden Ihnen die beiden Tabellen im Anhang des C-64 Handbuches schon einmal aufgefallen sein: Ich meine die Liste mit den Bildschirmcodes und die mit den ASCII-Werten. Während der Bildschirmcode dazu benutzt wird, Zeichen durch den POKE-Befehl in den Bildschimspeicher ($0400, 1024) zu bringen, läßt sich der ASCII-Code in einen String verwandeln, der mit PRINT ausgegeben werden kann (ASC-CHR$). In beiden Tabellen befinden sich dieselben Zeichen, lediglich die Code-Werte unterscheiden sich voneinander.
In manchen Programmen müssen jedoch ASCII-Codes in Bildschirm-Codes verwandelt werden (siehe beispielsweise Listing Zeichen-Editor). Dies läßt sich durch folgende kleine BASIC-Routinen erreichen:

```
10 REM ASCII in BS-CODE
20 T$="A": T=ASC(T$): PRINT"ASCII-CODE =";T
30 PRINT "(home)";T$
40 BS=256*PEEK(648): REM BS=ANFANG BILDSCHIRMSPEICHER
50 CO=PEEK(BS): REM CO=BILDSCHIRMCODE
60 PRINT"BILDSCHIRM-CODE =";CO
70 END
```

Die Stringvariable T$ enthält das betreffende Zeichen, hier der Buchstabe "A". Der ASCII-Wert wird mit ASC ermittelt und ausgegeben. Um den Bildschirmcode zu ermitteln, wird zunächst die Anfangsadresse des Bildschirmspeichers errechnet (Adresse 648 = High-Byte des Video-RAM).
Der Buchstabe in T$ wird in die erste Speicherstelle des Bildschirmspeichers gePRINTet.
Durch PEEK kann jetzt diese Speicherstelle ausgelesen werden, und man erhält so den Bildschirmcode.
Der Nachteil dieser Routine ist der etwas unschöne Eingriff in den Bildschirmspeicher. Das betreffende Zeichen wird

dabei links oben auf dem Bildschirm ausgegeben.
Deshalb finden Sie nun zwei kleine Hilfsroutinen, die die Umrechnung auf rein rechnerischem Weg vornehmen:

```
10 REM VIDEO- IN ASCII-CODE
20 INPUT"WIE LAUTET DER BILDSCHIRMCODE";CO
30 CO=CO AND 127
40 IF CO AND 32 THEN 70
50 IF CO AND 64 THEN AC=CO OR 32: GOTO 90
60 AC=CO OR 64: GOTO 90
70 IF CO AND 64 THEN AC=CO AND 63 OR 12: GOTO 90
80 AC=CO
90 PRINT "ASCII-CODE =";AC
100 END
```

```
10 REM ASCII- IN VIDEO-CODE
20 INPUT " WIE LAUTET DER ASCII-CODE";AC
30 IF AC AND 128 THEN CO=AC AND 127 OR 64: GOTO 70
40 IF NOT AC AND 64 THEN CO=AC: GOTO 70
50 IF AC AND 32 THEN CO=AC AND 95: GOTO 70
60 CO=AC AND 63
70 PRINT "VIDEO-CODE =";CO
```

Variablen:

```
AC     =    ASCII-Wert
CO     =    Video-Code
```

Bequemes Zeilenlöschen

Oft müssen in Programmen Teile des Bildschirms gelöscht werden. Dies läßt sich natürlich durch eine PRINT-Anweisung und die Cursor-Steuertasten machen. Die folgende Möglichkeit ist jedoch einfacher und vor allem übersichtlicher. Die

aktuelle Cursorposition wird nämlich gar nicht angetastet. Steuerzeichen fallen also weg. Sie können eine bereits im ROM abgespeicherte Routine benutzen: $E9FF (59903) Löschen einer Bildschirmzeile.
Der Aufruf dieser Routine von BASIC sieht so aus:

```
POKE 781,x: SYS 59903
```

x = 0-24

Dem X-Register (781) wird die gewünschte Zeile, die zwischen 0 und 24 liegen kann, zugewiesen. Den Rest erledigt die Routine.

Durch eine kleine Manipulation läßt sich die ROM-Routine sogar noch etwas vielseitiger einsetzen: Die ersten zwei Bytes werden einfach übersprungen (neue Sprungadresse $EA01=59905). Jetzt kann dem Z-Register (Adresse 782) die Anzahl der Zeichen zugeordnet werden, die in der in X angegebenen Bildschirmzeile gelöscht werden sollen.

```
POKE 781,x: POKE 782,z: SYS 59905
```

x=0-24, z=0-39 (0-255)

HEX-EINGABE

Zur Eingabe von Maschinen-Programmen in den Speicher gibt es mehrere Möglichkeiten:

a) Monitor
b) Basic-Loader
c) Hexadezimale Zeichen

Wir wollen ein Programm zur letzten Möglichkeit vorstellen:
Vielleicht ist Ihnen auch schon folgendes passiert:
Sie schlagen eine Computer-Zeitschrift auf, finden ein Programm, das Sie gern haben wollen, doch statt eines Basic-Loaders finden Sie nur einige Seiten mit hexadezimalen Zahlen vor.
Was nun?
Mit Hilfe dieses Programmes sind Ihre Probleme gelöst.
Tippen Sie es ab und starten Sie es.
Zuerst fragt der Computer nach der Schrittweite. Gemeint ist damit die Anzahl der hexadezimalen Zahlen in einer Zeile. Sie darf 11 nicht überschreiten. Sollte es vorkommen, daß die Vorlage längere Zeilen benutzt, müßten Sie eventuell das Programm entsprechend abändern. Es wurde daher extra übersichtlich geschrieben.
Der Computer fragt nach der Anfangs-Adresse. Diese muß hexadezimal eingegeben werden und 4 Zeichen lang sein.
Der Computer schreibt die jeweilige Speicherstelle des ersten Elements jeder Zeile hin, und Sie müssen nun die Zahlen eingeben.
Abgebrochen werden kann das Programm durch Eingabe von RETURN.

```
0 PRINT"■■          HEX-EINGABE-PROGRAMM"
10 INPUT"■■■SCHRITTWEITE:";SW:IFSW<1ORSW>11THEN10
20 INPUT"ANFANGSADRESSE:";A$
30 IFLEN(A$)<>4THENPRINT"FALSCHE EINGABE":GOTO20
40 POKE19,0:PRINT:Y=3:FORZ=1TO4:B$=MID$(A$,Z,1):B=ASC(B$)
50 IFB>47ANDB<58THENSA=SA+VAL(B$)*16↑Y:GOTO80
60 IFB>64ANDB<71THENSA=SA+(ASC(B$)-55)*16↑Y:GOTO80
70 PRINT"FALSCHE EINGABE":GOTO20
80 Y=Y-1:NEXTZ
100 PRINT"■"SA;:FORZ1=0TOSW-1:FORZ2=1TO0STEP-1
110 POKE204,0:GETA$:IFA$=""THEN110
120 POKE207,0:B=ASC(A$):IFB>47ANDB<58THENPE=PE+VAL(A$)*16↑Z2:GOTO160
130 IFB>64ANDB<71THENPE=PE+(ASC(A$)-55)*16↑Z2:GOTO160
140 IFB=13THENEND
150 GOTO110
160 PRINTA$;:NEXTZ2:PRINT" ";:POKESA+Z1,PE:PE=0:NEXTZ1
170 SA=SA+SW:PRINT:GOTO100

READY.
```

DATA-GENERATOR

Dieses Programm hilft Ihnen, ein im Speicher befindliches Maschinen-Prgramm in Data-Zeilen zu übersetzen.
Eingeben müssen Sie den Anfang des Maschinen-Programms, die Länge, und die Zeilnennummer, ab der die Datas gebildet werden sollen. Diese Zeilennummer sollte nicht unter 221 liegen, da sonst das Programm sich selbst überschreibt.
Die Data-Zeilen werden in Einer-Schritten geschrieben.
Da das Programm sich nach dem Erstellen jeder Data-Zeile selbst unterbricht, mußten die Variablen in speziellen Speicherstellen untergebracht werden.

828-829 = Anfang des Maschinen-Programms
830-831 = Länge des Maschinen-Programms
832-833 = Momentane Zeilennummer

Wenn Sie CLR-Home + RETURN in den Tastatur-Puffer schreiben, dann wird die Zeile ins Programm übernommen. Da außerdem die Befehle RUN 100 und noch einmal RETURN eingegeben wurde, startet das Programm wieder bei Zeile 100.
In der Zeile 15 sind die Werte für den Tastatur-Puffer untergebracht.
Zeile 20 berechnet LOW- und HIGH-Byte.
Möglich gewesen wäre auch, die Variablen selbst in Data-Zeilen unterzubringen und sie dann bei jedem Lauf neu zu lesen. Wenn Sie wollen, können Sie das Programm ja als kleine Übung in diesem Sinne umschreiben.

```
10 GOTO25
15 DATA19,13,82,213,49,48,48,13
20 HB=INT(X1/256):LB=X1-256*HB:RETURN
25 PRINT"■■■■■■■■■■■■ DATA - GENERATOR  ■"
26 PRINT"■■GEBEN SIE FOLGENDE PARAMETER EIN:■■■"
30 INPUT"■PROGRAMM-ANFANG";PA
40 INPUT"■PROGRAMM-LAENGE";PL
50 INPUT"■ANFANGS-ZEILE";AZ
60 X1=PA-1:GOSUB20:POKE828,LB:POKE829,HB
70 X1=PL:GOSUB20:POKE830,LB:POKE831,HB
80 X1=AZ:GOSUB20:POKE832,LB:POKE833,HB
100 X=PEEK(832)+256*PEEK(833)
110 Y=PEEK(828)+256*PEEK(829)
120 Z=PEEK(830)+256*PEEK(831)
130 PRINT"■"X" DATA";
140 FORZ1=1TO10:PRINTPEEK(Y+Z1)"■,";
150 Z=Z-1:IFZ=0THENPRINT"■ ":GOTO220
160 NEXTZ1:PRINT"■ "
170 X=X+1:Y=Y+10
180 X1=Y:GOSUB20:POKE828,LB:POKE829,HB
190 X1=Z:GOSUB20:POKE830,LB:POKE831,HB
200 X1=X:GOSUB20:POKE832,LB:POKE833,HB
210 FORZ1=631TO638:READA:POKEZ1,A:NEXTZ1:POKE198,8:END
220 POKE631,19:POKE632,13:POKE198,2:END
```

Wenn Sie selbst Programme schreiben, werden Sie sicherlich schon oft Probleme beim Erstellen der Print-Befehle gehabt haben, da es

a) nicht sehr übersichtlich und
b) nicht so schnell

ist, mit PRINT-Befehlen zu arbeiten.
SCREEN-COPY wird Ihre Mühen drastisch verringern.
Tippen Sie es ab und starten Sie es mit

```
RUN 2 (!)
```

Nun haben Sie einen Cursor und können auf dem Bildschirm Ihre Bildschirm-Maske erstellen.
Nachdem Sie alles fertig erstellt haben, gehen Sie auf die letzte Zeile und drücken RETURN.
Nach kurzer Zeit werden Sie nach der Anfangszeile gefragt.
Geben Sie die gewünschte Zahl ein.
Nun können Sie sich zurücklegen und entspannen, der Computer erledigt den Rest für Sie.
Er druckt der Reihe nach die Bildschirminhalte aus und übernimmt sie Zeile für Zeile ins Programm.
Nach Beendigung des Programms haben Sie ab der eingegebenen Zeile Ihren erstellten Bildschirm als PRINT-Anweisungen.
Sie haben jetzt die Möglichkeit, noch einen Bildschirm zu erstellen und unter einer anderen Zeilennummer abzuspeichern.
Sie können aber auch das Programm SCREEN-COPY löschen und nur die PRINT-Anweisungen übrig lassen.

Da es unpraktisch wäre, SCREEN-COPY zu löschen, indem man der Reihe nach die Zahlen von 1-25 und RETURN drückt, haben wir uns auch dafür ein Programm ausgedacht.
Wenn Sie also SCREEN-COPY löschen wollen, so tippen Sie ein:

```
0 GOTO27
26 POKE254,1
27 X=PEEK(254):PRINT"□"X
28 X=X+1:POKE254,X:IFX=27THENPRINT"SCREEN-COPY GELOESCHT":EN
D
29 POKE631,19:POKE632,13:POKE633,82:POKE634,213:POKE635,13:P
OKE198,5:END
```

RUN 26

Jetzt wird blitzschnell SCREEN-COPY gelöscht, und Sie müssen nur noch per Hand die Zeilen 0,26,27,28 und 29 löschen.
Dann haben Sie nur noch Ihre PRINT-Zeilen.
Wir halten das für eine sehr komfortable Art, Bildschirme zu erstellen, auch wenn Sie einen Nachteil hat:
Farbänderungen des Cursors, Revers on und Revers off müssen nachräglich eingefügt werden.
Dieser Nachteil ließe sich zwar beheben, der Aufwand wäre jedoch dem Nutzen gegenüber nicht gerechtfertigt.

```
1 GOTO10
2 PRINT"■":OPEN1,0
3 INPUT#1,A$:IFPEEK(214)=24THEN5
4 PRINT:GOTO3
5 FORZ=0TO999:POKE24576+Z,PEEK(1024+Z):NEXTZ
6 INPUT"■ANFANGS-ZEILE";AZ:IFAZ<30THEN6
7 HB=INT(AZ/256):LB=AZ-256*HB
8 POKE828,LB:POKE829,HB
9 POKE830,0:POKE831,96
10 SP=PEEK(830)+256*PEEK(831)
11 FORZ=0TO39:A=PEEK(SP+Z)
12 GOSUB21
13 AU$=AU$+CHR$(B):NEXTZ
14 ZE=PEEK(828)+256*PEEK(829)
15 PRINT"■"ZE"?"CHR$(34);AU$;CHR$(34)";"
16 SP=SP+40:IFSP>25536THEN20
17 HB=INT(SP/256):LB=SP-256*HB:POKE830,LB:POKE831,HB
18 ZE=ZE+10:HB=INT(ZE/256):LB=ZE-256*HB:POKE828,LB:POKE829,H
B
19 POKE631,19:POKE632,13:POKE633,82:POKE634,213:POKE635,13:P
OKE198,5:END
20 POKE631,19:POKE632,13:POKE198,2:END
21 A=AAND127:IFAAND32THEN24
22 IFAAND64THENB=AOR32:RETURN
23 B=AOR64:RETURN
24 IFAAND64THENB=AAND63OR128:RETURN
25 B=A:RETURN
```

Da das Programm nicht ganz so einfach ist, einige Erklärungen:

Zeilen 2-4: Bildschirm wird als Datei eröffnet. In Zeile 3 wird nachgeschaut, ob der Cursor in der untersten Zeile ist, wenn RETURN gedrückt wird.

Zeile 5 : Der Bildschirminhalt wird abgespeichert, da er sich bei dem weiteren Geschehen verändert.

Zeile 8 : Die Anfangszeilennummer wird eingegeben, auf ihre Richtigkeit überprüft, in LOW- und HIGH-BYTE zerlegt und in die Adressen 828-829 (Kassettenpuffer) geschrieben.

Zeile 9 : Die Adresse, auf der der gespeichert Bildschirminhalt beginnt, zerlegt in LOW- und HIGH-BYTE.

Zeile 13 : Der String für die Zeile wird gebildet.

Zeile 14 : Die Zeilennummer wird berechnet.

Zeile 15 : Die fertige Zeile wird auf den Bildschirm gebracht.

Zeile16-18: Die neue Zeilennummer und die neue Bildschirminhaltsadresse wird berechnet, in LOW- und HIGH-BYTE zerlegt und abgespeichert.
Diese beiden Werte müssen so abgespeichert werden, da das Programm sich selbst unterbricht, und bei einem neuen Start die Variablen verloren wären.

Zeile 19 : Das Programm unterbricht sich selbst und nimmt die soeben generierte Zeile ins Programm auf, indem der Cursor an den Bildschirmanfang springt und RETURN gedrückt wird.
Danach wird das Programm wieder automatisch gestartet, indem RUN ausgegeben wird.
Diese ganzen Operationen geschehen durch POKEn der entsprechenden ASCII-Werte in den Tastatur-Puffer.

Zeile21-25: Ein Unterprogramm, das Bildschirm-Code-Werte in ASCII-Werte umwandelt. Diese Umwandlung muß erfolgen, da die Werte auf dem Bildschirm und in dem Bereich, in dem sie gesichert wurden, als Video-Code-Werte vorliegen. Die generierte Bildschirmzeile (AU$) muss die Werte aber als ASCII-Werte bekommen.
Dieses Unterprogramm kann auch für andere Zwecke gebraucht werden.

BASIC-TIPS

Da Sie, wenn Sie dieses Buch lesen, wahrscheinlich schon fortgeschrittener BASIC-Programmierer sind, wird Sie dieses Kapitel wahrscheinlich überraschen. Lesen Sie es aber trotzdem durch, da wir sicher sind, daß auch Sie etwas dazulernen können.
Wenn Sie schon einige Spiele in BASIC geschrieben haben, so wird Ihnen bestimmt ein großer Nachteil von BASIC aufgefallen sein: BASIC ist langsam.
Manchmal ist der entsprechende Programmierer aber auch nicht ganz unschuldig daran...
Wir wollen Ihnen hier zeigen, wie Sie Ihre BASIC-Programme schneller machen können.

BENUTZEN SIE EINEN PUNKT STATT DER NULL

Es wird Sie vielleicht überraschen: Wenn der Computer einen (alleinstehenden) Dezimal-Punkt in einem BASIC-Programm findet, so interpretiert er diesen automatisch als Null im Fließkommaformat.
Doch wieso sollte der Dezimal-Punkt anstatt einer Null verwendet werden?
Nun, die "Null" im BASIC-Programm ist für den Computer gar keine Zahl, sondern ein ASCII-Zeichen. Dieses ASCII-Zeichen muß erst in eine Zahl verwandelt werden, was natürlich Zeit kostet.
Wenn Sie also den Dezimal-Punkt statt dessen verwenden, so braucht der Computer keine Umwandlung vorzunehmen und spart so viel Zeit.
Beispiel:

1.PROGRAMM:

```
10 TI$="000000"
20 FORX=1TO100
30 A=A+0
```

```
                    40 NEXTZ
                    50 PRINTTI
```

2.PROGRAMM:

```
                    10 TI$="000000"
                    20 FORX=1TO100
                    30 A=A+.
                    40 NEXTX
                    50 PRINTTI
```

Noch öfter kann man den Dezimal-Punkt in IF-Anweisungen verwenden:

```
Statt               10 IFA=0THEN20
nehmen Sie          10 IFA=.THEN20
```

Denken Sie nur daran, wie oft Sie die 1.Version schon in Programmen benutzt und welche Zeit Sie dabei vergeudet haben!

BENUTZEN SIE VARIABLEN STATT ASCII-ZEICHEN

Wenn Sie eine Zahl wie folgt schreiben, so wird diese Zahl im BASIC-Speicher als ASCII-Werte abgespeichert:

```
                    10 POKE1024,1
```

Wird dieser Befehl jetzt ausgeführt, so muß der Computer diesen ASCII-String erst in eine Integer-Zahl, und dann in eine Fließkommazahl (mit der der Computer ausschließlich rechnen kann) umwandeln. Diese Umwandlung kostet erwartungsgemäß viel Zeit.
Es muß also ein Weg gefunden werden, den ASCII-String, den die Zahl darstellt, gleich als Fließkommazahl abzuspeichern. Dafür bietet sich die Variable an.
Beispiel:

1.PROGRAMM:

```
10 TI$="000000"
20 FORX=1TO100
30 A=A+100
40 NEXTX
50 PRINTTI
```

2.PROGRAMM:

```
10 TI$="000000":B=100
20 FORX=1TO100
30 A=A+B
40 NEXTX
50 PRINTTI
```

Der Unterschied kann ganz schön beachtlich werden, besonders bei größeren Zahlen, da dort der ASCII-String, der umgewandelt werden muß, immer länger wird, die Fließkommazahl dagegen immer noch gleichlang ist (nur die Werte der einzelnen Bytes werden geändert).

BENUTZEN SIE FLIEßKOMMA- STATT INTEGER-VARIABLEN

Wie Sie schon im vorigen Punkt gesehen haben, kostet die Umwandlung in Fließkomma-Zahlen viel Zeit.
Wenn Sie nun eine Integer-Variable (z.B. A%) benutzen, so muß auch diese Zahl in eine Fließkomma-Zahl umgewandelt werden. Benutzen Sie statt dessen gleich eine Fließkomma-Variable (z.B. A).
Beispiel:

1.PROGRAMM:

```
10 TI$="000000"
20 B%=100:C%=50:D%=25
30 FORX=1TO100
40 A%=A%+B%-C%-D%
50 NEXTX
60 PRINTTI
```

2.PROGRAMM:

```
10 TI$="000000"
20 B=100:C=50:D=25
30 FORX=1TO100
40 A=A+B-C-D
50 NEXTX
60 PRINTTI
```

BENUTZEN SIE FOR-NEXT-SCHLEIFEN

Sofern Sie es noch nicht machen, wollen wir Sie darauf hinweisen: FOR-NEXT-Schleifen sind schneller und einfacher zu programmieren als IF-Anweisungen.
Beispiel:

1.PROGRAMM:

```
10 TI$="000000":X=1
20 A=A+X
30 X=X+1:IFX=101THEN20
40 PRINTTI
```

2.PROGRAMM:

```
10 TI$="000000"
20 FORX=1TO100
30 A=A+X
40 NEXTX
50 PRINTTI
```

LEGEN SIE UNTERPROGRAMME AN DEN PROGRAMMANFANG

Unterprogramme solten immer am Anfang des BASIC-Programms stehen, da der Computer das BASIC-Programm vom Anfang her nach der gesuchten Zeilennummer untersucht. Je später diese Zeilennummer im BASIC-Programm zu finden ist, desto länger braucht der Computer.
Beispiel:

1,PROGRAMM:

```
10 TI$="000000"
20 FORX=1TO100
30 GOSUB100
40 NEXTX
50 PRINTTI
60 END
100 A=A+X
110 RETURN
```

2.PROGRAMM:

```
10 A=A+Z
20 RETURN
100 TI$="000000"
110 FORX=1TO100
120 GOSUB10
130 NEXTX
140 PRINTTI
```

Das zweite Programm ist mit "RUN 100" zu starten.
Der Unterschied macht sich hier noch nicht so bemerkbar, da die Programme nicht so lang sind, und der Computer deswegen auch nicht so viele Zeilen durchzusuchen hat, bis er die Richtige findet. Probieren Sie deswegen diesen Punkt einmal an längeren Programmen aus.

Wenn Sie Ihr Programm nach all diesen Punkten durchsuchen, können Sie es um einiges schneller machen.

ESCAPE-FUNKTION

Innerhalb von Anführungszeichen befinden Sie sich, wie Sie vielleicht bereits bemerkt haben, in einem besonderen Mode (engl. Quote Mode). Wenn Sie beispielsweise die Cursor-Steuertasten betätigen, erscheinen die entsprechenden Cursor-Steuerzeichen. Dasselbe gilt für die Farbtasten. Die Funktion der Steuertasten tritt erst bei Betätigung der PRINT-Anweisung ein.

Das ist zwar sehr hübsch, aber wie Sie aus eigener Erfahrung sicherlich wissen, ist es oft sehr aufwändig, wenn Sie aus diesem Modus wieder heraus wollen, um z.B. eine falsche Eingabe wieder zu korrigieren.

Diese kleine Routine schafft da Abhilfe. Mit einem Druck auf F-1 springt der Computer aus dem Hochkomma-Modus (engl. Quote-Mode) in den normalen Modus.

Aber es geht auch andersherum. Wenn Sie wieder in den Quote-Mode zurückwollen, brauchen Sie nur F-2 zu drücken.

Um zu klären, ob Sie im Quote-Mode sind oder nicht, wird dem jeweiligen Modus entsprechend ein Hochkomma in der rechten oberen Bildschirm-Ecke angezeigt/nicht angezeigt.

```
0 REM ESCAPE
10 FORI=40704TO40795
20 READA
30 S=S+A
40 POKEI,A
50 NEXT
60 IFS<>11008THENPRINT"?FEHLER IN DATAS":END
70 PRINT"DATAS OK"
80 SYS40704
90 DATA120,169,15,141,20,3,169,159,141
100 DATA21,3,133,56,88,96,72,165,203
110 DATA201,4,240,34,201,5,240,59,165
120 DATA212,208,4,165,216,240,13,169,34
130 DATA141,39,4,169,1,141,39,216,76,52
140 DATA159,169,32,141,39,4,104,76,49
150 DATA234,165,212,240,18,169,0,133
160 DATA212,133,216,169,20,141,119,2
170 DATA169,1,133,198,76,52,159,165,216
180 DATA240,226,76,60,159,169,255,133
190 DATA216,76,52,159
```

Das Maschinensprache-Listing wollen wir Ihnen auch nicht vorenthalten:

```
9F00   SEI
9F01   LDA #$0F
9F03   STA $0314
9F06   LDA #$9F
9F08   STA $0315
9F0D   STA $38
9F0D   CLI
```

```
9F0E   RTS
9F0F   PHA
9F10   LDA $CB
9F12   CMP #$04
9F14   BEQ 9F38
9F16   CMP #$05
9F18   BEQ $9F55
9F1A   LDA $D4
9F1C   BNE $9F22
9F1E   LDA $D8
9F20   BEQ $9F2F
9F22   LDA #$22
9F24   STA $0427
9F27   LDA #$01
9F29   STA #D827
9F2C   JMP $9F34
9F2F   LDA #$20
9F31   STA $0427
9F34   PLA
9F35   JMP $EA31
9F38   LDA $D4
9F3A   BEQ $9F4E
9F3C   LDA #$00
9F3E   STA $D4
9F40   STA $D8
9F42   LDA #$14
9F44   STA $0277
9F47   LDA #$01
9F49   STA $C6
9F4B   JMP $9F34
9F4E   LDA $D8
9F50   BEQ $9F34
9F52   JMP $9F37
9F55   LDA #$FF
9F57   STA $D8
9F59   JMP $9F34
```

BILDSCHIRMFARBEN ÄNDERN

Häufig passiert es einem, daß man irgendetwas auf dem Bildschirm nicht lesen kann, weil die Bildschirmfarben (Hintergrundfarbe, Rahmenfarbe, Farbe der Schrift) unglücklich gewählt sind. Das tritt besonders oft bei Benutzern von Schwarz-Weiß-Geräten auf. Mit Hilfe dieses Programms können Sie in einigen Maschinensprache-Programmen, aber selbstverständlich in (fast) allen Basic-Programmen und im Direktmodus die Hintergrund- und die Rahmenfarbe ändern.
Die Bildschirmrahmenfarbe wird durch den Druck auf F1 geändert. Mit F3 ändern Sie die Hintergrundfarbe des Bildschirms. Diese Routine liegt im IRQ. Das ist der Grund, warum ein Flimmern auf dem Bildschirm entsteht, wenn Sie auf F1 oder F3 drücken. Diese Routine fragt ab, ob eine dieser Tasten gedrückt wurde. War dies der Fall, so wird der Wert für Rahmen- oder Hintergrundfarbe um eins erhöht. Nun werden IRQ-Routinen aber etwa 50-60 mal in der Sekunde vom Computer angesprungen. Das bedeutet, bei gedrückter Taste wird die Farbe 50-60 mal erhöht.
Um die Routine so kurz wie möglich zu machen, haben wir darauf verzichtet, dieses Flackern abzuschaffen.
Um die richtige Farbkombination zu bekommen, müssen Sie mehrmals auf die entsprechende Taste drücken.
Wollen Sie diese Routine in einem anderen Programm gleichzeitig verwenden, sollten Sie zuerst diese Routine in den Computer laden und sie danach starten. Dann können Sie ein Programm ihrer Wahl hineinladen.
Probleme beim gleichzeitigen Gebrauch der Programme könnten sich ergeben, wenn das zweite Programm zu lang ist oder den Bereich des Speichers ab 40704, den diese Routine benötigt, anderweitig belegt, oder wenn es die Funktionstasten F1 und F3 benutzt.

Doch gucken wir uns jetzt den BASIC-Lader an:

```
10 FOR I = 40704 TO 40745
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S<>4625 THEN PRINT "?FEHLER IN DATAS" : END
70 PRINT "DATAS O.K."
80 SYS 40704
90 DATA 120,169,15,141,20,3,169,159,141
100 DATA 21,3,133,56,88,96,72,165,203
110 DATA 201,4,240,8,201,5,240,10,104,76
120 DATA 49,234,238,32,208,76,26,159,238
130 DATA 33 208,76,26,159
```

Als Profi sind Sie vielleicht auch am Maschinensprache-Listing interessiert, aber auch für Nicht-Profis könnte es interessant sein. Darum an dieser Stelle das Assembler-Listing.

```
9F00  SEI            : Interrupt aus
9F01  LDA #$0F       : ändern der Vektoren
9F03  STA $0314
9F06  LDA #$9F
9F08  STA $0315
9F0B  STA $38        : Routine abblocken
9F0D  CLI            : Interrupt an
9F0E  RTS            : Ende des 1.Teils
9F0F  PHA            : Register retten
9F10  LDA $CB        : Tastaturabfrage
9F12  CMP #$04       : F1 ?
9F14  BEQ $9F1E
9F16  CMP #$05       : F2 ?
9F18  BEQ $9F24
9F1A  PLA
```

```
9F1B  JMP $EA31       : setzt IRQ fort
9F1E  INC $D020       : erhöht Rahmenfarbe
9F21  JMP $9F1A
9F24  INC $D021       : erhöht Hintergrundfarbe
9F27  JMP $9F1A
```

Falls das Programm, das Sie gleichzeitig mit dieser Routine laufen lassen wollen, die Tasten F1 und F3 schon belegt hat, können Sie für die Routine auch andere Tasten nehmen. Statt 4 für F1 und 5 für F3 in Zeile 110 können Sie die Werte der Tasten ändern. Diese neuen Werte können Sie der Tastaturdecodierungs-Tabelle im Anhang entnehmen.

ZWEI BILDSCHIRME

Diese Routine ermöglicht es, zwei Bildschirme zu benutzen. Das Anwendungsgebiet für diese Routine ist riesengroß. Vor allen Dingen in der Grafik (Umschaltung Low-Res auf High-Res) findet diese Routine ihre Verwendung.
Wir wollen Ihnen hier diese Routine nur in Verbindung mit zwei Low-res-Grafikbildschirmen (normaler Bildschirm) zeigen, da dadurch die Routine leichter zu verstehen ist. Es bleibt Ihnen überlassen, diese Routine auf andere Verwendungszwecke zurechtzuschneidern.
Man kann mit dem zweiten Bildschirm viele seiner Programme stark verbessern. Z.B. kann man in Abenteuer-Spielen schon einen anderen Bildschirm aufbauen lassen, während der Spieler noch vor dem ersten Bildschirm ist. Aber auch bei anderen Anwendungen wie z.B. Textverarbeitung kann der zweite Bildschirm ungeahnte Vorteile mit sich bringen. Der Phantasie sind keine Grenzen gesetzt.
Doch nun zum eigentlichen Programm:
Tippen Sie und saven Sie es ab. Wichtig ist, daß Sie nach dem Starten des Programms NEW eingeben (siehe Assembler-Listing). Für die Nicht-Maschinen-Sprachler sei gesagt, daß der Computer sich sonst bei Eingabe eines Programms "verabschiedet".
Um in den zweiten Bildschirm zu gelangen, müssen Sie nur auf F1 drücken. Wenn Sie umgeschaltet haben, kann es Ihenn passieren, daß Ihr Cursor nicht mehr zu sehen ist. Aber keine Angst, der Computer hat sich nicht aufgehängt. Wenn Sie RETURN oder ähnliches drücken, erscheint er wieder. Auch ist der zweite Bildschirm beim ersten Umschalten mit diversen Zeichen vollgefüllt, da dieser Bereich vor dem Umschalten vom Basic-Speicher belegt wurde. Löschen Sie einfach den Bildschirm, und er wird voll zu Ihrer Verfügung stehen.

Der Farbspeicher ist für beide Bildschirme identisch. Dadurch werden die Farbinformationen von einem Bildschirm in den anderen mit übernommen. Um das zu verhindern, haben wir in die Routine einen Teil eingebaut, der den gesamten Farb-RAM auf die momentane Farbe des Cursors setzt.
Der Bildschirm-Speicher für den zweiten Bildschirm geht von 2048 ($0800) bis 3047 ($0BE7).
Gesagt werden muß noch, daß sich die Adressen der Zeiger für die Sprites ändern. Sie sind jetzt bei 3064 bis 3071.
Zur besseren Übersicht eine Tabelle:

		Mit Routine	
	Normalzustand	1.Bildschirm	2.Bildschirm
Bildschirmspeicher-Anfang	1024 ($0400)	1024 ($0400)	2048 ($0800)
Bildschirmspeicher-Ende	2023 ($07E7)	2023 ($07E7)	3047 ($0BE7)
Zeiger für Sprites	2040 ($07F8)	2040 ($07F8)	3064 ($0BF8)
Basic-Speicher-Anfang	2048 ($0800)	3072 ($0C00)	3072 ($0C00)
Farb-Speicher	55296 ($D800)	55296 ($D800)	55296 ($D800)
Erreichen des jeweiligen Zustandes	SYS 64738	Drücken von F-3	Drücken von F-1

Diese Routinen können Sie auch in Spielen zu benutzen. Es gibt dazu zwei Möglichkeiten:

a) Fordern Sie den Benutzer zum Drücken von F1 bzw. F3 auf
b) POKE 203,4 um in den zweiten Bildschirm zu gelangen
   POKE 203,5 um in den ersten Bildschirm zu gelangen

Nun der BASIC-Lader:

```
0 REM 2 Bildschirme
10 FOR I=40702 TO 40789
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S=11070 THEN PRINT"DATAS O.K.":POKE3072,0:
   SYS 40702: END
70 PRINT"FEHLER IN DATAS"
80 END
90 DATA169,12,133,44,120,169,17,141,20
100 DATA3,169,159,141,21,3,133,56,88,96
110 DATA165,203,201,4,208,13,169,37,141
120 DATA24,208,169,8,141,136,2,76,50
130 DATA159,201,5,208,36,169,21,141,24
140 DATA208,169,4,141,136,2,169,216,133
150 DATA252,160,0,132,251,173,134,2,145
160 DATA251,200,240,13,192,232,208,247
170 DATA166,252,224,219,208,241,76,49
180 DATA234,230,252,76,61,159
```

Und nun das Maschinen-Programm:

```
9EFE   LDA #$0C
9F00   STA $2C
9F02   SEI
9F03   LDA #$11
9F05   STA $0314
9F08   LDA #$9F
9F0A   STA $0315
9F0D   STA $38
9F0F   CLI
9F10   RTS
9F11   LDA $CB
9F13   CMP #$04
9F15   BNE $9F24
9F17   LDA #$25
9F19   STA $D018
9F1C   LDA #$08
9F1E   STA $0288
9F21   JMP $9F32
9F24   CMP #$05
9F26   BNE $9F4C
9F28   LDA #$15
9F2A   STA $D018
9F2D   LDA #$04
9F2F   STA $0288
9F32   LDA #$D8
9F34   STA $FC
9F36   LDY #$00
9F38   STY $FB
9F3A   LDA $0286
9F3D   STA ($FB),Y
9F3F   INY
9F40   BEQ $9F4F
9F42   CPY #$E8
9F44   BNE $9F3D
9F46   LDX $FC
```

```
9F48   CPX #$DB
9F4A   BNE $9F3D
9F4C   JMP $EA31
9F4F   INC $FC
9F51   JMP $9F3D
```

Noch zwei Tips für den Umgang mit zwei Bildschirmen:
Der erstellte Bildschirm könnte abgeSAVEt werden (Hardcopy für Floppy oder Datasette). Gehen Sie zu diesem Zweck folgendermaßen vor:

a) Notieren Sie sich die Inhalte der Adressen 45/46
b) Geben Sie folgende Werte ein:

```
POKE 43,0:POKE 44,X
```

X resultiert aus dem Bildschirm, den Sie abSAVEn wollen. Eine 4 SAVEt den ersten, eine 8 den zweiten Bildschirm.

c) Geben Sie weiterhin ein:

```
POKE 45,0:POKE 46,X+4
```

d) Geben Sie SAVE "(Name)",8 ein.
e) Nach dem SAVEen muß der BASIC-Anfang wieder zurückgesetzt werden. Dies geschieht durch:

```
POKE 43,1: POKE 44,12: POKE 3072,0

POKE 45,...: POKE 46,... (gemerkte Werte)
```

LAUFSCHRIFT IN MASCHINENSPRACHE

Im Kapitel über das Spiel finden Sie ein kleines Laufschriftprogramm in BASIC. Wir wollen Ihnen hier eine Laufschrift vorstellen, die in Maschinensprache geschrieben ist. Der große Unterschied zur BASIC-Routine liegt darin, daß diese Routine im IRQ liegt und so im Direktmodus, in BASIC-Programmen und sogar in einigen Maschinensprache-Spielen laufen kann.
Sie können Ihren eigenen Text eingeben. Er erscheint dann in der obersten Zeile des Bildschirms.
Der Text sollte nicht mehr als 40 Buchstaben umfassen. Es werden ausschließlich Buchstaben angenommen. Graphik-Symbole wandelt das Programm ab. Wollen Sie trotzdem Graphik-Symbole verwenden, so müssen Sie die Werte dieser Zeichen direkt in die entsprechenden Speicherstellen POKEn.
Der Text wird ab Speicherstelle 40448 abgespeichert.
Die Farbe des Textes können Sie ebenfalls frei verändern. Dazu brauchen Sie nach dem Start des Programmes nur folgenden Poke einzugeben:

```
POKE 40760,(Farbe)
```

Um die Geschwindigkeit, in der der Text über den Bildschirm läuft, bestimmen zu können, brauchen Sie lediglich folgendes in den Computer einzugeben:

```
POKE 40783,(Geschwindigkeit)
```

Normalerweise enthält diese Adresse den Wert 5. Sie können die Geschwindigkeit und die Farbe auch im Programm ändern. Der Farbwert steht an zweiter Stelle in der DATA-Zeile 260,

der Geschwindigkeitswert an siebter Stelle in Zeile 280. Danach muß jedoch auch die Prüfsumme in Zeile 60 entsprechend abgeändert werden.

Hier der BASIC-Lader:

```
0 REM LAUFSCHRIFT
10 FOR I=40704 TO 40789
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S<> 10107 THEN PRINT "?Fehler in
   Datas" : END
70 PRINT "Datas ok"
80 PRINT : POKE 19,1
90 INPUT "Text : ";A$
100 POKE 19,0: PRINT
110 FOR I=1 TO LEN ( A$ )
120 POKE 40448+I-1, ASC( MID$( A$,I,1 ))
    AND NOT 64
130 NEXT I
140 FOR I=LEN( A$ ) TO 40
150 POKE 40448+I,32
160 NEXT I
170 SYS 40704
180 INPUT "Neuer Text (j-n) ";A$
190 IF A$= "j" THEN 80
200 DATA 120,169,28,141,20,3,169,159,141
210 DATA 21,3,88,133,56,162,0,189,0,158
220 DATA 157,40,158,232,224,40,144,245
230 DATA 96,206,255,159,208,50,162,0,173
240 DATA 50,159,105,1,201,40,144,2,169,0
250 DATA 141,50,159,189,26,158,157,0,4
260 DATA 169,1,157,0,216,232,224,40,144
```

```
270 DATA 240,174,50,159,232,224,40,144,2
280 DATA 162,0,142,50,159,162,5,142,255
290 DATA 159,76,49,234
```

Um diese Routine in Verbindung mit einem anderen Programm gleichzeitig laufen zu lassen, müssen Sie zunächst diese Routine in den Computer hereinladen und starten. Anschließend kann das zweite Programm nachgeladen werden. Es wird hierbei zwar der o.g. BASIC-Lader überschrieben, die Laufschrift ist jedoch bereits in den IRQ eingebunden.

Für Interessierte hier das Maschinensprache-Listing der Routine:

```
9F00   SEI
9F01   LDA #$1C
9F03   STA $0314
9F06   LDA #$9F
9F08   STA $0315
9F0B   CLI
9F0C   STA $38
9F0E   LDX #$00
9F10   LDA $9E00,X
9F13   STA $9F28,X
9F16   INX
9F17   CPX #$28
9F19   BCC $9F10
9F1B   RTS
9F1C   DEC $9FFF
9F1F   BNE $9F53
9F21   LDX #$00
9F23   LDA $9F32
9F26   ADC #$01
9F28   CMP #$28
9F2A   BCC $9F2E
```

```
9F2C   LDA #$00
9F2E   STA $9F32
9F31   LDA $9E1B,X
9F34   STA $0400,X
9F37   LDA #$01
9F39   STA $D800,X
9F3C   INX
9F3D   CPX #$28
9F3F   BCC $9F31
9F41   LDX $9F32
9F44   INX
9F45   CPX #$28
9F47   BCC $9F4B
9F49   LDX #$00
9F4B   STX $9F32
9F4E   LDX #$05
9F50   STX $9FFF
9F53   JMP $EA31
```

Diese Routine ist nach dem Schema aufgebaut, das im Kapitel "...und wie man den IRQ programmiert" beschrieben wird. Wenn Sie wissen wollen, wie eine solche IRQ-Rotuine geschrieben wird, sollten Sie sich dieses Kapitel einmal näher ansehen!

DIE STOP-FUNKTION

Diese Routine arbeitet im IRQ. Mittels F1 können Sie den Computer "anhalten". Er wartet jetzt solange, bis Sie F3 drücken und fährt erst dann fort in seiner Arbeit. Sie können die Routine im Direktmodus als List-Stop benutzen oder in einem Basic-Programm oder auch in einigen Maschinensprache-Spielen als Stopper, falls das Telefon klingelt oder ähnlich Unvorhersehbares passiert. Sinnvoller ist diese Routine sicherlich als List-Stop. Sie können sich damit lange Programme viel einfacher und bequemer ansehen. Es bleibt einem das lästige BREAK-Drücken und danach das erneute Eingeben des LIST-Befehls erspart.
Sehen wir uns zunächst den BASIC-Lader an:

```
0 REM STOP-FUNKTION
10 FOR I=40704 TO 40752
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S<> 5628 THEN PRINT "?Fehler in
   Datas" : END
70 PRINT "Datas ok"
80 SYS 40704
90 DATA 120,169,15,141,20,3,169,159,141
100 DATA 21,3,133,56,88,96,72,165,203
110 DATA 201,4,240,4,104,76,49,234,169,1
120 DATA 59,72,169,41,72,8,72,138,72,152
130 DATA 72,76,49,234,165,203,201,5,208
140 DATA 235,240,229
```

Das Assembler-Listing sieht folgendermaßen aus:

```
9F00    SEI
9F01    LDA #$0F
9F03    STA $0314
9F06    LDA #$9F
9F08    STA $0315
9F0B    STA $38
9F0D    CLI
9F0E    RTS
9F0F    PHA
9F10    LDA $CB
9F12    CMP #04
9F14    BEQ $9F1A
9F16    PLA
9F17    JMP $EA31
9F1A    LDA #9F
9F1C    PHA
9F1D    LDA #29
9F1F    PHA
9F20    PHP
9F21    PHA
9F22    TXA
9F23    PHA
9F24    TYA
9F25    PHA
9F26    JMP $EA31
9F29    LDA $CB
9F2B    CMP #05
9F2D    BNE $9F1A
9F2F    BEQ $9F16
```

Sollten Sie die F1 und die F3 Taste schon belegt haben, so können Sie ganz einfach diese Routine auf eine andere Taste legen, indem Sie in den Adressen $9F12 und $9F2B die entsprechenden Werte abändern.
Für diejenigen, die keinen Maschinensprache-Monitor besitzen und den BASIC-Lader benutzen, bedeutet das, daß der erste Wert "4" in Zeile 110 und der Wert "5" in Zeile 130 geändert werden muß. Die entsprechenden Werte für andere Tasten erhalten Sie aus der Tabelle über die Tastatur-Belegung im Anhang.

ZUFALL?

Haben Sie sich eigentlich schon einmal die RND-Routine angeschaut?
Es wird Sie wahrscheinlich überraschen (oder hatten Sie vielleicht schon immer den Verdacht ?): Die Zahl, die da generiert wird, ist gar nicht zufällig gewählt, denn so etwas kann der logisch arbeitende Computer nicht.
Diese Zahl ist vielmehr in einem komplizierten Algorithmus ausgerechnet worden. Dieser Algorithmus sieht folgendermaßen aus:

1) Der letzte RND-Wert (abgespeichert in Adresse 139-143) wird in den Fließkomma-Akkumulator (FAC) geladen.
2) Der FAC wird mit der Konstanten 11879546 (abgespeichert im Bereich $E08D-$E091), multipliziert.
3) Zu dieser Zahl wird die Konstante 3.92767774E-08 (abgespeichert im Bereich $E092-$E096), addiert.
4) Jetzt werden einige Speicherinhalte vertauscht:

$65 wird getauscht mit $62
$63 wird getauscht mit $64
In $66 wird der Wert Null geschrieben
In $61 wird der Wert $80 geschrieben

5) Schließlich wird der FAC noch linksbündig gemacht, gerundet und zum Schluß wieder abgespeichert.

Noch eine Erklärung zu den Konstanten:
Wenn Sie sich den Bereich, in dem die Konstanten liegen, einmal anschauen, so werden Sie folgende (hexadezimale) Zahlen vorfinden:

$E08D 98 35 44 7A 00

```
$E092   68  28  B1  46  00
```

Wenn Sie sich nicht näher mit dem FAC auskennen, so werden Sie wahrscheinlich Probleme haben, aus diesen Zahlen die zwei Konstanten herauszufinden.
Folgendes Programm erledigt diese Aufgabe für Sie:

```
LDA #$LB
LDY #$HB
JSR $BBA2     :Zahl in den FAC
JSR $BDDD     :FAC in ASCII und ab $0100 ablegen
LDA #$00      :Low-Byte  Adresse des Strings
LDY #$01      :High-Byte Adresse des Strings
JSR $AB1E     :String ausgeben
RTS
```

LB-HB geben die Adresse an, ab der die (5) Bytes, die in den FAC kommen sollen, stehen. Wollen Sie den FAC selbst auslesen, so muß die Adresse $0061 lauten.
Als BASIC-Lader (im Kassetten-Puffer) sieht das Programm so aus:

```
10 FOR X=0 TO 17: READ A:POKE 828+X,A: NEXT X
20 DATA169,LB,160,HB,32,162,187
30 DATA32,221,189,169,0,160,1
40 DATA32,30,171,96
```

Gestartet wird die Routine mit SYS 828. Für LB-HB müssen Sie jedesmal die entsprechenden Werte einsetzen. Sie können diese Werte aber auch POKEn:

```
POKE 829,LB: POKE 831,HB
```

Einige Beispiele für LB-HB:

LB	HB	WERT
141	224	Konstante 1
146	224	Konstante 2
139	0	Letzter RND-Wert
97	0	FAC

Doch weiter zur RND-Routine.
Diese Routine kann auch in der Maschinensprache angesprungen werden. Will man das Maschinensprache-Äquivalent von RND(1) haben, so heißt die Einsprungadresse $E0BE (57534).
Ein direktes Äquivalent zu RND(-1) und RND(0) haben wir nicht gefunden. Probieren Sie aber ruhig einmal andere Einsprungadressen zwischen $E097 und $E0EF aus.
Eine Zufallszahl kann man auch in Maschinensprache erzeugen, indem man einen Zähler liest. Beispiele dafür sind Adresse 160-162 für die Uhr, und Adresse 53266 für die Zeile, die gerade auf dem Bildschirm (vom Elektronenstrahl) geschrieben wird.

RND kann man übrigens nicht nur für Spiele benutzen. Auch für Dateien, deren Daten nicht von jedem eingelesen werden sollen, kann man diese Routine sehr gut gebrauchen.
Wenn Sie den oberen Teil sorgfältig gelesen haben, haben Sie mitbekommen, daß der letzte RND-Wert in den Adressen 139-143 abgespeichert ist. Da der Computer den neuen (nächsten) RND-Wert immer mit Hilfe dieses Wertes berechnet, kann man RND ganz einfach auf einen festen Anfangswert setzen. Da die RND-Werte bei jedem Durchgang immer gleich berechnet werden, kann man die Daten mit einem Zahlen-Wert, der durch RND bestimmt worden ist, ver- und bei einem neuen Durchlauf wieder entschlüsseln.
Folgendes Programm veranschaulicht dies:

```
10 GOSUB 1000
20 PRINT CHR$(147): X=20480
30 POKE 204,0
40 GETA$: IFA$=""THEN 40
50 IFA$= CHR$(133) THEN 100
60 IF ASC(A$)<32 OR ASC(A$) >95 THEN 40
70 IF PEEK(207) THEN 70
80 POKE 204,1: PRINTA$;
90 FOR YY=0 TO 3: Y=Y+INT(RND(1)*40): NEXT :Y=Y+ASC(A$)
95 POKE X,Y: X=X+1: Y=0: GOTO 30
100 POKE X,0: GOSUB 1000
110 PRINT CHR$(147): X=20480
120 Z=PEEK(X): IF Z=0 THEN END
130 FOR YY= 0 TO 3: Z=Z-INT(RND(1)*40): NEXTYY
140 PRINT CHR$(Z);: X=X+1: Z=0: GOTO 120
1000 PRINT CHR$(147): FORX=0 TO 4
1010 INPUT A: IF A<0 OR A>255 THEN 1010
1020 POKE 139+X,A
1030 NEXT X
1040 RETURN
```

Nach Starten dieses Programms müssen Sie erst einmal fünf Werte eingeben. Nun können Sie einen beliebigen Text eintippen. Sobald Sie fertig sind, müssen Sie die F1-Taste drücken.
Nun kommt der zweite Teil. Sie geben wieder die fünf Werte ein, die Sie auch schon beim ersten Mal eingegeben haben, und der Computer wird Ihnen den Text wieder ausgeben.
Sollten die Werte nicht mit den ersten Werten übereinstimmen, so wird der Text mehr oder weniger verstümmelt.
Dieses Programm ist noch nicht absolut sicher, da kleine Abweichungen der Werte nicht berücksichtigt werden (das liegt an der INT-Funktion in Zeile 90 und 130). Deutlich wird aber hier, wie Sie Daten-Texte verschlüsseln können:

a) Setzen Sie den RND-Wert fest, indem Sie bestimmte Wer-

te (Code des Benutzers) in die Speicherstellen 139-143 POKEn.

b) Verschlüsseln Sie die Daten mittels der RND-Funktion. Wie diese Verschlüsselung erfolgt (welche Formel Sie benutzen), bleibt Ihnen überlassen.

c) Bei Einlesen der Daten muß ein Schlüssel eingegeben werden, mit dessen Hilfe RND wieder auf einen bestimmten Wert gesetzt wird.

d) Mittels RND werden die Daten wieder entschlüsselt. War der Schlüssel falsch, so werden auch die Daten falsch ausgegeben.

Durch diesen einfachen Trick ist es möglich, Texte und Zahlen so zu verschlüsseln, daß nur ein befugter Benutzer diese Daten wieder erreichen kann. Theoretisch kann man zwar auch an die Daten herankommen, indem man alle möglichen Werte für den Benutzer-Code durchgeht, in der Praxis würde das aber ziemlich lange dauern.
Noch ein Tip:
Wenn man es illegalen Benutzern noch schwerer machen will, kann man auch noch die Funktion: z.B. Y=Y+RND(1) , mit der die Daten verschlüsselt bzw. entschlüsselt werden, vom Benutzer eingeben lassen. Da hier die Möglichkeiten im Gegensatz zum Code unbegrenzt sind, dürfte es unmöglich sein, Daten, die so verschlüsselt sind, unbefugt zu bekommen.

Noch einmal kurz zusammengefaßt:

Zur Erlangung einer Pseudo-Zufallszahl verwendet man in BASIC den Befehl RND. Diesem Befehl folgt zwingend ein Argument. Die Wahl dieses Argumentes ist keinesfalls unbedeutend. Man unterscheidet vielmehr zwischen positivem und negativem Argument sowie Null.
Ein positives Argument erzeugt eine Pseudo-Zufallszahl basierend auf dem vorangegangenen Zufallswert. Ein negatives

Argument setzt die abgespeicherte Zufallsbasis, die über die nächste Zufallszahl entscheidet, (Adressen 139-143), in Abhängigkeit zum Betrag des negativen Arguments.
Um eine wirklich zufällige Zahl durch RND(x) zu erlangen, sollte man deshalb die Basis neu setzen. Dies geschieht mit folgender Zeile im Programm:

```
10 X=RND(-TI): X=INT(RND(1)*...
```

MODIFIZIERTES INPUT

In bezug auf Eingabe-Befehle sieht es beim C 64 recht mager aus: Es gibt standardmäßig nur die Grundbefehle GET und INPUT.
Für sehr viele Anwendungen reichen diese beiden Befehle jedoch nicht aus: Bei GET fehlt der Cursor. Das INPUT-Fragezeichen, das zwangsläufig bei diesem Befehl erscheint, bereitet Kummer. Manche Satzzeichen nimmt INPUT erst gar nicht an (da sie bestimmte Funktionen ausüben).

Im nun folgenden wollen wir Ihnen in Sachen Eingabe ein paar Tips geben, wie die oben genannten Mängel wenigstens teilweise umgangen werden können.
Da wäre zum Beispiel das Fragezeichen, das so charakteristisch für die INPUT-Anweisung ist. Aber was tun, wenn das Fragezeichen fehl am Platze ist, wenn es sich beim INPUT um eine Aufforderung handeln soll, etc. ?
Um das Fragezeichen des INPUTs auszuschalten, gibt es mehrere Möglichkeiten:

```
10 POKE 19,1: INPUT"RETURN DRUECKEN !";A$:PRINT:POKE 19,0
```

oder

```
10 OPEN 1,0
20 INPUT#1,A$
30 CLOSE 1
```

Man eröffnet einfach die Tastatur als Peripherie-Gerät.

Schwieriger wird es, wenn man neben den normalen Zeichen auch Satzzeichen wie Kommata, etc. eingeben können muß, was beispielsweise für Textverarbeitung notwendig ist.
Hierzu haben wir uns ein nicht allzu langes BASIC -

Äquivalent einfallen lassen. Es ist schnell genug, um auch für Textverarbeitung eingesetzt werden zu können.
Gegenüber dem normalen INPUT bietet es folgende Besonderheiten:

- Es können einem String beliebig viele Elemente zugeordnet werden. Dazu muß lediglich die DIM-Anweisung entsprechend verändert werden.
- Es können auch Steuerzeichen eingegeben werden, die bei der Ausgabe des Strings berücksichtigt werden.
- Es können lediglich Texteingaben gemacht werden. Das Ergebnis der Abfrage ist in IN$ gespeichert. Durch VAL dürften auch Zahleneingaben kein Problem sein.

Hier zunächst das Listing:

```
10 REM SIMULIERTES INPUT
20 FR$="WIE LAUTET DIE EINGABE ??":GOSUB60000:REM AUFRUF DER
 ROUTINE
25 PRINTIN$
30 END
40 :
50 :
60 :
60000 REM SIMULIERTES INPUT (ROUTINE)
60010 DIM IN$(1000):PRINTFR$;" ";
60050 POKE204,0
60051 POKE212,1:GETIN$:IFIN$=""THEN60051
60052 POKE 207,0
60055 IFIN$=CHR$(13)THENPOKE204,1:PRINT" ":IN$="":GOTO60100:
REM RETURN-TASTE
60056 IFIN$=CHR$(20)THENGOTO60120
60060 PRINTIN$;
60070 IN$(IN)=IN$:IN=IN+1
60080 GOTO60050
60100 FORSL=0TOIN-1
60101 IN$=IN$+IN$(SL)
60102 NEXT:RETURN
60120 IFIN=0THENGOTO60050
60125 POKE212,0:POKE207,0:PRINTCHR$(157)CHR$(32)CHR$(32)CHR$
(157)CHR$(157);
60130 IN=IN-1:IN$(IN)="":GOTO60050
```

Verwendete wichtige Adressen:

212 : Hochkomma-Flag
204 : Cursor ein-ausschalten

Wichtige Variablen:

IN$ Ergebnis der Abfrage
FR$ Fragestring (wird bei Aufruf der Routine auf den Bildschirm ausgegeben)

Eine weitere Möglichkeit, in einer Input-Anweisung ein Komma benutzen zu können, bietet folgende Basic-Zeile:

```
10 POKE 198,1 : POKE 631,34 : INPUT A$
```

Innerhalb des Hochkommas werden die Kommata nicht als Kennzeichen sondern als reguläre Eingabe interpretiert und somit akzeptiert. In den Tastaturpuffer (Adresse 631-640) wird ein Hochkomma (Code 34) geladen. Über Adresse 198 wird dies dem Computer mitgeteilt.
Bei der anschließenden INPUT-Anweisung wird ein Hochkomma ausgegeben. Ab sofort können auch Kommata eingegeben werden. Das ausgegebene Hochkomma wird später im String nicht berücksichtigt und beeinflußt auch nicht den LEN-Befehl.

FLOPPY-TRICKS

Hardwaremäßig hat der C-64 die Möglichkeit, bis zu 8 Floppy Disks anzuschließen. Für diese Floppys sind die Kanäle 8-15 reserviert. Da jede Floppy aber vom Werk aus auf die Geräte-Nummer 8 eingestellt ist, muß diese Nummer geändert werden.
Folgendes Programm soll Ihnen dabei helfen:

```
10 INPUT"ALTE NUMMER";AN
20 INPUT"NEUE NUMMER";NN
30 OPEN1,AN,15
40 PRINT#1,"M-W"CHR$(119);CHR$(0);CHR$(2);CHR$(NN+32);CHR$(
   NN+64)
50 CLOSE1
```

Sollten Sie also mehrere Floppys anschließen wollen, so gehen Sie folgendermaßen vor:
Schließen Sie eine Floppy an und ändern Sie die Geräte-Adresse in eine Nummer ungleich 8 um. Nun schließen Sie die zweite Floppy an. Da diese hardwaremäßig die Nummer 8 hat, haben beide Floppys unterschiedliche Adressen, und können jetzt eindeutig angesprochen werden.
Nun ändern Sie auch diese Floppy-Nummer um, usw.
Wollen Sie mehrere Floppys benutzen, so ist dieses Programm noch besser:

```
10 PRINT"WIE VIELE FLOPPYS WOLLEN SIE ANSCHLIESSEN"
20 INPUTAY
30 IFAY<1ORAY>8THEN2O
40 AN=8:FORY=1TOAY
50 PRINT"MACHEN SIE NUN FLOPPY NR. "Y"AN"
60 GETA$:IFA$=""THEN6O
```

```
70 OPEN1,AN,15
80 PRINT#1,"M-W"CHR$(119);CHR$(0);CHR$(2);CHR$(39+Y);
   CHR$(71+Y)
90 CLOSE1:PRINT"DIESES FLOPPY HAT DIE NUMMER"AN
100 AN=AN+1:NEXTY
```

Diese Art der Floppy-Umstellung hat jedoch ein entscheidenden Nachteil: Sobald das Floppy wieder ausgeschaltet wird, ist die Geräte-Nummer wieder 8.
Es gibt jedoch auch die Möglichkeit, die Floppy-Nummer hardwaremäßig zu ändern. Da man bei dieser Methode aber nur die Wahl zwischen Geräte-Nummern von 8-11 hat, muß man bei mehr als vier Geräten die Softwarelösung vorziehen. Vor jedem Arbeiten müssen also erst die Floppys initialisiert werden. Dazu würde sich am besten ein EPROM eignen, das man mit dem entsprechenden Programm und einem Autostar versehen hat.
Man kann aber auch das zweite hier vorgestellte Programm vor jedem Arbeiten laden.
Doch kommen wir zur Hardwarelösung. Da bei dieser Operation das Floppy geöffnet werden muß, sollten Sie sich darüber im klaren sein, daß damit die Garantie des Gerätes verfällt.
Haben Sie jedoch keine Angst: Bei vorschriftsmäßigem Arbeiten passiert der Floppy nichts.
Ziehen Sie also zuerst einmal den Netzstecker aus Ihrem Gerät. öffnen Sie es nun indem Sie die vier Schrauben am Boden lösen (vorsichtig, da sonst der Schreib-Lese-Kopf der Floppy durch die Erschütterung zerstört werden kann).
Heben Sie danach den Deckel ab.
Nun haben Sie die Platine vor sich. Ungefähr in der Mitte befinden sich mehrere größere ICs. Suchen Sie zunächst einmal die CPU, ein 6502. Direkt darunter befindet sich ein Ein-Ausgabe-IC, ein 6522. Von diesem IC gehen Sie etwas in Richtung der Vorderseite des Floppys. Sie stoßen dann auf einen Elektrolyt-Kondensator. Auf der Platine trägt er die Bezeichnung C64. Direkt neben diesem Kondensator befinden

sich zwei Lötbrücken. Sie haben die Form von zwei Halbkreisen, die durch eine schmale Verbindung miteinander verbunden sind.
Abhängig davon, welche Sie durchschneiden, erhalten Sie folgende Geräte-Nummern:

Getrennter Halbkreis	Geräte-Nummer
Keiner	8
1 (der Rechte)	9
2 (der Linke)	10
1 & 2	11

Nach dieser Operation, die sich komplizierter anhört, als sie tatsächlich ist, behält Ihre Floppy die eingestellte Geräte-Nummer 9 auch noch nach dem Ausschalten bei.

Da Sie Ihre Floppy jetzt schon einmal geöffnet haben, wollen wir Ihnen noch etwas anderes zeigen:
Normalerweise kann man einfache Disketten auch beidseitig benutzen, sofern man eine zweite Einkerbung vornimmt. Es geht aber auch anders.
An der einen Längsseite der Platine befinden sich zahlreiche Steckanschlüsse. Der größte davon muß die Bezeichnung P6 tragen. An diesem Anschluß befindet sich ein oranges und ein violettes Kabel. Zwischen diese beiden Kabel müssen Sie einen Schalter setzen, den Sie aus dem Floppy-Gehäuse herausführen. Nun können Sie Ihre Disketten ohne Einkerbung beidseitig benutzen, da die Lichtschranke, die normalerweise die Einkerbung überprüft, durch einen geschlossenen Schalter überbrückt wird.
Nun wollen wir Ihnen eine kleine Routine zeigen, die Sie in jedes Programm einbauen sollten, das mit dem Floppy arbeitet.
Wenn Sie von einem Programm aus Dateien abSAVEn, etc., und

Sie haben vergessen, das Floppy einzuschalten, so stürtzt das Programm mit Ausgabe der Fehlermeldung "device not present" ab. Viel besser wäre es doch, wenn der Benutzer darauf aufmerksam gemacht werden würde, daß sein Floppy nicht eingeschaltet ist.
Folgende Routine schaut nach, ob das Floppy eingeschaltet ist. Wenn ja, so wird die Speicherstelle $FF (255) auf Null gesetzt, andernfalls enthält sie den Wert 5. Diese Werte können nun von Ihrem Programm abgefragt werden, und das Programm kann entsprechend darauf reagieren.
Wir haben die Routine in den Kassettenpuffer gelegt, da dieser beim Floppy-Betrieb ungenutzt ist.

```
033C   LDA #$01    :Länge des Filenamens
033E   LDX #$D0    :Adresse low
0340   LDY #$FF    :Adresse high
0342   JSR $FFBD   :Filenamenparameter setzen
0345   LDA #$01    :logische Filenummer
0347   LDX #$08    :Geräte-Nummer
0349   LDY #$00    :Sekundär-Adresse
034B   JSR $FFBA   :Fileparameter setzen
034E   JSR $FFC0   :Eröffnen des Files (OPEN)
0351   BCS $0355   :keine Antwort des Geraetes
0353   LDA #$00    :Gerät vorhanden
0355   STA $FF     :Flag speichern
0357   LDA #$01    :logische Filenummer
0359   JSR $FFC3   :Schließen des Files (CLOSE)
035C   RTS
```

Wir haben die Adresse des Filenamens ($FFD0) gewählt, da in dieser Adresse der Wert 36 steht, was dem ASCII-Wert für "$" entspricht. Man hätte auch $FFE5 wählen können.
Dann würde der Filename "*" lauten. Dies wäre jedoch unzweckmäßig, da ja nicht irgendein Name gesucht werden kann. Es käme sonst zur Fehlermeldung "file not found" .

Als BASIC-Lader:

```
10 FORY=OTO32:READA:POKE828+Y,A:NEXTY
20 DATA169,1,162,208,160,255,32,189,255
30 DATA169,1,162,8,160,0,32,186,255
40 DATA32,192,255,176,3,169,0,133,255,169,1,32,195,255,96
```

Die Routine müssen Sie von Basic her mit SYS 828 aufrufen.

Und zum Schluß noch eine kleine Routine, die den Fehler-Kanal der Floppy liest.
Wie sich dies von BASIC aus machen läßt, können Sie unter "Fehlerkanal auslesen" in diesem Kapitel nachlesen.
Wir wollen nun diese Abfrage in Maschinensprache umformen...

Wenn Sie sich im Kernal auskennen (siehe Kapitel KERNAL), wissen Sie, daß es eine spezielle OPEN-Routine gibt. Schwieriger wird es schon mit dem Befehl INPUT#1. Doch im Kernal gibt es eine Routine, mit der man ein Zeichen vom IEC-Bus holen kann.
Wenn Sie sich die Routine anschauen, wird Ihnen wahrscheinlich auffallen, daß die OPEN-Routine gar nicht benutzt wird.
Da es in diesem Fall auch anders ging, haben wir uns zu dieser Lösung entschlossen.
Die Routine ist wieder im Kassettenpuffer.

```
033C   LDA #$08     :Geräte-Nummer für Floppy
033E   STA $BA      :Abspeichern
0340   JSR $FFB4    :TALK senden(Floppy soll senden)
0343   LDA #$6F     :Sekundär-Adresse
0345   STA $B9      :Abspeichern
0347   JSR $FF96    :Sekundär-Adresse nach TALK senden
034A   JSR $FFA5    :Zeichen von der Floppy holen
034D   JSR $FFD2    :Zeichen ausgeben (BSOUT)
0350   CMP #$0D     :Return?
```

```
0352   BNE $034A     :Nein, also weiter
0354   JSR $FFAB     :UNTALK senden (Schluß mit Senden)
0357   RTS           :Zurück
```

Wahrscheinlich werden Sie sich fragen, wieso die Sekundär-Adresse nicht wie in der Basic-Version dieses Programms 15 lautet. Nun, die 15 ist in dieser Adresse enthalten ($0F). Die andere Adresse ($60) ist für die Floppy. Normalerweise wird $60 in der OPEN-Routine zu der Sekundär-Adresse dazugerechnet. Da die OPEN-Routine hier aber nicht benutzt wird, muß dieser Wert gleich dazugerechnet werden.

Als BASIC-Lader:

```
10 FORX=0TO27:READA:POKE828+X,A:NEXTX
20 DATA169,8,133,186,32,180,255,169,111,133,185,32,150,255
30 DATA32,165,255,32,210,255,201,13,208,246,32,171,255,96
```

Aufgerufen werden muß diese Routine wieder mit SYS 828.

Da das hier keine Programmsammlung werden soll, haben wir einige Routinen, die nur für Floppy-Benutzer interessant sind, nicht aufgeführt. Versuchen Sie sich doch selbst einmal daran. Schreiben Sie z.B. die letzte Routine so um, daß die OPEN-Routine benutzt wird, oder schreiben Sie eine Routine, die das Directory von der Floppy liest, ohne ein gerade laufendes Programm zu unterbrechen.

Und zum Schluß noch einige Tips:
Wenn Sie nach einem bestimmten Programm suchen, aber nicht wissen, auf welcher Diskette es sich befindet, dürfte es für Sie sehr nützlich sein, zu wissen, daß es möglich ist, nur bestimmte Teile des Directorys zu laden:

```
LOAD "$0:(name)",8
```

Ist das gesuchte Programm nicht auf dieser Diskette, so wird nur der Disk-Header (der Name der Disk) angezeigt.
Sie können hier auch den Stern benutzen:

```
LOAD "$0:PAS*",8
```

Hier werden alle Programme auf dieser Diskette aufgelistet, die mit "PAS" beginnen.

Sie wissen wahrscheinlich, daß man ganz einfach Programme, die sich auf Band befinden, laden und automatisch starten kann, indem man SHIFT & RUN-STOP gleichzeitig drückt.
Das geht aber auch mit Disk:

```
LOAD "GRAFIK",8:(SHIFT & RUN-STOP)
```

Sie müssen einfach hinter der normalen LOAD-Eingabe statt "RETURN" "SHIFT+RUN-STOP" drücken. Es erscheint "LOAD" und das betreffende Programm wird geladen und automatisch gestartet.
Auch hier kann natürlich wieder der Stern verwendet werden:

```
LOAD "*",8:(SHIFT+RUN-STOP)
```

Nun wird das erste Programm, das sich auf der Diskette befindet, geladen und gestartet.

Man kann beim Einladen des Directorys auch nur bestimmte Programm-Typen einladen lassen:

```
LOAD "$*=T",8
```

T muß von Ihnen folgendermaßen gewählt werden:

T	FILE-TYPE
P	PROGRAMM
S	SEQUENTIELL
R	RELATIV
U	USER

Diese kann man auch für alle anderen Floppy-Befehle benutzen. So ist es z.B. möglich, alle Programm-Files zu löschen, während alle anderen File-Typen auf der Diskette bleiben:

```
OPEN15,8,15
PRINT#1,"S:*=U"
```

Hier werden alle User-Files gelöscht.

Versuchen Sie auch ruhig einmal, verschiedene Tips miteinander zu kombinieren.

3. SOFTWARESCHUTZ

MANIPULATION DER LIST-FUNKTION

Der LIST-Befehl ist wohl einer der meistbenutzten Befehle des BASIC-Programmierers. Mit seiner Hilfe lassen sich Programme "durchleuchten".
Manchem Programm-Autor ist dies aber gar nicht recht, weil so einerseits Kenn- und Codewörter einfach ausgelesen werden können und zweitens das Programm verändert werden kann.

Nachfolgend soll einiges über die LIST-Funktion gesagt werden. Wie beispielsweise das LISTen eines Programmes wirkungsvoll verhindert wird, und anderes mehr.

LISTEN OHNE ZEILENNUMMERN

Durch das Verändern der Adresse 22 (Zeiger auf den temporären Stringstapel) läßt sich ein eigenartiger Effekt erzielen: Die LIST-Funktion wird zwar ordnungsgemäß durchgeführt, Zeilennummern werden jedoch unterschlagen.

```
POKE 22,35
```

ruft diese Veränderung hervor. Mit Hilfe von

```
POKE 22,25
```

läßt sich wieder der Ausgangszustand herstellen.
Aber vorsichtig: Naturgemäß haben solche kleinen Manipulationen immer einen kleinen Nebeneffekt, über den man sich im klaren sein sollte:
Solange die Adresse 22 den Wert 35 enthält, werden sämtliche PRINT-Anweisungen unterschlagen. Jede Fehlermeldung (SYNTAX ERROR, etc.) stellt wieder den Ausgangszustand her.

Soll ein Programm ohne Zeilennummern auf dem Drucker

ausgegeben werden, so ist zu beachten, daß die letzte Zeile des Listings unterschlagen wird. Sie erscheint erst nach Eingabe von:

PRINT#1 (bzw. benutzte Dateinummer)

Selbstverständlich ist es auch möglich, andere Werte in die Adresse 22 zu schreiben. Wir haben folgende Effekte beobachtet:

POKE 22,25.....normal
POKE 22,32.....Zeilennummern werden unlesbar
POKE 22,33.....An Stelle der Zeilennummern !-Zeichen
POKE 22,34.....?FORMULA TOO COMPLEX ERROR, wieder Normalzustand
POKE 22,35.....Zeilennummern werden gänzlich unterschlagen.

Listschutz - Abschalten der LIST-Funktion

Manchmal ist es nützlich, die Möglichkeiten des LIST-Befehls ganz zu unterbinden. Auch hierfür gibt es mehrer Möglichkeiten:

1. Die folgende REM-Anweisung sorgt dafür, daß sämtliche folgenden Programmzeilen nicht mehr aufgelistet werden können. Statt dessen wird die Fehlermeldung SYNTAX ERROR ausgegeben:

```
10 PRINT "LISTSCHUTZVARIANTE 1"
20 REM (SHIFT & L)
30 PRINT "NICHT MEHR AUFLISTBAR !"
```

Diese Art des Listschutzes ist jedoch leicht zu durchschauen und somit leicht wieder zu entfernen (Es sei denn, man kombiniert mehrere Schutzvarianten, beispielsweise die

Manipulation der BASIC-Zeilennummern miteinander, so daß die REM-Anweisungen nicht entfernt werden können).
Wenn ca. alle 5 Zeilen im Programm eine solche REM-Anweisung eingebaut wird, so wird das Entfernen dieser Zeilen zur Tortur.

Die zweite Variante ist auf den ersten Blick gar nicht als Listschutz zu erkennen. Geben Sie dazu folgendes Beispielprogramm ein:

```
10 REM LISTSCHUTZVARIANTE 2
20 PRINT"GESCHUETZT": REM""
```

Anschließend fahren Sie mit Ihrem Cursor hinter das REM in Zeile 20. Drücken Sie so oft die INST-Taste, wie Buchstaben in dieser Zeile sind (in diesem Fall 26 Mal !).
Drücken Sie anschließend ebenso oft die DEL-Taste.
Hinter der REM-Anweisung in Zeile 20 befnden sich nun 26 reverse "T"s. Jedes "T" löscht ein Zeichen dieser Zeile.
Beim Auflisten wird die geschützte Zeile zwar ausgegeben, durch die reversen "T"s jedoch sofort wieder gelöscht.
Sollten dennoch Teile der Zeile erscheinen, so wurden zu wenig "T"s hinter die REM-Anweisung gebracht.
Empfehlenswert ist dieser Schutz allerdings nur bei Zeilen mit einer Länge kleiner als 10 Zeichen die REM-Anweisung ausgenommen. Sie werden selbst sehen, daß sich längere Zeilen beim Auflisten durch das kurze Aufblitzen verraten.

Vielleicht werden Sie jetzt bemerken, diese Schutzvarianten seien doch alle Schnee von gestern, bekannt, uninteressant.
Wir meinen jedoch, daß diese beiden Varianten als Grundprinzip recht nützlich sind.
Nachfolgend finden Sie weitere, in ihrer Art völlig anders konzipierte Schutzvarianten.

Die dritte Schutzvariante mutet zunächst vielleicht etwas

kompliziert an, ist es jedoch bei näherer Betrachtung nicht und erweist sich als recht hartnäckig gegenüber "Programm-Einbrechern".
Folgendes Programm wird hinter ein eigenes Programm gehängt:

```
62000 FOR A=PEEK(43)+256*PEEK(44)TOPEEK(45)+256*PEEK(46)-3
62010 IFPEEK(A)=58ANDPEEK(A+1)=58ANDPEEK(A+2)=58THENGOSUB
      62030
62020 NEXT A:END
62030 IFPEEK(A+3)=58ANDPEEK(A+4)=58THENPOKEA,0:A=A+4:RETURN
```

Die Zeilen, die geschützt werden sollen, werden mit fünf Doppelpunkten gekennzeichnet. Aus...

```
45 PRINT "BEISPIEL"
```

wird die gekennzeichnete Zeile...

```
45 :::::PRINT "BEISPIEL"
```

Auf diese Weise können beliebig viele Zeilen gekennzeichnet werden. Nachdem dies geschehen ist, wird das Schutzprogramm mittels RUN 62000 gestartet.
Je nach Länge Ihres zu schützenden Programmes dauert es nun einige Zeit, bis sich der blinkende Cursor zurückmeldet.
Probieren Sie deshalb diesen Listschutz zunächst an kleineren Programmen aus, damit Sie die Geschwindigkeit des Schutzvorganges kennenlernen.

Wenn Sie nun das Programm auflisten, so sind zwar die Zeilennummern der geschützten Zeilen, nicht aber deren Inhalt zu sehen.
Nun können die Zeilen 62000 - 62030 gelöscht und das gschützte Programm abgeSAVEt werden.

Das Prinzip des Schutzes:
Zunächst wird eine Schleife initialisiert, die das gesamte im Speicher befindliche BASIC-Programm durchläuft (43-44 = BASIC-Anfang, 45-46 = Ende Programm).
Nun wird nach den fünf Doppelpunkten (Code 58) gesucht. Sind sie gefunden, wird der erste Doppelpunkt durch den Code 0 ersetzt, die übrigen vier Doppelpunkte bleiben unverändert.
Wird das Programm jetzt von der LIST-Routine durchlaufen, findet diese den 0-Code. Das ist für sie das Zeilen-Ende-Kennzeichen: Der Listvorgang bricht ab.
Dennoch wird die Zeile ordnungsgemäß ausgeführt, denn die nachfolgenden vier Doppelpunkte werden als Koppelglieder interpretiert.

Vielleicht wird Sie die Geschwindigkeit dieses Schutzes leicht irritiert haben. Aus diesem Grunde finden Sie nachfolgend ein ähnliches Schutzprogramm, jedoch in Maschinensprache.
Gleich vorweg: Es schützt ein 13 KByte großes Programm in ca. einer Sekunde !

Hier der BASIC-Lader:

```
10 REM MASCHINENPROGRAMM ZUM SCHUETZEN BELIEBIGER BASIC-LIST
EN
20 REM BEGINN $8000 (DEZ.32768), LAENGE: 80 BYTES
30 REM GESCHUETZTE ZEILE MUSS MIT > UND 4 DOPPELPUNKTEN GEKE
NNZEICHNET WERDEN:
40 REM BEISPIEL: STATT '10 REM' LAUTET DIE ZEILE DANN: '10 >
::::REM'
50 REM START DES M-PRGS DURCH SYS8*4096
60 :
70 :
80 :
90 READA:REM EINLESEN DES M-PROGRAMMES
100 IFA=-1THENGOTO150:REM ENDE-KENNZEICHEN KONTROLLIEREN
110 CH=CH+A:REM BERECHNUNG DER CHECK-SUMME
120 POKE8*4096+I,A:REM SPEICHERN DER DATEN
130 I=I+1:GOTO70:REM NAECHSTEN WERT HOLEN
150 IFCH<>9839THENPRINT"FEHLER IN DATAS!":LIST200-
160 PRINT"■DATAS OK.  STARTEN DES PRGS. MITTELS■■■■■SYS 8*40
96 !":END
200 DATA162,120,169,8,160,0,132,34,133,35,177,34,201,177,240
,9,200,208
210 DATA247,230,35,202,208,242,96,142,80,128,141,81,128,140,
82,128,165,35
220 DATA133,37,165,34,133,36,162,4,200,240,28,177,36,201,58,
208,10,202
230 DATA208,244,172,82,128,169,0,145,34,174,80,128,173,81,12
8,172,82,128,76
240 DATA16,128,230,37,76,47,128,-1
```

Die Schutzroutine wird durch

SYS 8*4096

aufgerufen.
Die zu schützenden Zeilen werden nicht mehr mit fünf Doppelpunkten, sondern mit einem größer-als-Symbol und vier Doppelpunkten gekennzeichnet.

VERÄNDERN DES BASIC-LINK

Eine weitere Methode, Listen vor fremden Augen zu verstecken, ist die Veränderung des BASIC LINKs.
Doch zunächst die Frage, was ist der BASIC LINK ?
Um diese Frage beantworten zu können, ein Auszug aus dem BASIC-Speicher:

```
2048    0
2049   16 LINK low
2050    8 LINK high
2051   10 ZEILENNUMMER low
2052    0 ZEILENNUMMER high
2053  153 Beginn Inhalt der Zeile
(...)
2063    0 Zeilenende
2064   31 LINK low (vorheriger LINK zeigt auf diese Adresse)
2065    8 LINK high
2066   20 ZEILENNUMMER low
2067    0 ZEILENNUMMER high
2068  153 Beginn Inhalt der Zeile (153 = Code für PRINT)
```

In Adresse 2048 befindet sich eine 0 als Kennzeichen des BASIC-Anfanges. In den darauffolgenden beiden Adressen steht der LINK. Dieser LINK enthält im Low-High-Byte-Format die Adresse, ab der die nächste BASIC-Zeile zu finden ist.
Die nächsten zwei Bytes bilden die Zeilennummer der momentanen Zeile.
Es folgt der Zeileninhalt, der durch eine Null als Endezeichen beendet wird. Wieder folgt ein LINK, der die Adresse der nun folgenden Zeile enthält. Es folgt die Zeilennummer der Zeile, auf die der vorherige LINK zeigte.
Folgen keine BASIC-Zeilen mehr, so ist der letzte LINK 0,0.

Wichtig ist der LINK jedoch im Wesentlichen nur für die LIST-Routine. Will man nun eine Zeile verschwinden lassen, so genügt es, den LINK auf diese Zeile zu verändern: Man verbiegt ihn einfach auf die nächstfolgende Zeile.

Diese Aufgabe erledigt das nun folgende BASIC-Programm.

```
60000 POKE53280,1:POKE53281,1:POKE646,6:PRINT"■";
60001 PRINTTAB(10)"■ LINK-CHANGER "
60002 PRINTTAB(5)"■■■■DIESES PROGRAMM MANIPULIERT DEN":PRINT
60003 PRINTTAB(5)"ZEILEN-LINK. SIE WERDEN NACH DER":PRINT
60004 PRINTTAB(5)"ERSTEN UND DER ZWEITEN ZEILENNUM-":PRINT
60005 PRINTTAB(5)"MER GEFRAGT.":PRINT
60006 PRINTTAB(5)"SAEMTLICHE ZEILEN ZWISCHEN DIESEN":PRINT
60007 PRINTTAB(5)"INCL. DER ZWEITEN NUMMER VERSCHWIN-":PRINT
60008 PRINTTAB(5)"DEN! JE LAENGER ■IHR■ PROGRAMM,":PRINT
60009 PRINTTAB(5)"UMSO LAENGER BRAUCHT DER VORGANG!":PRINT:P
RINT
60010 PRINTTAB(10)"BITTE TASTE DRUECKEN !"
60011 GETA$: IFA$=""THEN 60011
60012 PRINT"■";
60015 INPUT"DIE ERSTE ZEILENNUMMER";A:INPUT"DIE ZWEITE ZEILE
NNUMMER";E
60019 PRINT"GEFUNDENE ZEILENNUMMER:"
60020 Q=PEEK(43)+256*PEEK(44)
60030 ZL=PEEK(Q+2)+256*PEEK(Q+3):IFZL=ATHENA1=Q:A2=Q+1:GOTO6
0060
60031 IF ZL>ATHENPRINT"■ZEILE EXISTIERT NICHT !":END
60035 PRINTTAB(23)"■";ZL
60040 Q=PEEK(Q)+256*PEEK(Q+1):IFQ>=PEEK(45)+256*PEEK(46)-3TH
ENEND
60050 GOTO60030
60060 Q=PEEK(43)+256*PEEK(44)
60070 IFPEEK(Q+2)+256*PEEK(Q+3)=ETHENPOKEA1,PEEK(Q):POKEA2,P
EEK(Q+1):END
60080 Q=PEEK(Q)+256*PEEK(Q+1):IFQ>=PEEK(45)+256*PEEK(46)-3TH
ENEND
60090 GOTO60070
```

ZEILEN LÖSCHEN? SYNTAX ERROR !

Eine weitere Methode bedient sich an Stelle des BASIC-LINKs der im BASIC-Speicher abgelegten Zeilennummern.
Diese Zeilennummern sind wie auch die LINKs im Low-High-Byte-Format abgelegt.

Wie Sie vielleicht wissen, ist es normalerweise nicht möglich, größere Zeilennummern als 63999 zu benutzen. Andernfalls kommt es zu einem SYNTAX ERROR.
Wie man dennoch Zeilennummern bis 65535 erzeugen kann, soll im Folgenden gezeigt werden. Weiterhin wird gezeigt, wie man in einer Programmliste mehrere Zeilen mit gleicher Zeilennummer oder in durcheinandergewürfelter Reihenfolge erzeugen kann.

Wie bereits erwähnt, ist auch die Zeilennummer durch Low- und High-Byte festgelegt. Daraus folgt, daß die kleinste mögliche Zeilennummer 0 (0+256*0), die größte hingegen den Wert 65535 (255+256*255) annehmen kann.
Bei der Abarbeitung eines Programmes ist es dem Computer weitestgehend gleich, welche Zeilennummern die Zeilen eines BASIC-Programmes besitzen. Er arbeitet die Zeilen in der Reihenfolge ab, in der sie im Speicher liegen.
Aus diesem Grund ist es ohne weiteres möglich, die Zeilennummern künstlich zu ändern, ohne den Programmablauf zu gefährden.
Es muß lediglich beachtet werden, daß die Befehle GOTO,GOSUB und THEN den veränderten Bedingungen angepaßt werden.

Alles schön und gut, werden Sie nun vielleicht einwenden, wie ermittelt man aber nun die Adresse des Low- und Highbytes der Zeilennummer der entsprechenden Zeile im Speicher ?
Das dauert doch ewig !

Das haben wir uns auch gedacht und folgendes Hilfsprogramm geschrieben. Mit seiner Hilfe wird es Ihnen keine Mühe machen, die Zeilennummern auch längerer Programme zu ändern.

Zunächst das Listing:

```
60000 Q=PEEK(43)+256*PEEK(44)
60001 PRINT"■":GOSUB60100
60002 Q=PEEK(Q)+256*PEEK(Q+1):IFQ>=PEEK(45)+256*PEEK(46)-3TH
ENEND
60003 GOTO60001
60010 IFPEEK(Q)=0THENGOSUB60100
60020 NEXTQ:END
60100 LO=PEEK(Q+2):HI=PEEK(Q+3)
60105 POKE199,1:PRINT"ZEILENNUMMER-CHANGER":PRINT
60110 PRINT"GEFUNDENE ZEILENNUMMER :";LO+256*HI
60120 PRINT:PRINT"[1] AENDERN"
60130 PRINT:PRINT"[2] WEITER"
60140 PRINT:PRINT"[3] ENDE"
60150 GETA$:IFA$=""THEN60150
60160 IFA$="2"THENRETURN
60170 IFA$="3"THENEND
60180 A$="0":PRINT:INPUT"NEUE ZEILENNUMMER (0-65535)";A$:A=V
AL(A$)
60190 HI=INT(A/256):LO=A-(256*HI)
60200 POKEQ+2,LO:POKEQ+3,HI:RETURN
```

PROGRAMMERKLÄRUNG:

In Zeile 60000 wird die erste LINK-Adresse bestimmt. Anschließend wird zur Routine ab Zeile 60020 verzweigt. Dort wird Low- und High-Byte der ersten Zeilennummer erfragt (LINK-Adresse +2 und +3). Die somit errechnete Zeilennummer wird als gefunden gemeldet (Zeile 60030). Nun kann gewählt werden zwischen 1) Ändern und 2) Weiter
Punkt 2) läßt die gefundene Zeilennummer unverändert und sucht die nächste. Bei der Wahl von Punkt 1) kann die gefundene Nummer manipuliert werden: Sie kann durch eine beliebige Zahl zwischen 0 und 65535 ersetzt werden. Die von Ihnen eingegebene Dezimalzahl wird in Low- und High-Byte-Format verwandelt (Zeile 60070), und als neues Low- bzw. High-Byte in den BASIC-Speicher gePOKEt.
Anschließend wird in Zeile 60010 der alte durch den neuen LINK ersetzt. Wieder wird nach Zeile 60020 verzweigt...
Sofern das Programm nicht von Ihnen gestoppt wird, hält es automatisch, sobald das Ende des im Speicher befindlichen Programmes erreicht ist (Zeile 60010).

Die von Ihnen manipulierten Zeilen haben gegenüber herkömmlichen Zeilen folgende Vorteile:

Ist die manipulierte Zeilennummer größer als 63999, kann die Zeile nicht mehr durch Überschreiben gelöscht werden.
Ist die manipulierte Zeilennummer kleiner als die

vorhergehende, kann die manipulierte Zeile ebenfalls nicht gelöscht werden. Dies ist nur dann möglich, wenn die manipulierte Zeile die erste Zeile des Programms ist.

Ist Ihr Programm nach Ihren Wünschen manipuliert worden, so können Sie das Hilfsprogramm löschen und Ihr Programm getrost abspeichern. Der Schutz wird mitgespeichert.

Es folgt nun noch eine weitere Schutzvariante, die zunächst nicht als solche erkennbar ist !

KÜNSTLICHES STEUERZEICHEN

Sicherlich werden Ihnen die "natürlichen" Steuerzeichen nicht unbekannt sein: Es sind dies die Cursor-Steuerzeichen, die man in so mancher PRINT-Anweisung findet, oder die Farb-Steuerzeichen, die durch gleichzeitiges Drücken der CTRL- und einer der Farbtasten erzeugt werden.
Ebenso gibt es auch sogenannte "künstliche" Steuerzeichen. Steuerzeichen also, die auf Tastendruck erzeugbar sind.

Wir beschäftigen uns an dieser Stelle nur mit einem künstlichen Steuerzeichen. Geben Sie einmal folgende Beispielzeile ein:

```
10 REM""
```

Fahren Sie mit dem Cursor auf das zweite Anführungszeichen. Drücken Sie nun bitte gleichzeitig die CTRL- und RVS ON(9)-Taste. Anschließend gleichzeitig SHIFT und M !
An Stelle des zweiten Anführungszeichens erscheint jetzt ein dunkles Kästchen mit einem hellen Querstrich. Dies ist das künstliche Steuerzeichen. Was es vermag, wird gleich deutlich: Drücken Sie nun gleichzeitig SHIFT und Q.!
Wieder erscheint ein Steuerzeichen: CURSOR UP. Schalten Sie nun den Revers-Mode ab, indem Sie gleichzeitig die CTRL- und

RVS OFF(O)-Taste drücken. Anschließend tippen Sie bitte " TESTZEILE" dahinter. Jetzt kann die RETURN-Taste betätigt werden.

Auf Ihrem Bildschirm müßte nun folgendes zu sehen sein:

```
10 REM"■TESTZEILE
```

Listen Sie die Zeile auf.
Sie ist nicht mehr sichtbar, lediglich noch das Wort TESTZEILE! Das 1. Steuerzeichen führt einen Wagenrücklauf (CR) durch . Außerdem sorgt es dafür, daß nachfolgende, normale Steuerzeichen hinter der REM-Anweisung ausgeführt werden.
Es folgt das Steuerzeichen Cursor-UP. Anschließend wird der Text TESTZEILE ausgegeben, der den ersten Teil der Zeile, 10 REM", überschreibt.

Wie läßt sich damit nun ein Listschutz realisieren ?
Beispielsweise so:

```
100 REM"■90 SYS 4096
```

Scheinbar wird in Zeile 90 ein Maschinenprogramm aktiviert. Werden nun alle Zeilen nach Zeile 100 durch eine der vorangegangenen Methoden unsichtbar gemacht, hält ein Außenstehender das Programm für ein Maschinenprogramm und sucht gar nicht erst nach einem versteckten BASIC-Listing.

Oder so:

```
100 PRINT"GESCHUETZT!":REM"■
```

Die kritische Zeile wird zwar aufgelistet, vom nachfolgenden Listing allerdings sofort wieder überschrieben.

Probieren Sie selbst einmal ein wenig mit diesem künstlichen Steuerzeichen herum. Ihnen werden sicherlich noch eine Menge weiterer Möglichkeiten einfallen !

Bisher wurde die LIST-Routine nur mehr oder weniger geschickt ausgetrickst. Beim nun folgenden Listschutz wird die gesamte LIST-Routine beeinflußt.

SCHUTZ DURCH POKES

Eine weitere Listschutzvariante verwendet die Adressen 774 und 775 ($0306-$0307). Diese beiden Adressen bilden den sogenannten LIST-VEKTOR, der normalerweise auf die Adresse $A71A zeigt. Diese Adresse liegt im BASIC-Interpreter und ist die Anfangsadresse der Routine zur Umwandlung von BasicToken in verständlichen Klartext.
Diese Routine wird von der LIST-Funktion benutzt. Durch Ändern des Listvektors auf eine beliebige andere Routine läßt sich die LIST-Funktion manipulieren.
Eine kleine Kostprobe:

```
POKE 774,226: POKE 775,252
```

Geben Sie diese beiden POKEs einmal ein. Wird jetzt der Befehl LIST verwendet, begibt sich der Rechner in den Einschaltzustand.
Der Grund ist klar: Der LIST-Vektor zeigt, durch die beiden POKEs verbogen, auf Adresse $EA31, die RESET-Routine.
Diese wird jetzt an Stelle der ursprünglichen LIST-Routine aufgerufen. Ebenso bietet es sich an, den Vektor auf eigene Maschinenroutinen zu biegen, die dann durch LIST aufgerufen werden.

Der offensichtliche Nachteil dieser Variante liegt allerdings darin, daß dieser Listschutz erst nach Starten des Programmes wirksam wird.

Bei gleichzeitiger Verwendung eines Autostartes ist diese Variante allerdings sehr interessant.

BLOCKIEREN "GEFÄHRLICHER" TASTEN

Es gibt so manche "gefährliche" Taste an Ihrem Computer. Gefährlich für Programm-Autoren beispielsweise, di verhindern möchten, daß ihr im Ablauf befindliches Programm gestoppt wird.
Da wäre also zunächst die RUNSTOP-Taste, die es gilt, außer Gefecht zu setzen. Es ist nicht weiter schwierig, den Interrupt-Vektor, der normalerweise auf die Adresse $EA31 zeigt, um drei Bytes nach oben zu schieben. So wird die STOP-Tasten-Abfrage einfach überschlagen.
Dies geschieht mit:

```
POKE 788,PEEK(788)+3     oder ganz einfach:  POKE 788,52
```

Wieder in den Normalmode gelangt man mit:

```
POKE 788,PEEK(788)-3     oder aber:          POKE 788,49
```

Mit der STOP-Taste läßt sich aber im ausgeschalteten Zustand noch weit mehr anfangen: Sie läßt sich jetzt wie eine normale Taste von BASIC aus abfragen:

```
100 GET A$:IF A$="" THEN 100
110 IF A$=CHR$(3) THEN PRINT"STOP-TASTE GEDRUECKT!"
```

So kehrt beispielsweise ein Programm bei Betätigung der STOP-Taste ins Menü zurück, etc.

Spätestens jetzt wird die RESTORE-Taste unangenehm auffallen. Mit ihrer Hilfe läßt sich noch immer ein Programm unterbrechen. Doch auch ihr kann man den Garaus machen!
Verantwortlich für diese Aktion ist der NMI-Vektor. Dieser zeigt normalerweise auf Adresse $FE47. Hier macht man

einfach Nägel mit Köpfen und überspringt die gesamte NMI-Routine, indem man den Vektor nach Adresse $FEC1 verbiegt (RTS).

Dies erreicht man durch:

POKE 792,193 NMI aus

Wieder zurück in den Normalmode gelangt man mit:

POKE 792,71 NMI ein

Möchte man ohnehin beide Tasten, RUN STOP und RESTORE, ausschalten, kann man sich komfortablerweise auch gleich dem sogenannten STOP-Vektor zuwenden.
Dieser zeigt im Normalfall auf Adresse $F6ED. Auch hier ist es lohnenswert, den Vektor zu verbiegen, zumal man gleich zwei "Werner" auf einen Streich erledigt hat. Wieder erledigt ein POKE das Verbiegen:

POKE 808,254

Dieser POKE setzt nicht nur gleichzeitig RUNSTOP und RESTORE außer Gefecht, außerdem wird auch das Listing je nach Länge mehr oder weniger stark verfremdet, was den Programmablauf an sich jedoch im Normalfall nicht beeinträchtigt.
Zurück in den Normalmode gelangt man wieder mit:

POKE 808,237

Zwar kann ein Programm auch nach Benutzung der beschriebenen POKEs unterbrochen werden, und zwar durch einen von außen zugeführten Hardware-Reset. Erstens ist nach diesem Reset das BASIC-Programm nicht mehr ohne weiteres listbar.
Zweitens kann auch dieser Hardware-Reset softwaremäßig unterbunden werden. Wie dies gemacht wird, ist im Kapitel

über Reset und Interrupts nachschlagbar.

Es gibt jedoch noch weitere "gefährliche" Tasten, mit denen das Programm zwar nicht gestoppt werden kann, die aber dennoch störend wirken können:
Durch gleichzeitiges Drücken von SHIFT- und C=-Taste schaltet der Rechner den Zeichensatz um. Dies kann verhindert werden durch

```
POKE 657,128
```

und wird durch

```
POKE 657,0
```

wieder rückgängig gemacht.

Unser Repertoire an "abschaltwürdigen" Tasten ist nun erschöpft. Sie finden an dieser Stelle nun eine kleine Auflistung weiterer interessanter POKEs. Viel Spaß damit !

Effekt	POKE-Kommando
STOP ausschalten	POKE 788,52:POKE 808,239
STOP wieder einschalten	POKE 788,49:POKE 808,237
STOP, RESTORE und LIST ausschalten	POKE 808,234
STOP, RESTORE und LIST ausschalten	POKE 808,225
STOP, RESTORE und LIST wieder ein	POKE 808,237
RESTORE ausschalten	POKE 793,203
RESTORE ausschalten	POKE 792,193
SAVE verhindern	POKE 818,32:POKE 819,245
SAVE wieder einschalten	POKE 818,237:POKE819,245

LIST verhindern	POKE 775,200
LIST erlauben	POKE 775,167
TASTATUR ausschalten	POKE 649,0
TASTATUR wieder einschalten	POKE 649,10

****

VORTÄUSCHEN EINES MASCHINENSPRACHE-SPIELS

Es gibt viele Versuche, ein BASIC-Programm wirklich sicher vor dem Listen und Kopieren zu schützen. Aber es ist inzwischen so, daß jeder, der ein bißchen Fachwissen besitzt, und der mit einem Maschinensprache-Monitor umzugehen weiß, einen List- oder Kopier-Schutz umgeht.
Viele von diesen Leuten jedoch lassen die Finger von Maschinensprache-Programmen, weil Sie glauben, in Maschinensprache-Programmen ohnehin nichts ändern zu können, weil sie Maschinensprache nicht verstehen.
Außerdem steigt häufig der Wert eines Programmes, wenn der oder die Benutzer an ein Maschinensprache-Spiel glauben.
Diese Routine soll also ein Maschinensprache-Spiel vortäuschen, obwohl es sich im Grunde "nur" um ein BASIC-Spiel handelt.
So ein Programm kennzeichnet sich für den Benutzer dadurch, daß es nur eine Zeile in BASIC besitzt, in der der Sprungbefehl zum Maschinenprogramm zu finden ist: ein SYS-Befehl. Aber wie erreichen wir es, daß nur eine Zeile mit einem SYS-Befehl erscheint?
Ganz einfach. Zuerst geben wir den SYS-Befehl ein. Sie können nach den Beispielen unten verfahren. Lassen Sie entweder den Text ganz weg, oder verändern Sie ihn nach Ihren Vorstellungen. Die Zeilennummern können Sie ebenfalls frei wählen.
Hier die Beispiele:

```
1984 SYS (2110) BY MEDLAY SPARROW
  10 SYS 2110
```

Wenn Sie sich für eine Form entschieden haben, schreiben Sie sie als BASIC-Zeile. Jetzt müssen Sie einige Zeilen eingeben:

```
POKE 43,80 : POKE 2127,0 : NEW
10 FOR I = 2110 TO 2126
20 READ A
30 POKE I,A
40 NEXT I
50 DATA 169,80,133,43,169,52,141,20,3
60 DATA 169,193,141,24,3,76,113,168
```

Wir haben die Kontrolle der Daten weggelassen, weil es nur sehr wenige sind. Achten Sie deshalb um so genauer auf die Richtigkeit der DATA-Zeilen.
Jetzt ganz kurz ein dokumentiertes Maschinensprache-Listing dieser kurzen Routine :

```
#$80 :neuer Basic-Anfang
0840 STA $2B  :setzen
0842 LDA #$34 :setzt RUN-STOP aus
0844 STA $0314
0847 LDA #$C1 :setzt RESTORE aus
0849 STA $0318
084C JMP $A871:Aufruf der RUN-Routine
```

Starten Sie das Programm mit "RUN" und löschen Sie es anschließend wieder. Danach können Sie Ihr eigene Programm, das geschützt werden soll, in den Computer hineinladen.
Wichtig ist, daß das Programm in Zeile 0 anfängt. Tippen Sie nun

```
POKE 43,1
```

ein und speichern Sie dann das Spiel ganz normal ab.
Das Programm wird auch wieder ganz normal hereingeladen,

aber bei LIST erscheint eben nur die eine SYS-Zeile. Das Programm wird wie üblich mit RUN gestartet. Jetzt springt der Computer zur kleinen Routine, die den Speicher heraufsetzt und außerdem RUN-STOP-RESTORE aussetzt. Das BASIC-Programm wird gestartet. Wenn Sie es einmal gestartet haben, läßt es sich nicht mehr ohne weiteres unterbrechen.

Ein 100%iger Schutz ist diese Routine natürlich auch nicht, aber zusammen mit anderen hier vorgestellten Methoden wird es unerlaubten Benutzern schwer gemacht, in Ihre Programme einzudringen.

4. BEFEHLS-ERWEITERUNG SELBST GEMACHT !

Um eigene Befehle ins Betriebssystem einzufügen, kann man mehrere Möglichkeiten nutzen.
Drei davon wollen wir hier vorstellen:

a) Verändern des BASIC-CODE-LINKS

b) Verändern der CHRGET-Routine

c) Verändern der IRQ-Routine

A) ÄNDERN DES BASIC-CODE-LINKS

Das ist die effektivste Methode, um mehrere Befehle einzufügen. Wir nutzen dabei aus, daß es einen Vektor gibt, der auf die Routine zeigt, die die eingegebenen BASIC-Befehle auswertet.
Dieser Vektor hat die Adresse $0308-$0309 (776-777). Diesen Vektor ändern wir einfach auf ein von uns erstelltes Maschinen-Programm. Doch wie sieht dieses Programm aus?
Nehmen wir an, wir haben es in den Kassetten-Puffer ($033C-$03FB) gelegt:

```
 033C   LDA #$47
 033E   STA $0308
 0341   LDA #$03
 0343   STA #0309
 0346   RTS
------------------
 0347   JSR $0073
 034A   CMP #$5F
 034C   BEQ 0351
 034E   JMP $A7E7
 0351   LDA #$05
```

```
0353   STA $0286
0356   JMP $A7E4
```

Das Programm läßt sich in zwei Teile gliedern:

TEIL 1 ($033C-$0346) legt den BASIC-Code-Link auf TEIL 2.
TEIL 2 ($0347-$0356) stellt die eigentliche Befehls-Erweiterung dar:

JSR $0073 : Der Computer holt sich das nächste Zeichen, das eingegeben worden ist ($0073=CHRGET-Routine)

CMP #$5F : Dieses Zeichen wird mit dem ASCII-Wert für "(Pfeil nach links)" verglichen

JMP $A7E7 : Da die Zeichen nicht gleich sind, springt er zur normalen Adresse, die den Vektor angibt. Dort wird das eingegebene Zeichen daraufhin untersucht, ob es einen anderen normalen BASIC-Befehl darstellt.

LDA #$05:STA $0286 : Das ist nun die eigentliche Befehlserweiterung. In diesem Fall ist es nichts Weltbewegendes:
Die Zeichenfarbe wird einfach grün gesetzt.

JMP $A7E4 : Der Computer kehrt wieder in den normalen Betriebs-Modus zurück

Probieren wir es aus:
Das obige Maschinen-Programm als BASIC-Lader sieht so aus:

```
10 FOR X= 828 TO 856: READ A: POKE X,A: NEXT X
20 DATA 169,71,141,8,3,169,3,141,9,3,96
30 DATA 32,115,0,201,95,240,3,76,231,167,169,5,141,134,2,76
      ,228,167
RUN

SYS 828
```

Wenn Sie jetzt die "Pfeil links"-Taste drücken, (RETURN muß gedrückt werden), wird Ihre Zeichenfarbe grün.
Sie sehen, es ist gar nicht schwierig. Doch die Art, in der das Programm geschrieben ist, hat einen entscheidenden Nachteil:
Bei mehreren erweiterten Befehlen wird die Zeit, die der Computer zum Nachschauen braucht, sehr lang.
Um das zu vermeiden, kann man sich folgendermaßen behelfen:
Allen neu definierten BASIC-Befehlen muß ein einheitliches Erkennungszeichen vorangestellt werden, z.B. den "Pfeil nach links" oder das Ausrufezeichen.
In einem Programm zur Befehls-Erweiterung muß also zuerst das Erkennungszeichen abgefragt werden. Wenn dieses nicht gefunden wird, springt der Computer zur normalen Tastaturdekodierung. Andernfalls prüft er auf einen neuen Befehl.

Zum besseren Verständnis ein Beispielprogramm:

```
033C  LDA #$47
033E  STA $0308
0341  LDA #$03
0343  STA #0309
0346  RTS
0347  JSR $0073
034A  CMP #$21    :ASCII-Wert für "!"
034C  BEQ 0351    :Neuer Befehl vorhanden
034E  JMP $A7E7   :Kein neuer Befehl
0351  JSR $0073   :Liest Zeichen nach Ausrufe-Zeichen
0354  CMP #$91    :ON? ($91=Token für ON)
0356  BEQ 035F    :Ja
0358  CMP #$A8    :NOT? ($A8=Token für NOT)
035A  BEQ 0374    :Ja
035C  JMP $AF08   :Kein gültiger Befehl, also SYNTAX-ERROR
035F  SEI         :Befehl ON
```

```
0360  LDA #$6E   :Legt IRQ auf $036E
0362  STA $0314
0365  LDA #$03
0367  STA $0315
036A  CLI
036B  JMP $A7E4  :Befehl ON abgeschlossen
036E  INC $0286  :Neuer IRQ
0371  JMP $EA31  :Alter IRQ weiter
0374  SEI        :Befehl NOT
0375  LDA #$31   :IRQ wieder normal
0377  STA $0314
037A  LDA #$EA
037D  STA $0315
0380  CLI
0381  JMP $A7E4  :Befehl NOT abgeschlossen
```

Als BASIC-Lader:

```
10 FOR X= 828 TO 898: READ A: POKE X,A: NEXTX
20 DATA 169,71,141,8,3,169,3,141,9,3,96
30 DATA 32,115,0,201,33,240,3,76,231,167,32,115,0,201,145,
   240,7,201,168
40 DATA 240,24,76,8,175,120,169,110,141,20,3,169,3,141,21,3
   ,88,76,228,167
50 DATA 238,134,2,76,49,234
60 DATA 120,169,49,141,20,3,169,234,141,21,3,88,76,228,167
```

Nach RUN und SYS 828 stehen Ihnen zwei neue Befehle zur Verfügung: !ON stellt den IRQ-Vektor auf die Adresse $036E um. Nun wird jede 1/60 SEk. die Zeichenfarbe erhöht.
Abstellen läßt sich dieser Effekt mit dem zweiten neuen Befehl: !NOT.
Nun sehen Sie auch noch einen weiteren Vorteil des Erkennungszeichens:
Es können alte Befehle mit neuen Funktionen belegt werden, ohne daß die ursprünglichen Befehle ihre Funktion verlieren.

Nun noch einige Tips:
Am elegantesten wäre es, die Befehlswörter in einer Tabelle unterzubringen, und dann der Reihe nach mit dem eingegebenen Befehl zu vergleichen (so ähnlich arbeitet der BASIC-Interpreter). Interessant ist auch die Möglichkeit, neue Befehle in Tokens umzuwandeln. Dadurch läßt sich Speicherplatz sparen.
Der zuständige Vektor heißt $0304-$0305 (772-773). Frei sind noch die Tokens 202-254.

B) ÄNDERN DER CHRGET-ROUTINE

Nun die zweite Möglichkeit der Befehls-Erweiterung:
Im Bereich $0073-$0084 liegt die sogenannte CHRGET-Routine. Sie sieht folgendermaßen aus:

```
0073  INC $007A
0075  BNE $0079
0077  INC $007B
0079  LDA $HHLL  :Holt Zeichen aus dem BASIC-Text
007C  CMP #$3A
007E  BCS $008A
0080  CMP #$20
0082  BEQ $0073
0084  SEC
0085  SBC #$30
0087  SEC
0088  SBC #$D0
008A  RTS
```

Da diese Routine im RAM liegt (sie wird beim Einschalten des Computers vom ROM ins RAM kopiert), können wir sie nach Belieben ändern. Der Vektor $0073-$007B kann so bleiben.
$007C muß den Befehl JMP $(eigenes Programm) enthalten.

In unserem Programm wird geprüft, ob das Zeichen, das gerade gelesen wird, ein neuer Befehl ist. Wenn nicht, so muß die normale CHRGET-Routine ausgeführt werden.
Als Beispiel geben wir dem Befehl NEW eine neue Bedeutung: Er führt einen RESET aus.
Die neue Routine liegt wieder in dem Kassetten-Puffer.

```
033C  LDA #$4C
033E  STA $007C
0340  LDA #$49
0342  STA $007D
0344  LDA #$03
0346  STA $0076
0348  RTS
0349  CMP #$A2  :NEW?
034B  BEQ $035F :Ja
034D  CMP #$3A  :Nein, aber CHRGET fortsetzen
034F  BCS $035E
0351  CMP #$20
0353  BNE $0358
0355  JMP $0073
0358  SEC
0359  SBC #$30
035B  SEC
035C  SBC #$D0
035E  RTS
035F  JMP $FCE2
```

Im ersten Teil wird einfach die CHRGET-Routine verändert. Der zweite Teil wird bei jedem Ansprung der CHRGET-Routine durchlaufen.

Doch hier zunächst der BASIC-Lader:

```
10 FOR X= 828 TO 865: READ A: POKE X,A: NEXT X
```

```
20 DATA 169,76,133,124,169,73,133,125,169,3,133,126,96
30 DATA 201,162,240,18,201,58,176,13,201,32,208,3,76,114,0
40 DATA 56,233,48,56,233,208,96,76,225,252
```

Wenn Sie nun NEW eingeben, so führt der Computer einen RESET aus. Wie Sie sehen, ist auch diese Möglichkeit, neue Befehle zu implementieren, nicht schwierig.

Nun zur letzten Möglicheit:

C) ÄNDERN DES IRQ-VEKTORS:

Auch diese Möglichkeit ist denkbar, wenn auch nicht so oft gebraucht.
Wie Sie vielleicht wissen, ist die Interrupt-Routine (IRQ) eine Routine, die jede 1/60 SEK. angesprungen wird.
Das können Sie ausnutzen:

```
033C  SEI
033D  LDA #$49
033F  STA $0314
0342  LDA #$03
0344  STA $0315
0347  CLI
0348  RTS
0349  LDA $CB
034B  CMP #$39
034D  BEQ $0352
034F  JMP $EA31
0352  JSR $E544
0355  JMP $EA31
```

Als BASIC-Lader:

```
10 FOR X= 828 TO 855: READ A: POKE X,A: NEXTX
20 DATA 120,169,73,141,20,3,169,3,141,21,3,88,96
30 DATA 165,203,201,57,240,3,76,49,234,32,68,229,76,49,234
```

Wenn Sie nun RUN und SYS 828 eingeben, sieht alles zunächst ganz normal aus. Sollten Sie jedoch ^ drücken, so wird der Bildschirm gelöscht.
Vorsicht! Es kann passieren, daß der Cursor nach dem Benutzen dieser Taste nicht mehr sichtbar ist - do not panic, wenn Sie etwas schreiben, wird er wieder sichtbar.
Zur Erklärung:
Der IRQ-Vektor wird auf die eigene Routine verbogen. Dort wird geprüft, ob eine Taste gedrückt ist (in unserem Fall wird auf die ^-Taste geprüft). Wenn dies der Fall ist, so wird der Bildschirm gelöscht.
In jedem Fall muß aber zur ursprünglichen IRQ-Routine zurückgesprungen werden.
Wie Sie vielleicht bemerkt haben, besteht zwischen den ersten beiden Möglichkeiten und dieser letzten ein großer Unterschied:
Die ersten beiden nehmen den Befehl erst nach Drücken der RETURN-Taste an. Bei dieser Möglichkeit reagiert der Computer sofort.

Nachdem Sie jetzt verschiedene Methoden kennengelernt haben, eigene Befehle in den Interpreter einzubinden, müssen Sie selbst entscheiden, welche für Ihre Zwecke die vorteilhafteste ist. Gut gebrauchen lassen sich alle drei.

5. GRAFIK

GRUNDLAGEN

Jeder, der früher oder später einmal mehr als die über die Tastatur erreichbare "Grafik" benötigt, um ansprechende Spiele oder Sonderzeichen zu erhalten, wird sich mit dem Character-Generator des C-64 näher auseinandersetzen müssen.
Was ist eigentlich der Character (= Zeichen)-Generator ?
Sämtliche Zeichen, die durch Tastendruck auf dem Bildschirm ausgegeben werden können, müssen in ihrem Aussehen fest definiert sein.
Der Speicherbereich, in dem dieses abgespeichert ist, nennt man Character-Generator. Er liegt von $D000 - $DFFF im ROM.
Es bleibt nun die Frage offen, wie das Zeichen in diesem Bereich gespeichert wird. Um diese Frage zu klären, sieht man sich zweckmäßigerweise einmal die vergrößerte Matrix eines beliebigen Bildschirmzeichens an. Beispielsweise das "A":

```
 76543210      Bit-Nummern
1...**...1
2..****..2
3.**..**.3
4.******.4
5.**..**.5
6.**..**.6
7.**..**.7
8........8
 76543210
```

Wie Sie erkennen können, besteht das Zeichen aus einer Matrix von 8*8 Feldern. Es können somit maximal 64 Punkte gesetzt werden.

Um das Äußere des Zeichens nun zu digitalisieren, faßt man jeweils eine Zeile des Zeichens in einem Byte zusammen. Jedem möglichen Punkt der Zeile ist nun jeweils ein Bit des Bytes zugeordnet.
Auch dies soll an einem Beispiel erklärt werden. Dazu nehmen wir als Beispiel die dritte Zeile des Buchstabens "A". Diese sieht folgendermaßen aus:

```
.**..**.
```

Um nun den Wert des Bytes dieser Zeile errechnen zu können, werden die Bits der gesetzten Punkte addiert, während nicht gesetzte Punkte unbeachtet bleiben:

```
2↑7 2↑6 2↑5 2↑4 2↑3 2↑2 2↑1 2↑0   Werte der Bits
128 64  32  16  8   4   2   1             "

 .   *   *   .   .   *   *   .   *(gesetzter Punkt)

 0 +64 +32 + 0 + 0 + 4 + 2 + 0  =  102  (Byte-Wert)
```

Sollte die Art und Weise der Digitalisierung eines Zeichens, in diesem Fall einer Zeile eines Zeichens, unklar geblieben sein, so kann das folgende kleine Trainingsprogramm zum besseren Verständnis beitragen. Zunächst das Listing:

Benutzte Variablen:

```
J     =     enthält den Wert des aktuellen Bits (2^A)
EG$   =     enthält die Darstellung der errechneten Zeile
BY    =     enthält den Byte-Wert
```

```
10 REM TRAINIGSPROGRAMM CHARACTERGENERATOR
20 PRINTCHR$(147):POKE53280,1:POKE53281,1:POKE646,6:REM COLO
UR
30 CLR:PRINT"■ BYTE - BIT - TRAININGSPROGRAMM          ■":PRI
NT
40 PRINT"[1] ZEILE EINGEBEN":PRINT
50 PRINT"[2] BYTE EINGEBEN":PRINT:PRINT
60 PRINT"BITTE WAEHLEN SIE !"
70 GETA$:IFA$=""THEN70
80 IFA$="1"THEN200
90 REM *** BYTE EINGEBEN ***
100 PRINTCHR$(147)
110 INPUT"DER WERT DES BYTES (0-255)";BY:PRINT:IFBY<0ORBY>25
5THEN110
120 FOR A=7TO0STEP-1
130 J=2↑A
140 IFJ<=BYTHENPRINT"";:BY=BY-J:GOTO160
150 PRINT"■";
160 NEXT
170 PRINT"■■TASTE DRUECKEN !"
180 GETA$:IFA$=""THEN180
190 GOTO20
200 REM *** ZEILE EINGEBEN ***
210 PRINTCHR$(147)
220 PRINT"X=PUNKT GESETZT   M=PUNKT NICHT GESETZT":PRINT:PRI
NT
230 FOR A=7TO0STEP-1
240 PRINT8-A"PUNKT ?";
241 GETA$:IFA$<>"M"THENIFA$<>"X"THEN241
250 PRINTA$:IFA$="X"THENBY=BY+2↑A:EG$=EG$+"":GOTO270
260 EG$=EG$+"■"
270 NEXTA
280 PRINT:PRINTEG$"="BY
290 PRINT:PRINT"BITTE TASTE DRUECKEN !"
300 GETA$:IFA$=""THEN300
310 GOTO20
```

Nach dem Starten des Programms können Sie zwischen zwei Modi wählen:

- Zeile eingeben
- Byte eingeben

Wenn Sie Nummer 1 (Zeile eingeben) wählen, können Sie nun die acht Punkte einer Zeichenzeile setzen. Sie werden nach den acht Punkten gefragt. Ein X setzt den entsprechenden Punkt, ein anderer Buchstabe löscht ihn.
Sind alle acht Punkte definiert worden, wird die definierte Bildschirmzeile ausgedruckt und der Byte-Wert der Zeile errechnet.

Nummer 2 (Byte eingeben) bewirkt den umgekehrten Vorgang. Sie geben den Byte-Wert ein, der zwischen 0 und 255 liegen muß, und der Computer druckt die zugehörige Zeile aus.

DER CHARACTER-GENERATOR IM SPEICHER

Die Lage des Character-Generators im Speicher (ROM) ist aus folgendem Diagramm ersichtlich:

```
                $D000 (53248)
1a                      Groß/Grafik normal
                $D400 (54272)  Zeichensatz 1
1b                      Groß/Grafik revers
                $D800 (55296)
2a                      Klein/Groß normal
                $DC00 (56320)  Zeichensatz 2
2b                      Klein/Groß revers
                $DFFF (57343)
```

Der Zeichengenerator ist also in zwei Blocks unterteilt, die wiederum in zwei kleinere Blocks unterteilt sind.
Block 1 ist der erste Zeichensatz des C-64: Großbuchstaben und Blockgrafik.
Block 2 ist dementsprechend der zweite Zeichensatz mit großen und kleinen Buchstaben.
Jeder dieser Blöcke beinhaltet seinen Zeichensatz wiederum zweimal: Einmal in normaler und einmal in reverser Darstellung.
Daraus läßt sich auch die Größe des Zeichengenerators ableiten. Jedes Zeichen benötigt zur Abspeicherung 8 Bytes (pro Zeile 1 Byte). Pro Block gibt es 256 Zeichen: 128 normale und 128 reverse Zeichen.

Diese lassen sich zur Demonstration kurz auf den Bildschirm bringen:

```
10 FOR A=0 TO 255: POKE 1024+A,A: POKE 55296+A,1: NEXT
```

Da der zweite Zeichensatz ebenfalls 256 Zeichen enthält, kommt man insgesamt auf 512 verschiedene Zeichen. Es werden also 512*8 = 4 KByte benötigt.

AUSLESEN DES ZEICHENSATZES

Kommen wir zu einem weiteren Problem. Der Character-Generator liegt im ROM. Er ist daher nicht ohne weiteres veränderbar, sondern muß in einen anderen Speicherbereich, der mit RAM ausgestattet ist, kopiert werden.
Der Kopiervorgang an sich dürfte recht einfach mit einer kleinen Schleife zu erledigen sein:

```
10 DIM B(4095)
```

```
20 FOR A=0 TO 4095: REM $D000 - $DFFF
30 B(A) =PEEK(53248+A)
40 NEXT A: REM AUSLESEVORGANG BEENDET
50 FOR A=0 TO 4095
60 PRINT B(A);: NEXT A
70 END
```

Das Programm benötigt ca. eine Minute zum Auslesen des Speichers. Anschließend werden eine ganze Menge Zahlen ausgegeben.
Es läßt sich bereits auf den ersten Blick erkennen, daß es sich hierbei nicht um den Inhalt des Character-Generators handeln kann: Es dominieren Zahlen über 240 sowie die Null.
Doch der Fehler ist schnell gefunden: Der Speicherbereich von $D000 bis $DFFF ist doppelt belegt. Neben dem Character-ROM ist dort noch RAM-Speicher anzutreffen, der sich in demselben Adreßbereich befindet wie das ROM.
In diesem RAM sind die VICs, die beispielsweise die Sprites steuern, das SID (Sound Interface Device) und de Farbspeicher untergebracht.
Unser Lesezugriff hat also nicht das ROM mit den für uns wichtigen Informationen ausgelesen, sondern das (uninteressante) RAM.
Um an das begehrte ROM heranzukommen, bedarf es einer Information an den Computer. Daraufhin wird der Zugriff auf das Character-ROM ermöglicht. Gleichzeitig aber kann der Computer nicht mehr auf das RAM zurückgreifen.
Man erreicht den Zugriff durch Löschen des 2. Bits der Adresse 1 (Prozessorport). Gleichzeitig muß der Interrupt verhindert werden (SEI). In BASIC sieht das so aus (folgende Zeilen sind dem vorangegangenen Programm hinzuzufügen):

```
5  POKE 56334,0: POKE 1,51: REM INTERRUPT OFF,ZUGRIFF MOEGL.

45 POKE 1,55: POKE 56334,1: REM INTERRUPT ON, CHAR-ROM OFF
```

Zwei Dinge sind anzumerken, bevor Sie das so veränderte Programm starten: Mit Hilfe der Adresse 56334 wird der Interrupt verhindert. Bedingt durch diesen Eingriff kann das einmal gestartete Programm nicht mehr durch RUNSTOP & RESTORE vorzeitig abgebrochen werden.
Geben Sie nie POKE 1,51 ein, ohne zuvor den Interrupt verhindert zu haben. Es kommt sonst zu einem Absturz des Rechners!

Starten Sie das veränderte Programm. Nach ca. einer Minute werden wieder unzählige Werte ausgegeben. Diesmal handelt es sich aber um den wirklichen Inhalt des Character-Generators.

Vielleicht wissen Sie nun nichts mit diesen Zahlen anzufangen. Deshalb wollen wir sie zunächst umformen. Dazu verwandeln wir sie mit folgendem Zusatzprogramm in sichtbare Zeichen.
Die nun folgenden Zeilen müssen an das vorangegangene Listing angehängt werden. Die Zeilen 50-70 des alten Listings werden dabei überschrieben.

Benutzte Variablen:

B(0)-B(4095)	=	Inhalt des Zeichengenerators
A	=	Flag: 0=Punkt gesetzt
X	=	Schleife, holt Punkte einer Zeile
2^X	=	derzeitiger Bit-Wert
Z	=	derzeitiger Character-Wert

```
50 REM *** ZAHLEN IN ZEICHEN WANDELN ***
55 Z=B(I)
60 FOR X=7 TO 0 STEP-1
70 A=Z AND 2↑X
80 IF A THEN PRINT ".";: GOTO 100: REM A=0, GESETZTER PUNKT
```

```
90 PRINT " ";:REM KEIN GESETZTER PUNKT
100 NEXT X: PRINT
110 I=I+1: IF I=4096 THEN END
120 GOTO 50
```

Starten Sie das so erweiterte Programm. Wieder dauert es ca. eine Minute, bis der Speicher ausgelesen ist. Nun aber hat sich einiges verändert:
Aus nüchternen Zahlen sind vergrößerte Zeichen geworden, die auf den Bildschirm ausgegeben werden.

KOPIEREN DES CHARACTER-GENERATORS

Wie bereits erwähnt liegt der originale Character-Generator im ROM. Um ihn ändern zu können, muß er an eine andere mit RAM ausgestattete Stelle im Speicher kopiert werden. Dies ist nun keine Schwierigkeit mehr, da ja bekannt ist, wie man den Generator ausliest.
Es bleibt nur zu überlegen, wohin der neue, veränderbare Zeichensatz am zweckmäßigsten kopiert wird. Hierfür bieten sich mehrere Speicherpositionen an:

a) im BASIC-Speicher

Die Kopie des Character Generators könnte an den Anfang des BASIC-Speichers gelegt werden. Es müßte lediglich der BASIC-Start entsprechend verschoben werden. Hierbei gehen dem BASIC-Programmierer allerdings 2 KByte Speicherplatz verloren.
Weiterhin bietet es sich an, ihn ans Ende des BASIC-Speichers zu legen. Das etwas umständliche

Heraufsetzen des BASIC-Starts entfällt.

b) Unter das ROM

Sofern das Betriebssystem nicht durch Eingriffe des Benutzers im RAM liegt, stehen hier jeweils 8 KByte ungenutztes RAM zur Verfügung. Da die PEEK-Anweisung jedoch auf das ROM zurückgreift, könnte der dort liegende, kopierte Zeichensatz nicht ausgelesen werden (was jedoch nicht unbedingt störend ist). Benutzen Sie die modifizierte PEEK-Anweisung aus Tips & Tricks I.!

Im Folgenden finden Sie zwei Kopierprogramme, die den Character-Generator einmal in den Anfang des BASIC-Speichers und zum anderen unter das ROM des Betriebssystems kopieren.

Kopierprogramm 1: Character-Generator-Kopie an BASIC-Anfang

```
POKE 44,16: POKE 16+256,0: NEW

10 POKE 56334,0: POKE 1,51: REM ZUGRIFF AUF CHAR-ROM ERMöGL.

20 FOR K=0 TO 2047
30 POKE 2048+K, PEEK(53248+K)
40 NEXT
50 POKE 1,55: POKE 56334,1
60 POKE 53272,PEEK(53272) AND NOT 12 OR 2:REM UMSCHALTEN
```

Beginn Character-Gen.: 2048 ($0800), Bildschirm-RAM: 1024

(Aus internen Gründen hat der Character-Generator am BASIC-Anfang lediglich 2 KByte zur Verfügung. Es kann daher nur einer der beiden Zeichensätze des Character-Generators untergebracht werden. In diesem Fall Groß/Grafik. Möchten Sie hingegen lieber den Klein/Großschrift-Zeichensatz, so ändern Sie Zeile 30 ab in:

```
30 POKE 2048+K, PEEK(55296+K)
```

Kopierprogramm 2: Character-Generator-Kopie unter ROM

Der BASIC-Start ist hierbei nicht relevant.

```
10 POKE 56334,0: POKE 1,51: REM ZUGRIFF AUF CHAR-ROM
20 FOR K=0 TO 4095
30 POKE 57344+K, PEEK(53248+K)
40 NEXT K
50 POKE 1,55: POKE 56334,1
60 POKE 56576,196: REM HOECHSTER 16-K-BLOCK
70 POKE 648,3*64+4:REM BILDSCHIRM-RAM NACH (3*64+4)*256
80 POKE 53272,PEEK(53272) AND NOT 6 OR 8: REM UMSCHALTEN
```

Beginn Character-Gen.: 57344 ($E000), BS-RAM: 50176

In den folgenden Beispielen verwenden wir den mit Kopierprogramm 2 kopierten Character-Generator. Es sind dabei folgende Dinge zu beachten:

- Statt 1024 lautet die Anfangsadresse des neuen Bildschirmspeichers 50176
- In den Normal-Mode gelangt man wieder mit:
  POKE 56576,199: POKE 648,4: POKE 53272,21
- RUNSTOP & RESTORE bewirken keine vollständige Rücksetzung in den Normal-Mode. Es muß zusätzlich "POKE 648,4" eingegeben werden.

Geben Sie einmal das zweite Kopierprogramm ein und starten Sie es. Nach ca. 60 Sekunden meldet sich der Cursor wieder. Wenn das Programm korrekt eingegeben wurde, dürfte der gewohnte Zeichensatz sichtbar sein.
Zur Demonstration soll nun ein Zeichen verändert werden. Das erste Zeichen im Character-Generator ist der Klammeraffe. Er

soll jetzt sein Äußeres verlieren und verändert werden.
Die ersten acht Bytes des Character-Generators werden durch den Klammeraffen belegt: 57344 bis 57351. Adresse 57344 beinhaltet dabei die oberste Zeile des Klammeraffen, 57351 hingegen die unterste.
Wir verändern den Klammeraffen nun oben ein wenig:

```
POKE 57344,255
```

Drücken Sie die Klammeraffen-Taste.! Der Klammeraffe ist oben merklich platter geworden. Um das Zeichen beispielsweise auch unten abzuplatten, genügt:

```
POKE 57351, 255
```

UMSCHALTEN DES CHARACTERGENERATORS

Wie Sie den beiden Kopierprogrammen entnehmen können, sind neben dem eigentlichen Kopiervorgang noch die Adressen 53272 und 56576 sowie 648 notwendig.
Was bewirken diese Adressen ?
Es muß dem Computer mitgeteilt werden, wo sich der neue Zeichensatz im Speicher befindet. Hierfür ist zunächst nur die Adresse 53272 zuständig.
Die Bits 1-3 regeln dabei die Verschiebung des Zeichensatzes. Bit 0 ist unbenutzt, die Bits 4-7 verschieben den Bildschirmspeicher.

Übersicht der möglichen Startadressen und der zugehörigen Bitkombinationen:

Bildschirmspeicher:			Zeichensatz:	
0000xxxx	0		xxxx000x	0
0001xxxx	1024	(normal)	xxxx001x	2048

0010xxxx	2048	xxxx010x	4096
0011xxxx	3072	xxxx011x	6144(normal)
0100xxxx	4096	xxxx100x	8192
0101xxxx	5120	xxxx101x	10240
0110xxxx	6144	xxxx110x	12288
0111xxxx	7168	xxxx111x	14336
1000xxxx	8192		
1001xxxx	9216		
1010xxxx	10240		
1011xxxx	11264		
1100xxxx	12288		
1101xxxx	13312		
1110xxxx	14336		
1111xxxx	15360		

Soll der Bildschirmspeicher verschoben werden,so muß gleichzeitig auch das High-Byte des Videoram verändert werden:

```
POKE648,(Startadd)*256
```

Wie aus der Übersicht zu entnehmen ist, können sowohl Zeichensatz als auch Bildschirmspeicher nur innerhalb der ersten 16 KByte verschoben werden. Es fehlen die vier Adreßbits, die nötig sind, um im gesamten C-64-Speicher verschieben zu können. Diese finden sich in Adresse 56576. Wird ein anderer als der erste 16K-Bereich gewählt, so verschiebt sich automatisch auch immer der Bildschirmspeicher um den entsprechenden 16K-Schritt.
Auch die Sprite-Blöcke und deren Zeiger werden in den entsprechenden Schritten verschoben!

16-K-Bereich:

```
$0000 - $3FFF      0 - 16383  POKE 56576,199: POKE648,4
$4000 - $7FFF  16384 - 32767  POKE 56576,198: POKE648,4+1*64
$8000 - $BFFF  32768 - 49151  POKE 56576,197: POKE648,4+2*64
$C000 - $FFFF  49152 - 65535  POKE 56576,196: POKE648,4+3+64
```

Zugegeben: Zu Demonstrationszwecken reicht die HerumPOKErei

in Sachen Character-Generator gerade noch aus. Möchte man aber für sein neuestes Spiel einen neuen Zeichensatz entwerfen, dann wird diese Art der Zeichendefinition zur langwierigen und -weiligen Tortur.
Aus diesem Grund finden Sie auf den folgenden Seiten zwei Listings, die die Definition des eigenen Zeichensatzes stark vereinfachen.

HILFSPROGRAMME ZUR ZEICHENSATZDEFINITION

Auf den folgenden Seiten werden Sie das umfangreiche Listing eines kompletten Zeicheneditors finden. Das Listing ist zugegeben sehr lang, aber die Vielzahl der Möglichkeiten, die dieser Editor aufweist, werden seine Länge wohl rechtfertigen.
Das Programm ist so aufgebaut, daß der Anwender im Prinzip keinerlei Vorkenntnisse auf dem Character-Generator-Sektor haben muß.
Das Erstellen der einzelnen Zeichen geschieht bequem mittels Tastatur oder Joystick 2. Zeichensätze können auf Disk gespeichert werden. Zahlreiche Editierhilfen wie beispielsweise das Drehen eines beliebigen Zeichens um 90, 180 oder 270 Grad, Invertieren, Vertauschen oder Duplizieren von Zeichen etc., etc.
Abgerundet wird das Programm durch den eingebauten Programm-Generator, der auf Wunsch den BASIC-Lader für den selbstdefinierten Zeichensatzes schreibt (bis zu 256 Zeichen).

Doch hier zunächst das Listing:

```
5 REM PROGRAM BY T.WELTNER*BAHNHOFSALLEE 10*3260 RINTELN 1*T
EL.:(05751) 5676
10 POKE45,255:POKE46,66:CLR
20 POKE788,52:O=53248:POKEO+32,1:POKEO+33,1:PRINT"■■":POKE21
4,4:PRINT
30 IFPEEK(51000)=1THENPOKE51000,0:GOTO350
40 POKE53247,0:PRINTTAB(7)"■                              ":PRINT
50 POKEO+42,3:PRINTTAB(7)" * ZEICHEN-EDITOR *":PRINT
60 PRINTTAB(7)"■■                          ":POKE214,16:PRINT
70 PRINT"■■■][ AENDERN DES ZEICHENSATZES":PRINT
80 PRINT"■■■][ EINGEBAUTER PROGRAMMGENERATOR"
90 POKE214,21:PRINT:PRINT"■■■■BY T.WELTNER / 22.SEPTEMBER 1
984 "
110 :
120 REM *** SPRITE INITIALISIERUNG ***
130 FORK=0TO7:POKE704+K*3,255:POKE705+K*3,0:POKE706+K*3,0:NE
XTK
140 FORK=0TO38:POKE704+8*3+K,0:NEXTK
150 FORK=0TO63:POKE832+K,0:NEXT:REM BLOCK 13 FULL
170 :
180 REM *** CURSOR FLASH ***
190 READA:IFA=-1THEN240
200 POKE912+B,A:C=C+A:B=B+1:GOTO190
210 DATA174,142,3,232,224,16,240,6,142,142,3,76,52,234,169,0
,141,142
220 DATA3,173,143,3,201,4,240,11,141,21,208,169,4,141,143,3,
76,52,234
230 DATA141,21,208,169,0,76,175,3,120,169,144,160,3,141,20,3
,140,21,3,88,96,-1
240 IFC<>6009THENLIST200-230
260 :
270 C=0:REM *** KOPIERROUTINE ***
280 FORA=0TO48:READB:POKE12*4096+A,B:C=C+B:NEXT
290 DATA162,16,169,0,141,14,220,169,51,133,1,169,208,160,0,1
32,34,133
300 DATA35,169,112,132,36,133,37,177,34,145,36,200,208,249,2
30,35,230
310 DATA37,202,208,242,169,55,133,1,169,1,141,14,220,96
320 IFC<>5798THENPRINT"? DATENFEHLER":LIST290-310
330 SYS49152:REM AKTIVIEREN
350 :
360 CLR:O=53248:POKE198,0:REM *** MENUE ***
370 PRINT"■■                * MENUE *                    ■■"
380 PRINT"■ 1 ■■  ERSTELLEN EINES ZEICHENS":PRINT
400 PRINT"■ 2 ■■  DATENEINGABE":PRINT
410 PRINT"■ 3 ■■  ZEICHENUEBERSICHT":PRINT
420 PRINT"■ 4 ■■  ZEICHEN TAUSCHEN":PRINT
430 PRINT"■■■";:FORT=0TO34:PRINT"=";:NEXTT:PRINT:PRINT
440 PRINT"■■ 5 ■■  ZEICHENSATZ LADEN":PRINT
450 PRINT"■■ 6 ■■  ZEICHENSATZ SAVEN":PRINT
455 PRINT"■■ 7 ■■  PROGRAMM-GENERATOR":PRINT
456 PRINT"■■■";:FORT=0TO34:PRINT"=";:NEXTT:PRINT:PRINT
460 PRINT"■■ 8 ■■  PROGRAMM BEENDEN"
```

```
470 GETA$:IFA$=""THEN470
480 IFA$=CHR$(3)THENPRINT"■? BREAK IN MENUE";:END
490 A=VAL(A$):IFA=0ORA>8THENGOTO470
500 ONAGOTO530,590,800,1020,1490,1270,3390,1230
520 :
530 REM *** ZEICHENEDITOR ***
540 GOSUB1790: REM ABFRAGE ZEICHEN
550 GOSUB1970:GOTO2150:REM BILDSCHIRM INITIALISIEREN/BEWEGUN
GSROUTINE
580 :
590 REM *** DATENEINGABE ***
600 PRINT"■■                DATENEINGABE                    "
610 PRINT:PRINT:PRINT"■GEBEN SIE NACHFOLGEND DIE 8 DATEN EIN
!"
620 PRINT:PRINT:FORVV=0TO7:PRINTVV+1"■";:K(VV)=0:K$(VV)="":I
NPUT",";K$(VV):NEXT
621 FORVV=0TO7:K(VV)=VAL(K$(VV)):IFK(VV)<0ORK(VV)>255THENK(V
V)=0
622 NEXTVV
630 F4=5:F3=1:FL=12:GOSUB3210:F3=0:FL=0:F4=0
640 POKE214,15:PRINT:PRINT"SOLL DIESES ZEICHEN ABGESPEICHERT
 WER-  DEN (J/N) ?"
650 GETA$:IFA$=""THEN650
660 IFA$="J"THENKL$="■ZEICHENEINGABE":GOTO720
670 PRINT:PRINT"ZURUECK ZUM MENUE (J/N)?"
680 GETA$:IFA$=""THEN680
690 IFA$="J"THENGOTO360
700 GOTO590
705 :
710 REM *** EINGABE ***
720 P=53246:PRINT"■";KL$:PRINT"■■■ZEICHEN ?■■";:GOSUB785
721 POKE204,0:GETA$:IFA$=""THEN721
722 A=PEEK(207):IFATHEN722
723 PRINT:IFA$=CHR$(13)THENA$=" "
724 POKE204,1:PRINT"■■VERAENDERBARES ZEICHEN: ";A$:FORT=0TO2
4:PRINTCHR$(101);
725 NEXTT:PRINT:PRINT:PRINT"■ 1 ■■■■";E$
726 PRINT:PRINT"■ 2 ■■■■";F$
727 PRINT:PRINT"■ 3 ■■■■UTO ";X$
728 PRINT:PRINT:PRINT"■■ 4 ■■■■FEHLER"
729 PRINT:PRINT:PRINT"■■ 5 ■■■■O.K., WEITER"
731 GETC$:IFC$=""THEN731
733 IFC$="5"THEN750
734 IFC$="3"THENGOSUB780
735 IFPEEK(P-1)=1THENGOTO739
737 IFC$="1"THENRV=RV+128:POKE214,5:PRINT:PRINTTAB(5)"NEGA":
POKEP-1,1:GOTO731
739 IFC$="1"THENRV=RV-128:POKE214,5:PRINT:PRINTTAB(5)"POSI":
POKEP-1,0:GOTO731
741 IFPEEK(P-2)=1THENGOTO745
743 IFC$="2"THENRV=RV+256:POKE214,7:PRINT:PRINTTAB(5)"KLEIN/
GROSS":N=1:GOTO790
745 IFC$="2"THENRV=RV-256:POKE214,7:PRINT:PRINTTAB(5)"GROSS/
GRAF.":N=0:GOTO790
747 IFC$="4"THENGOTO720
```

```
750 POKE646,1:PRINT"■■"A$:POKE646,0:CO=PEEK(1024+40)+RV:IFCH
<>1THENGOSUB3380
760 IFCH=1THENRETURN
770 POKE214,20:PRINT:PRINT"OK  ":PRINT"       ■":GOTO670
780 IFPEEK(P)=0THENGOTO782
781 AU$="":POKE214,9:PRINT:PRINTTAB(5)"■■■■■AUS":POKEP,0:GOT
O731
782 AU$="A":POKE214,9:PRINT:PRINTTAB(5)"■■■■■EIN":POKEP,1:GO
TO731
785 RV=0:X$="AUS":E$="POSITIV":F$="GROSS/GRAF.":IFPEEK(P)=1T
HENX$="EIN"
786 IFPEEK(P-1)=1THENE$="NEGATIV":RV=RV+128
787 IFPEEK(P-2)=1THENF$="KLEIN/GROSS":RV=RV+256
788 RETURN
790 POKEP-2,N:GOTO731
791 :
800 MD=189:REM *** UEBERSICHT ***
810 POKE53272,MD:POKE56576,150:POKE648,108:PRINT"■":A0=55296
:FORA=0TO255
820 POKE27648+2*A,A:POKEA0+2*A,14:IFCH<>1THENPOKE27648+A+600
,A:POKEA0+A+600,3
829 NEXTA:IFCH=1THENRETURN
830 GETA$:IFA$=""THEN830
840 POKE53272,21:POKE56576,151:POKE648,4
841 GOTO360
1000 :
1020 REM *** TAUSCHEN ***
1030 PRINT"■■          ZEICHEN TAUSCHEN          "
1040 PRINT"■■■ 1 ■■■■■ZEICHEN KOPIEREN"
1041 PRINT"■■ 2 ■■■■■ZEICHEN TAUSCHEN"
1042 PRINT"■■■■ 3 ■■■■■MENUE"
1050 GET0$:IF0$=""THEN1050
1055 IFVAL(0$)<1ORVAL(0$)>2THENPOKE51000,1:RUN
1060 IF0$="2"THENKL$="■TAUSCHEN: A <-> B":GOTO1120
1100 KL$="■KOPIEREN: A(A) -> A(B)"
1120 CH=1:GOSUB720:C0=CO:GOSUB720:CH=0:PRINT
1130 IF0$="2"THENGOSUB1190:GOTO1150
1140 FORV=0TO7:POKE28672+C0*8+V,PEEK(28672+C0*8+V):NEXTV
1150 POKE214,15:PRINT:PRINT"ZURUECK ZUM MENUE (J/N) ?"
1160 GETA$:IFA$=""THEN1160
1170 IFA$="J"THENGOTO360
1180 PRINT"■":GOTO1060
1190 FORV=0TO7:C1(V)=PEEK(28672+C0*8+V):C2(V)=PEEK(28672+C0*
8+V):NEXTV
1200 FORV=0TO7:POKE28672+C0*8+V,C2(V):POKE28672+C0*8+V,C1(V)
:NEXTV:RETURN
1210 :
1230 REM *** PROGRAMM BEENDEN ***
1240 PRINT"■BIS ZUM NAECHSTEN MAL...":POKE788,49:PRINT"■■■NE
W":END
1250 :
```

```
1270 REM *** ABSAVEN ***
1280 PRINT"■■        ZEICHENSATZ SPEICHERN                "
1290 PRINT"■■■DER VON IHNEN GEAENDERTE ZEICHENSATZ AB"
1300 PRINT"ADRESSE (DEZ.) 28672 KANN NUN ABGESPEI-"
1310 PRINT"CHERT WERDEN.":PRINT:IFHI=0THENHI=112:HII=128
1320 PRINT"ER BELEGT AUF DISKETTE CA. 17(9) KBYTE."
1330 PRINT"AUS PROGRAMMTECHNISCHEN GRUENDEN KANN"
1340 PRINT"DER ZEICHENSATZ NICHT AUF DATASETTE"
1350 PRINT"GESAVED WERDEN.":PRINT:PRINT
1360 IFPEEK(186)<>8THENPRINT"■NUR FUER DISKETTE!":FORT=1TO99
9:NEXTT:GOTO360
1380 PRINT:PRINT"UNTER WELCHEM NAMEN SOLL DER ZEICHEN-"
1385 PRINT"SATZ ABGESPEICHERT WERDEN (DURCH EINGABE";
1387 PRINT"VON 'PARAMETER' PARAM.AENDERUNGEN MOEG-"
1390 A$="":INPUT"LICH)";A$:IFA$=""THENGOTO360
1395 IFA$="PARAMETER"ORRIGHT$(A$,3)="PARAMETER"THENGOTO1470
1400 IFKE=0THENIFLEFT$(A$,1)<>CHR$(215)THENB$=CHR$(215)+A$:G
OTO1420
1410 B$=A$
1420 PRINT"■■■■"
1430 PRINT"POKE43,0:POKE44,";H2;":POKE45,0:POKE46,";H4;"■■"
1440 PRINT"SAVE"CHR$(34)"@:"B$CHR$(34)",8■■■"
1450 PRINT"POKE43,1:POKE44,8:POKE51000,1:GOTO10":PRINT"■"
1460 FORA=0TO7:POKE631+A,13:NEXT:POKE198,8:END
1470 POKE646,12:PRINT"■■                    PARAMETER
    "
1471 PRINTW1$:PRINT:PRINT"■ 1 ■  1/1 SAVEN":PRINT:PRINT"■ 2
  1.HAELFTE":PRINT
1472 PRINT"■ 3 ■  2.HAELFTE":PRINT:PRINT"■ 4 ■  KENNUNG AUS"
1473 PRINT:PRINT"■ 5 ■  FEHLER":PRINT:PRINT"■ 6 ■  O.K., WEI
TER"
1474 GETW$:IFW$=""THEN1474
1475 IFW$="5"THENKE=0:H2=112:H4=128:POKE53247,0:W2$="":W3$="
":GOTO1470
1476 IFW$="2"THENH2=112:H4=120:W2$="1.HAELFTE SAVEN.."
1477 IFW$="3"THENH2=120:H4=128:W2$="2.HAELFTE SAVEN.."
1478 IFW$="4"THENPOKE53247,1:W3$="KENNUNG AUS"
1479 IFW$="1"THENH2=112:H4=128:W2$="1/1 SAVEN........"
1480 IFW$="6"THENGOTO1280
1481 PRINT"■■";W2$;"■■■";W3$
1482 W1$="■■■":GOTO1471
1483 :
1490 REM *** ZEICHENSATZ LADEN ***
1500 PRINT"■■          ZEICHENSATZ LADEN                "
1520 PRINT"■■■■■SIE KOENNEN NUN EINEN ZUVOR AUF DISKETTE";
1530 PRINT"ABGESPEICHERTEN ZEICHENSATZ LADEN.":PRINT:PRINT"■
('$' = DIRECTORY)"
1540 A$="":PRINT:PRINT:PRINT:INPUT"NAME DES ZEICHENSATZES";A
$
1550 IFA$=""THENGOTO360
1560 IFA$="$"ANDPEEK(186)=8THENGOSUB1640:PRINT:GOTO1540
```

```
1570 IFPEEK(53247)=0THENIFLEFT$(A$,1)<>CHR$(215)THENB$=CHR$(
215)+A$:GOTO1600
1590 B$=A$
1600 IFPEEK(186)=1THENPRINT"■NUR FUER DISK-BENUTZER":FORT=0T
0999:NEXTT:GOTO360
1610 FORA=0TO8:POKE631+A,13:NEXT:PRINT"■■■■"
1620 PRINT"■■■■":PRINT"LOAD"CHR$(34)B$CHR$(34)",8,1■■■■■■"
1630 PRINT"POKE51000,1:GOTO10":PRINT"■":POKE198,8:END
1640 PRINT"■ =ZEICHENSATZ"
1650 PRINT:OPEN1,8,15,"I0":OPEN2,8,2,"#":Q=18:W=1
1660 PRINT#1,"B-R";2;0;Q;W:PRINT#1,"B-P";2;0
1670 GET#2,X$:IFX$=""THENX$=CHR$(0)
1680 Q=ASC(X$)
1690 GET#2,X$:IFX$=""THENX$=CHR$(0)
1700 W=ASC(X$)
1710 FORX=0TO7:PRINT#1,"B-P";2;X*32+5:Y$="":FORY=0TO15
1720 GET#2,X$:IFX$=""THENX$=CHR$(0)
1730 IFASC(X$)=160THEN1750
1740 Y$=Y$+X$:NEXTY
1750 PRINTY$:NEXTX:IF0<>0THEN1660
1760 CLOSE1:CLOSE2:RETURN
1780 :
1790 REM *** ZU AENDERNDES ZEICHEN ***
1810 KL$="■            ZEICHENEDITOR                "
1840 CH=1:GOSUB720:CH=0
1850 RETURN
1950 :
1960 REM *** BILDSCHIRMINITIALISIERUNG ***
1970 PRINT"■■              ZEICHENEDITOR                "
1980 RV$="OFF":P=53246:IFPEEK(P-1)THENRV$="ON"
1981 ZS$="GROSS/GRAFIK":IFPEEK(P-2)=1THENZS$="KLEIN/GROSS"
1982 PRINT"ZEICHEN: ";A$;"■■REVERS ";RV$;"■■■";ZS$;"■";:IFPE
EK(P)=1THENPRINT"A";
2010 PRINT"■■■■■■■■■■■76543210■■■■■76543210"
2020 PRINT"■■■■■■■■■■"
2030 FORK=0TO7:PRINT"■■"K"■■■■■■■■■"K"■■■■■■■■■"K:NEXTK
2040 PRINT"■■■■■■■■■"
2050 PRINT"■■■■■■76543210■■■■■76543210"
2060 PRINT"■■■■■■■■■ USER ■■■■■■■■MATRIX■■■"
2070 IFFL=7THENRETURN
2080 PRINT"■■  F1  ■  F3  ■  F5  ■  F7  ■■■  F2  "
2100 PRINT"■ TEST ■ DATEN■ ROM  ■ RAM  ■■■ROTATE"
2140 :
```

```
2150 REM *** BEWEGUNGSROUTINE ***
2160 POKE959,144:POKE961,3:SYS957:POKE0+21,5:POKE2042,11
2170 SH=73:SV=106:BV=0:BH=0:REM SPRITE-POSITION
2180 GETA$:POKE198,0:JO=PEEK(56320):IFA$=""THENAS=0:GOTO2190
2185 AS=ASC(A$)
2190 IFAS=145OR(JOAND1)=0THENPOKE911,4:GOSUB2450:REM CRSR UP
2200 IFAS=17OR(JOAND2)=0THENPOKE911,4:GOSUB2480:REM CRSR DOW
N
2210 IFAS=29OR(JOAND8)=0THENPOKE911,4:GOSUB2510:REM CRSR RIG
HT
2220 IFAS=157OR(JOAND4)=0THENPOKE911,4:GOSUB2540:REM CRSR LE
FT
2230 IFAS=19THENBH=0:BV=0:SH=73:SV=106
2240 IFAS=3THENPOKE959,52:POKE961,234:SYS957:POKE0+21,0:GOTO
360
2250 IFAS=133THENGOSUB2820: REM F1
2260 IFAS=134THENGOSUB2570: REM F3
2270 IFAS=137THENGOSUB3700: REM F2
2280 IFAS=135THENGOSUB2780: REM F5
2290 IFAS=139THENF2=12:FL=1:GOSUB3210:F2=0:FL=0:GOTO2180:REM
 F6
2300 IFAS=140THENF2=12:GOSUB3210:F2=0:GOTO2180:REM F8
2310 IFAS=136THENGOSUB2800: REM F7
2320 IFAS=147THENGOSUB2760: REM CLEAR
2330 IFAS=138THENGOSUB3060: REM F4
2340 IFAS=32THENGOSUB3000: REM LOESCHEN (SPACE)
2350 IFAS=13THENGOSUB2970: REM RETURN
2370 IFAS<128ANDAS>32OR(JOAND16)=0THENFL=1:GOSUB3020:FL=0
2380 POKE0+4,SH:POKE0+5,SV:GOTO2180
2450 REM *** UP ***
2460 IFSV>106THENSV=SV-8:BV=BV-40
2470 RETURN
2480 REM *** DOWN ***
2490 IFSV<106+7*8THENSV=SV+8:BV=BV+40
2500 RETURN
2510 REM *** RIGHT ***
2520 IFSH<73+7*8THENSH=SH+8:BH=BH+1
2530 RETURN
2540 REM *** LEFT ***
2550 IFSH>73THENSH=SH-8:BH=BH-1
2560 RETURN
2570 REM *** DATAS AUSGEBEN (F1) ***
2580 FORVV=0TO7:K(VV)=0:NEXT:GOSUB3130
```

```
2590 POKE646,12:PRINT"■■              MANIPULATIONEN
    "
2600 PRINT:PRINT:PRINT:PRINT"■ 1 ■  DATEN AUSGEBEN":PRINT
2610 PRINT"■ 2 ■  ZEICHEN AENDERN":PRINT:PRINT"■ 3 ■  ZEICHE
N SPEICHERN":PRINT
2620 PRINT:PRINT"■ 4 ■  MENUE":POKE198,0
2621 PRINT:PRINT"■ 5 ■  EDITOR"
2630 GETS$:IFS$=""THEN2630
2633 IFS$="5"THENGOTO530
2640 IFS$="4"THENPOKE51000,1:RUN
2650 IFS$="1"THEN2700
2655 IFS$="2"THENKL$="■EINGABE DES NEUEN ZEICHENS":CH=1:GOSU
B720:CH=0:GOTO2590
2660 IFS$="3"THENGOSUB3380:CLR:O=53248:GOTO2590
2700 PRINT"■■          DATENAUSGABE                    "
2710 POKE214,4:PRINT:PRINT"■DIE DATEN DES ZEICHENS LAUTEN:":
PRINT
2720 FORVV=0TO7:PRINTVVTAB(12)K(VV)"=":NEXT:FL=12:F4=5:F3=1:
GOSUB3210:FL=0:F3=0
2722 PRINT:POKE646,6:PRINT"BITTE EINE TASTE DRUECKEN !"
2730 GETF$:IFF$=""THEN2730
2731 GOTO2590
2760 REM *** USER CLEAR (CLEAR-TASTE) ***
2770 C2=CO:CO=32:F2=12:FL=1:GOSUB3210:F2=0:CO=C2:FL=0:RETURN
2780 REM *** MATRIX RECHTS (F5) ***
2790 FL=1:GOTO3210
2800 REM *** USED RECHTS (F7) ***
2810 FL=0:GOTO3210
2820 REM *** ZEICHEN TESTEN (F3) ***
2830 H=0:FORVV=0TO7:K(VV)=0:NEXT:GOSUB3130
2840 POKE199,1:POKE646,12:PRINT"■                  TEST
          "
2850 POKEO+21,3:POKEO,100:POKEO+1,100
2860 POKE2040,13:POKE2041,13:POKEO+39,6:POKEO+40,6
2870 POKEO+2,192:POKEO+3,94:POKEO+29,2:POKEO+23,2
2880 FORVV=0TO7:POKE832+ZA,K(VV):ZA=ZA+3:NEXT:ZA=0
2890 POKE214,7:PRINT:PRINTTAB(8)"1:1";TAB(20)"1:2"
2891 POKE214,15:PRINT:PRINT"SOLL DAS ZEICHEN GESPEICHERT WER
DEN ?(J/N)"
2900 GETA$:IFA$=""THEN2900
2910 IFA$<>"J"THENPOKEO+21,0:GOSUB1970:FL=12:F2=12:GOSUB3210
:FL=0:F2=0:GOTO2150
2920 POKEO+21,0:GOSUB3380
2930 CLR:O=53248:PRINT"ZURUECK ZUM MENUE (J/N) ?"
2940 GETA$:IFA$=""THEN2940
2950 IFA$="J"THENGOTO360
2960 GOTO530
2970 REM *** RETURN-TASTE ***
2980 IFSV<106+7*8THENSH=73:SV=SV+8:BH=0:BV=BV+40:RETURN
2990 SH=73:BH=0:RETURN
3000 REM *** SPACE-TASTE (LOESCHEN) ***
3010 POKE1024+6+7*40+BV+BH,32:RETURN
```

```
3020 REM *** PUNKT SETZEN ***
3030 POKE1024+6+7*40+BV+BH,81:POKE55296+6+7*40+BV+BH,3:RETUR
N
3050 :
3060 REM *** ZEICHEN INVERTIEREN ***
3070 POKE0+41,14
3080 FORK=0TO7:FORKK=0TO7:PE=PEEK(1024+286+KK+40*K):IFPE=32T
HENPE=81:GOTO3100
3090 IFPE=81THENPE=32
3100 IQ=286+KK+40*K:POKEIQ+1024,PE:POKEIQ+55296,7:NEXTKK:NEX
TK:POKE0+41,3
3110 RETURN
3120 :
3130 REM *** DATAS BERECHNEN ***
3140 POKE959,52:POKE961,234:SYS957:POKE0+21,0:H=0:REM U-INTE
RRUPT & SPRITE OFF
3150 FORVV=0TO7:FORV=0TO7
3160 A(V)=PEEK(1024+H+293-V):M=2↑V:IFA(V)=81THENK(VV)=K(VV)+
M
3180 NEXTV:PRINTK(VV)"■";:V=0:H=H+40:NEXTVV:H=0:RETURN
3200 :
3210 REM *** ZEICHEN VERGROESSERN ***
3220 POKE0+41,14:JJ=0:DD=293:HK=32:IFF3=1THENHK=87
3230 IFF2=12THENDD=DD-13
3240 IFFL=1THENCG=13*4096:FA=5:GOTO3260
3250 CG=28672:FA=13
3260 IFF4=0THENF4=FA
3270 FORI=0TO7:REM ZEILENZAEHLER
3280 IFFL=1THENPOKE56334,0:POKE1,51:REM CHARACTERGENERATOR A
USLESEN
3290 IFFL=12THENZZ=K(I):GOTO3310
3300 ZZ=PEEK(CG+8*CO+I)
3310 IFFL=1THENPOKE1,55:POKE56334,1
3311 POKE1028+6*40+JJ,32:POKE55300+7*40+JJ,FA:POKE1028+7*40+
JJ,31
3320 FORJ=0TO7:AK=ZZAND2↑J:GETST$
3322 IFST$<>""THENIFASC(ST$)=3THENPOKE1028+7*40+JJ,32:POKE0+
41,3:RETURN
3330 IFAKTHENPOKE1024+13+DD-J+JJ,81:POKE55296+13+DD-J+JJ,F4:
GOTO3350
3340 POKE1024+13+DD-J+JJ,HK:POKE55296+13+DD-J+JJ,FA
3350 NEXT:JJ=JJ+40:NEXTI:FL=0:F4=0:POKE0+41,3:POKE1028+6*40+
JJ,32:RETURN
3370 :
3380 REM *** ZEICHEN SPEICHERN ***
3382 AU=PEEK(53246):FORVV=0TO7:POKE28672+8*CO+VV,K(VV)
3383 IFCO>383ORCO>127ANDCO<256THENRV=1
3384 IFAU=1THENIFRV=1THENPOKE28672+(8*(CO-128))+VV,255-K(VV)
:GOTO3389
3386 IFAU=1THENPOKE28672+(8*(CO+128))+VV,255-K(VV)
3389 NEXT:RETURN
3390 :
```

```
3391 REM *** PROGRAMMGENERATOR ***
3392 PRINT"■M O M E N T !":WK=159:POKE53000,0:POKE53001,0
3393 DIMWH(255):DIMWI(255):FORA=0TO255:WH(A)=-1:WI(A)=-1:NEX
T:MD=189
3394 CH=1:GOSUB810:CH=0
3396 POKE27648+512,WK:POKE55296+512,13:POKE27648+514,141:POK
E55296+514,2
3397 POKE27648+516,144:POKE55296+516,2:GOTO4000
3399 XY=PEEK(53000):YX=PEEK(53001):IFCO>255THENGOTO3405
3400 FORG=0TO7:K(G)=PEEK(28672+8*CO+G):NEXTG:IFWH(CO)>-1THEN
3402
3401 WH(CO)=20000+XY*9:XY=XY+1:POKE53000,XY:IFXY+YX>255THENG
OTO3409
3402 POKEWH(CO),CO:FORYO=0TO7:POKEWH(CO)+1+YO,K(YO):NEXTYO:R
ETURN
3405 FORG=0TO7:K(G)=PEEK(28672+8*CO+G):NEXTG:KO=CO-256:IFWI(
KO)>1THEN3407
3406 WI(KO)=22049+YX*9:YX=YX+1:POKE53001,YX:IFXY+YX>255THENG
OTO3409
3407 POKEWI(KO),KO:FORI=0TO7:POKEWI(KO)+1+I,K(I):NEXTI:RETUR
N
3409 POKE53002,0:POKE53100,PEEK(53000):POKE53200,PEEK(53001)
3410 ZK=PEEK(53000):ZK=ZK-1:IFZK<0THEN3460
3420 POKE53000,ZK:CO=PEEK(20000+9*ZK):FORI=0TO7:K(I)=PEEK(20
000+9*ZK+I+1):NEXTI
3430 PRINT"■■■■":ZC=PEEK(53002):POKE53002,ZC+1
3440 PRINTZC+370"DATA"CO"■,";:FORI=0TO6:PRINTK(I)"■,";:NEXTI
:PRINTK(7)
3450 PRINT"GOTO3410":FORM=0TO9:POKE631+M,13:NEXTM:POKE198,10
:PRINT"■":END
3460 ZL=PEEK(53001):ZL=ZL-1:IFZL<0THEN3510
3470 POKE53001,ZL:CO=PEEK(22049+9*ZL):FORI=0TO7:K(I)=PEEK(22
049+9*ZL+I+1):NEXTI
3480 PRINT"■■■■":ZC=PEEK(53002):POKE53002,ZC+1
3490 PRINTZC+370"DATA"CO+256"■,";:FORI=0TO6:PRINTK(I)"■,";:N
EXT:PRINTK(7)
3500 PRINT"GOTO3460":GOTO3525
3510 ZC=PEEK(53002)+370:PRINT"■■■■"ZC+1"DATA-1":PRINTZC+2"::
:"
3511 PRINT"340 READA:IFA=-1THEN360"
3520 PRINT"350 FORK=0TO7:READB:POKE28672+A*8+K,B:NEXTK:GOTO3
40"
3521 PRINT"360 POKE53272,189:POKE56576,150:POKE648,108:?CHR$
(147)"
3522 PRINT"GOTO3530"
3525 FORM=0TO9:POKE631+M,13:NEXTM:POKE198,10:PRINT"■":END
3530 POKE53002,4:POKE53010,0
3540 AA=PEEK(53002):IFAA=0THENGOTO3620
3550 POKE53002,AA-1
3560 PRINT"■■■■":ZD=PEEK(53010):FORA=0TO7:PRINTZD+10*A:NEXTA
:POKE53010,ZD+6*10
```

```
3570 PRINT"GOTO3540":GOTO3525
3620 KK=PEEK(43)+256*PEEK(44)+500+(PEEK(53100)+PEEK(53200)*1
0)
3622 FORJ=KKTOPEEK(45)+256*PEEK(46)
3630 IFPEEK(J)=58ANDPEEK(J+1)=58ANDPEEK(J+2)=58THENGOTO3650
3640 NEXTJ
3650 VH=INT((J/256)+1)
3655 PRINT"■■■":PRINT"POKEJ-3,0:POKEJ-4,0:POKE45,0:POKE46,"V
H":CLR":GOTO3525
3699 :
3700 REM *** ROTATE ***
3710 INPUT"WIEVIEL RECHTSDREHUNGEN (1/2/3) ";A$
3720 POKE781,23:SYS59903:PRINT"■":FORA=0TO7:K(A)=0:W(A)=0:NE
XTA
3730 D=VAL(A$):IFD=0THENGOTO2150
3731 GOSUB3130:POKE781,24:SYS59903:IFD=<10RD>3THEND=1
3732 POKE0+41,14:POKE959,144:POKE961,3:SYS957:FORC=1TOD:FORA
=0TO7:W(A)=0:NEXTA
3825 FORB=0TO7:FORA=7TO0STEP-1:M=2↑A:IFK(B)-M>=0THENW(A)=W(A
)+2↑(B):K(B)=K(B)-M
3830 NEXTA:NEXTB:FORA=0TO7:K(A)=W(A):NEXTA:FORA=0TO7:A(A)=K(
A):NEXTA
3840 FORA=0TO7:K(A)=A(7-A):NEXTA:NEXTC
3850 F2=12:FL=12:GOSUB3210:PRINT"■■":F2=0:FL=0:POKE0+41,3:GO
TO2150
4000 FR=14:LE=0:R=0:IFKR=255THENPRINTCHR$(14)
4001 GETC$:JO=PEEK(56320):IFC$=""THENC$=CHR$(0)
4002 C=ASC(C$):IFC=157OR(JOAND4)=0THENIFLE>0THENR=-2:GOSUB40
50
4004 IFC=29OR(JOAND8)=0THENIFLE<516THENR=2:GOSUB4050
4006 IFC=13OR(JOAND16)=0THENCO=PEEK(27648+LE)+KR:FR=7:R=0:GO
SUB4050:GOSUB4060
4007 IFFL=3THENPOKE55296+LE,1
4008 IFFL=6THENFL=0:POKE55296+LE,FR
4009 FL=FL+1
4010 IFC=145OR(JOAND1)=0THENIFLE>=40THENR=-40:GOSUB4050
4015 IFC=17OR(JOAND2)=0THENIFLE<=11*40+36THENR=+40:GOSUB4050
4030 GOTO4001
4050 FL=3:POKE55296+LE,FR:LE=LE+R:FR=PEEK(55296+LE):RETURN
4060 IFLE=512THENIFPEEK(53272)<>191THENKR=256:WK=32:MD=191:G
OTO3394
4062 IFLE=514THENPRINTCHR$(142):POKE53272,21:POKE56576,151:P
OKE648,4:POKE51000,1:RUN
4064 IFLE=516THENGOSUB4070:GOTO3409
4065 GOSUB3399:RETURN
4070 POKE53272,21:POKE56576,151:POKE648,4
4080 PRINT"■":POKE214,10:PRINT:PRINT"DER EDITOR WIRD HIERDUR
CH GELOSCHT...!"
4081 PRINT"SIND SIE SICHER (J/N) ??!"
4090 GETK$:IFK$=""THEN4090
4095 IFK$="J"THENRETURN
4096 GOTO 360
```

Das Programm ist größtenteils durch Menütechnik selbsterklärend. Nun noch einige Detailerklärungen:

AUTO: Diese Funktion sorgt dafür, daß neben dem eigentlichen Zeichen auch dessen reverser Verwandte definiert wird. Man spart so viel Arbeit beispielsweise bei der Definition eines neuen Schriftsatzes, da die reversen Buchstaben automatisch umdefiniert werden.

PROGRAMMGENERATOR: Nach Aufrufen dieses Punktes wird der erste Zeichensatz aufgelistet. Mittels Tastatur oder Joystick führt man nun auf die Zeichen, die in den BASIC-Lader übernommen werden sollen. Anschließend drückt man die RETURN-Taste. Die Eingabe wird durch einen Farbwechsel des betreffenden Zeichens quittiert.
Unten rechts sind drei Funktionen wählbar:
1. (Pfeil links): Auflisten des 2.Zeichensatzes. Auch von ihm können Zeichen übernommen werden.
2.(revers M): Zurück ins Menü bei falscher Eingabe.
3.(revers P): Aktivieren des Programmgenerators, der den BASIC-Lader erzeugt.

Der entworfene Zeichensatz liegt am BASIC-Ende.

DESIGN IM LISTING

Eine weitere Möglichkeit, Zeichen bequem und übersichtlich umzudefinieren, bietet das folgende, relativ kurze Listing. Die Besonderheit hierbei ist, daß das Zeichen nicht durch einen herkömmlichen BASIC-Lader abgespeichert, sondern direkt im Listing als Figur abgelegt ist.
Bei der Zeichendefinition wird diese Figur ausgelesen und in Byte-Werte umgerechnet.
Es sollte jedoch nicht unerwähnt bleiben, daß diese Art der Zeichendefinition etwas langsamer ist als ein herkömmlicher BASIC-Lader, da die Bytewerte erst errechnet werden müssen. Auf der anderen Seite sind die Vorteile zu sehen: Es ist wesentlich übersichtlicher, und Fehler durch falsche Zahlenwerte können ebenfalls nicht auftreten, da das Zeichen ja in seiner äußeren Form eingegeben wird.

```
1 PRINT"■  BITTE WARTEN! KOPIERVORGANG BENOETIGT■   CA. 1 MI
NUTE !"
10 POKE56334,0:POKE1,51
15 FORK=0TO4095
20 POKE57344+K,PEEK(53248+K)
25 NEXT
30 POKE1,55:POKE56334,1
50 POKE53272,PEEK(53272)ANDNOT6OR8
60 POKE56576,196:POKE648,3*64+4:PRINT"■"
100 READA$:IFA$<>"DESIGN"THENGOTO100
110 READB$:B$=LEFT$(B$,1)
120 PRINT"■";B$
130 CO=PEEK(50176)
140 FORK=0TO7:READC$
141 IFC$="ENDE"THENPRINT"? MISSING STATEMENTS IN";PEEK(63)+2
56*PEEK(64):END
150 IFLEN(C$)<>8THENPRINT"? TOO MANY SIGNS IN";PEEK(63)+256*
PEEK(64):END
160 FORJ=7TO0STEP-1:K$=MID$(C$,8-J,1):IFK$="*"THENBY=BY+2↑J
170 NEXTJ:POKE57344+8*CO+K,BY:BY=0:NEXTK
180 READA$:IFA$="DESIGN"THENGOTO110
```

```
200 DATA DESIGN,A
210 DATA********
220 DATA*......*
230 DATA*......*
240 DATA*......*
250 DATA*......*
260 DATA*......*
270 DATA*......*
280 DATA********
290 DATA DESIGN,B
300 DATA          *******.
310 DATA          *......*
320 DATA          *......*
330 DATA          *.....*.
340 DATA          ******..
350 DATA          *......*
360 DATA          *......*
370 DATA          *******.
390 DATA ENDE
```

Wie Sie dem Listing entnehmen können, steht am Anfang in der ersten Datazeile der DESIGN-Befehl- eine Zeichendefinition, gefolgt von dem Zeichen, das verändert werden soll. Anschließend folgen die acht DATA-Zeilen, die die äußere Form des Zeichens beinhalten. Ein Stern steht dabei jeweils für einen gesetzten Punkt.

ZUM THEMA MULTI COLOR...

Sicherlich werden Sie die Möglichkeit kennen, Sprites im sogenannten Multi-Color-(MC-) Modus darzustellen: Die horizontale Auflösung nimmt um die Hälfte ab. Zur Gestaltung des Zeichens können jedoch gleichzeitig drei Farben benutzt werden. Auch selbstdefinierte Zeichen lassen sich im

Multi-Color-Mode darstellen. Wichtig hierfür ist das Register 22 (Steuerregister 2) des VIC. Dieses Register (genau genommen das 4.Bit dieses Registers) entscheidet darüber, ob Zeichen in MC-Darstellung abgebildet werden oder nicht.

Mit folgender Zeile schalten Sie den MC-Mode ein:

```
POKE 53270,PEEK(53270) OR 16
```

Durch

```
POKE 53270, PEEK(53270) AND NOT 16
```

gelangen Sie wieder in den Normalzustand.
Typische Erscheinungen des MC-Modes: Normalerweise stehen Ihnen 16 Farben zur Verfügung, in denen Zeichen auf dem Bildschirm ausgegeben werden können. Im MC-Mode schrumpft diese Zahl zunächst auf acht Farben. Es sind dies die Grundfarben schwarz bis gelb. Verwenden Sie im MC-Mode andere als diese Farben, so werden Sie die damit ausgegebenen Zeichen auf dem Bildschirm kaum wiedererkennen. Sie sind eckiger und bei manchen Farbkombinationen mit einem Schatten versehen.
Im MC-Code ist ein gesetzter Punkt nicht mehr gleich einem gesetzten Punkt auf dem Bildschirm. Vielmehr werden jeweils 2 Punkte nebeneinander zu einem Paar zusammengefaßt. So ergeben sich insgesamt vier Kombinationsmöglichkeiten:

00	=	Farbe 0	Hintergrundfarbregister 0	Adresse 53281
01	=	Farbe 1	Hintergrundfarbregister 1	Adresse 53282
10	=	Farbe 2	Hintergrundfarbregister 2	Adresse 53283
11	=	Farb-RAM	Zeichenfarbe	Adresse 646

Wurde eingangs erwähnt, daß sich im MC-Mode nur noch acht Farben darstellen lassen, so ist das nicht ganz richtig. Es

lassen sich im MC-Mode normale Buchstaben mit acht Farben ausgeben.
Zusätzlich können acht MC-Zeichen ausgegeben werden. Dabei hat nur das Farb-RAM acht verschiedene Farben zur Auswahl, die übrigen Farbregister können nach wie vor auf 16 verschiedene Farben zurückgreifen.
Im Folgenden finden Sie das überarbeitete Listing. Mit diesem Programm wird der DESIGN-Befehl erweitert: Es steht nunmehr auch der Befehl DESIGN-MC zur Verfügung. Mit ihm lassen sich Multi-Color-Zeichen entwerfen. Ein mit DESIGN-MC entworfenes Zeichen sieht dann beispielsweise so aus:

```
300 DESIGN-MC,B                 1=Farb-RAM
310 DATA 1111                   2=Farbregister 2
320 DATA 2..2                   3=Farbregister 1
330 DATA 2..2
340 DATA 3..3
350 DATA 3..3
360 DATA 1111
370 DATA ENDE
```

Hier das erweiterte Listing:

```
1 PRINT"■ BITTE CA. 1 MINUTE WARTEN !":POKE646,6
10 POKE56334,0:POKE1,51
15 FORK=0TO4095
20 POKE57344+K,PEEK(53248+K)
25 NEXT
30 POKE1,55:POKE56334,1
50 POKE53272,PEEK(53272)ANDNOT60R8
60 POKE56576,196:POKE648,3*64+4:PRINT"■"
100 READA$:IFLEFT$(A$,6)<>"DESIGN"THENGOTO100
101 LL=8:MM=7
102 IFA$="DESIGN-MC"THENMC=1:LL=4:MM=3
110 READB$:B$=LEFT$(B$,1)
120 PRINT"■";B$
130 CO=PEEK(50176)
140 FORK=0TO7:READC$
141 IFC$="ENDE"THENPRINT"? MISSING STATEMENTS IN";PEEK(63)+2
56*PEEK(64):END
150 IFLEN(C$)<>LLTHENPRINT"? TOO MANY SIGNS IN";PEEK(63)+256
*PEEK(64):END
```

```
160 FORJ=MMTO0STEP-1:K$=MID$(C$,LL-J,1):IFK$="*"THENBY=BY+2↑
J
161 IFMC=0THENGOTO170
162 IFK$="1"THENBY=BY+2↑(2*J):BY=BY+2↑(2*J+1)
163 IFK$="2"THENBY=BY+2↑(2*J)
164 IFK$="3"THENBY=BY+2↑(2*J+1)
170 NEXTJ:POKE57344+8*CO+K,BY:BY=0:NEXTK
180 READA$:IFLEFT$(A$,6)="DESIGN"THENGOTO101
190 POKE 53270,PEEK(53270)OR16:POKE646,8:PRINT"BBBB":REM MC
ON
200 DATA DESIGN,A
210 DATA********
220 DATA*......*
230 DATA*......*
240 DATA*......*
250 DATA*......*
260 DATA*......*
270 DATA*......*
280 DATA********
290 DATA DESIGN-MC,B
300 DATA          1111
310 DATA          2222
320 DATA          3333
330 DATA          ....
340 DATA          3333
350 DATA          2222
360 DATA          1111
370 DATA          ....
390 DATA ENDE
```

TRANSPARENTE DRUCKEN LEICHT GEMACHT

Das nun folgende Listing ermöglicht das Drucken eines beliebig langen Transparentes.
Das Prinzip ist recht einfach: Der Text des Bandes wird eingegeben. Anschließend wird der Vergrößerungsfaktor gewählt (Breite beliebig, Höhe max. 10).
Nun wird Zeichen für Zeichen des eingegebenen Textes ausgelesen. Die äußere Form wird aus dem ROM gelesen und mit den vorangegangenen Angaben vergrößert.

Letztlich wird das Zeichen gedreht, so daß auf dem Drucker ein endlos langes Transparent entstehen kann.

Doch nun das größtenteils sich selbst erklärende Listing:

```
5 PRINT"■":POKE53280,14:POKE53281,14:POKE646,1
10 REM *** TRANSPARENTDRUCKER ***
15 PRINT"■      TRANSPARENT-DRUCKER       ■■■■"
16 PRINT"DRUCKER EINSCHALTEN !■■■■"
20 OPEN4,4
21 INPUT"BREITE";BR
22 INPUT"HOEHE";HO
23 PRINT"■"
25 POKE214,21:POKE211,VV:SYS58732:POKE204,0
26 GETT$:IFT$=""THEN26
27 IFPEEK(207)THEN 27
28 POKE204,1:IFT$=CHR$(133)THENPRINT"■O.K.":END
29 K$=K$+T$:IFLEN(K$)>39THENK$=RIGHT$(K$,(LEN(K$)-1))
30 POKE214,20:PRINT:PRINTK$:IFVV<40THENVV=VV+1
31 TX=0:IFPEEK(53272)=23THENTX=256
51 POKE646,14:PRINT"■";T$:CO=PEEK(1024):POKE646,1
60 REM *** ZEICHEN AUSLESEN ***
70 POKE56334,0:POKE1,51
80 FORA=0TO7
90 CH(A)=PEEK(53248+(8*CO)+A+(TX*8))
100 NEXTA
110 POKE1,55:POKE56334,1
120 REM *** ZEICHEN DREHEN ***
130 FORA=0TO7:K(A)=0:NEXT
140 FORA=0TO7
150 FORB=7TO0STEP-1:W=2↑B:IFCH(A)>=WTHENCH(A)=CH(A)-W:K(7-B)
=K(7-B)+2↑A
160 NEXTB,A
200 REM *** ZEICHEN AUSGEBEN ***
210 FORI=0TO7:Q$="":FORJ=7TO0STEP-1:WI=K(I)AND2↑J
220 IFWITHENFORU=1TOHO:Q$=Q$+"":NEXTU:GOTO250
230 FORU=1TOHO:Q$=Q$+" ":NEXTU
250 NEXTJ:XY$="":FORU=1TOBR:XY$=XY$+Q$:NEXTU
255 LX=LEN(XY$)-1:IF RIGHT$(XY$,1)=" "THENXY$=LEFT$(XY$,LX):
GOTO255
260 PRINT#4,XY$:NEXTI:GOTO25
270 END
```

MACRO-LAUFSCHRIFT

Das folgende Listing gestattet es, einen in **TX$** gespeicherten Text als Laufschrift auszugeben.
Das Besondere dabei ist die Größe dieser Laufschrift: Die Zeichen sind um den Faktor 8 vergrößert!

```
5 POKE53280,1:POKE53281,1:POKE646,6:PRINT"■"
10 INPUT"TEXT";TX$:TX$=TX$+"      "
20 PRINT"■":FORZ=1TOLEN(TX$)
30 A$=MID$(TX$,Z,1)
40 GOSUB1000:NEXT Z
50 END
60 :
70 :
80 :
90 :
100 :
1000 POKE646,1:PRINT"■";A$:CO=PEEK(1024):POKE646,6
1010 POKE56334,0:POKE1,51
1020 FORJ=0TO7
1030 CO(J)=PEEK(53248+(8*CO)+J)
1040 NEXTJ
1050 POKE1,55:POKE56334,1
1060 FORX=7TO0STEP-1
1070 FORY=0TO7
1080 A=CO(Y)AND2↑X:IFATHENW$(Y)=W$(Y)+"o":GOTO2000
1090 W$(Y)=W$(Y)+" "
2000 NEXTY:P=P+1
4000 REM
4010 PRINT"■";:FORW=0TO7
4020 IFQ>37THENQ=38
4021 PRINTTAB(38-Q)W$(W)
4030 NEXTW:Q=Q+1
4040 IFP<39THEN4100
4050 FORT=0TO7
4060 LE=LEN(W$(T))-1:W$(T)=RIGHT$(W$(T),LE)
4070 NEXTT
4100 NEXTX:RETURN
```

In BASIC ist das Programm natürlich reichlich langsam. Das Listing soll Ihnen jedoch als Anregung dienen und das Prinzip veranschaulichen. Übertragen Sie das Programm doch einmal in Maschinensprache !

8 BLOCKS FÜR SPRITES

Was bei der Programmierung von Sprites unangenehm auffällt, sind die wenigen zur Verfügung stehenden Blocks, die zur Abspeicherung des Äußeren der Sprites erforderlich sind.
Es sind lediglich diese vier Blocks, die für diesen Zweck zu gebrauchen sind:

Block 11	Add. 704 - 766
Block 13	Add. 832 - 894
Block 14	Add. 896 - 958
Block 15	Add. 960 - 1022

Erschwerend kommt ferner hinzu, daß die Blöcke 13-15 im Kassettenpuffer liegen, der während des Programmablaufes im Normalfall allerdings nicht benutzt wird.
Diese vier freien Blöcke gewähren nun auch nur die Definition von vier verschiedenen Sprites.
Um sämtliche acht Sprites mit verschiedenem Äußeren auf dem Bildschirm abbilden zu können, muß also noch Platz für weitere Blöcke geschaffen werden.
Dies erreicht man durch Verlegen des BASIC-Startes. In unserem Beispiel werden so acht Blöcke, 32-39, frei.
Man ist so auch vom Kassettenpuffer unabhängig.

Der BASIC-Start wird von Adresse 2048 nach 2560 verschoben:

```
POKE 44,10      (Adresse 43 des Vektors bleibt unverändert,
                 neue BASIC-Startadresse = 10*256)
POKE 10*256,0   (0 an BASIC-Anfang)
NEW             (Pointer setzen)
```

Wie aus dem NEW-Befehl bereits ersichtlich wird, müssen diese drei Anweisungen vor dem Laden des eigentlichen Programmes eingegeben werden.

6. DAS SPIEL

Wir wollen Ihnen jetzt an dem Spiel ELEVATORBOY einen Weg der Programmierung erklären.
Anfänger können daran lernen, wie man ein längeres Spiel selbst programmiert. Fortgeschrittene können einzelne Tricks übernehmen. Vielleicht entdecken ja auch Sie etwas, daß Sie bei Ihrem nächsten eigenen Spiel verbessern können.
Doch vorher erst eine kurze Spielbeschreibung. Diese Spielbeschreibung soll später in ähnlicher Weise im Spiel erscheinen:
Sie sind Page in einem Hotel und sollen die Schuhe zum Putzen aus den Zimmern holen. Für jedes Paar erhalten Sie 1 DM Trinkgeld vom Gast. Wenn Sie über 50 Schuhe beim Portier abliefern, erhalten Sie ein sehr hohes Extra-Gehalt.
Aber sobald Sie einmal falsch auf den Paternoster gesprungen sind, fallen Sie in den Fahrstuhlschacht und verlieren alle Ihre Schuhe.
Sie bewegen sich mit dem Joystick an Port 2. Hoch und runter hat dabei keine Funktion. Drückt man auf den Feuerknopf, so ändert sich die Fahrtrichtung des Paternosters. Beim ersten Mal ist es sehr schwer, richtig auf den Fahrstuhl zu kommen. Sie müssen dabei im richtigen Moment den Joystick in die entsprechende Richtung drücken.
Währenddessen läuft an der Seite ein Bonus in Form eines langen, senkrechten Streifens ab. Bevor der Bonus ganz abgelaufen ist, müssen Sie zum Portier. Danach erscheint ein neues Bild, und Sie erhalten einen neuen Bonus.
Wenn Sie nicht vor dem Ablaufen des Bonus den Portier erreichen, wird Ihnen ein Leben abgezogen.
Zuerst einmal stellen wir ein "Grundgerüst" des Spieles auf. Dieses Gerüst ergänzen wir nach und nach, bis schließlich das fertige Spiel entstanden ist.

Im ersten Listing ist die Grafik sehr schlecht. Besser wird sie erst, wenn wir in dem nächsten Schritt die Zeichen neu definieren.
Da Sie wahrscheinlich aus den Zeichen jetzt noch nicht schlau werden, sagen wir Ihnen, was sie später einmal darstellen werden:

ZEICHEN	BEDEUTUNG
/Pound/	Zimmer mit einem Paar Schuhe
↑	Geschlossene Tür
#	Spieler (SIE)
%	Portier
"	Paternoster

Und noch eine Tabelle:
Die Reihenfolge der Ergänzungen. Diese Reihenfolge ist auch eine Skala für die Wichtigkeit der einzelnen Punkte.

(0. "Gerippe")
1. Grafik
2. Sound
3. Anleitung
4. Anfangsbild

DAS GERIPPE

Das Gerippe soll nur das Wichtigste enthalten. Also müssen wir genug Zeilen als Zwischenraum lassen, um später noch Grafik, usw. mit hineinzubringen.
Wir fangen auch erst bei Zeile 100 an, und lassen so Platz für das Anfangsbild. Zum vernünftigen Spielen ist dieses Listing noch nicht geeignet. Sie sollten es trotzdem einmal

starten, um zu sehen, wie das Spiel sich später entwickelt, und um eine ungefähre Vorstellung des Ganzen zu bekommen. Das komplette Listing ist am Ende des Kapitels noch einmal in "einem Stück" abgedruckt.
Doch zuerst das Listing des Gerüstes:

```
6 PRINT"■□":POKE53281,3
7 BBB=1024:R=54272
9 X=BBB+536:D=BBB+487:G=40:BO=23:PUN=0
10 FE=10:LI=3
11 PRINT"■■■■■■■■■■■■                    ELEVATOR BOY"
12 IFPEEK(56320)<>111 THEN12
100 BILD$(1)=" !↑↑£££ ↑↑£££ ↑↑££↑ £££££ £££££!"
110 BILD$(2)=" !↑↑↑↑£ £££££ £↑£££ ££££↑ £££££!"
120 BILD$(3)=" !↑££££ ↑↑↑££ £££££ £££££ ££££↑!"
130 BILD$(4)=" !£££££ £££££ £↑£££ ££↑↑£ ££↑↑↑!"
140 BILD$(5)=" !↑££££ £££££ £££££ £££££ ↑↑↑↑↑!"
150 BILD$(6)=" !£££££ ££↑↑↑ ££££↑ ££££↑ ££££↑!"
160 BILD$(7)=" !£££££ £↑↑££ ££↑££ ££££↑ £££↑↑!"
170 BILD$(8)=" !£££££ ↑↑£££ £££££ ££↑££ ££↑↑↑!"
180 BILD$(9)=" !£££££ ↑↑£££ £££££ ↑↑↑↑£ £££££!"
183 PRINT"□";
184 PRINT" ■←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
185 PRINT" ■←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
187 PRINT" ■!■←←←←← ←←←←← ←←←←← ←←←←← ←←←←←■!"
190 FORI=1TO9
195 W=INT(RND(1)*8)+1
200 PRINTBILD$(W)
205 PRINT" ■!■←←←←← ←←←←← ←←←←← ←←←←← ←←←←←■!"
210 NEXT
215 PRINT"■ !               %               !"
217 PRINT"■ !■←←←←← ←←←←← ←←←←← ←←←←← ←←←←←■!"
220 PRINT"■ ←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
221 FORI=BBB+77TOBBB+917STEP40:POKEI,38:POKEI+R,6:NEXT
230 POKEX,35:POKEX+R,0:POKEX+1,32:POKEX+2,32:POKEX-1,32:POKE
X-2,32
270 PRINT"             HI-SCORE ";HIS;
280 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
1000 GOSUB3000
1003 FORI=1TOFE:NEXT
1005 POKED,32:POKED+6,32:POKED+12,32:POKED+18,32
1010 IFPEEK(D+G)=31THENG=-G
1020 D=D+G
1030 POKED,34:POKED+6,34:POKED+12,34:POKED+18,34
1040 IFOL=0THEN2000
1050 POKEX,32:X=X+G
1060 IFPEEK(X)=31THENX=X+40:GOTO9000
1070 POKEX,35:POKEX+R,0
2000 BO=BO-0.025
2010 POKEINT(BO)*40+BBB+37,32:IFBO=<1THEN9000
2990 GOTO1000
```

```
3000 IFPEEK(56320)=127THENH=0:RETURN
3010 IFPEEK(56320)=<111THEN3500
3020 IFPEEK(56320)=123THENY=-1:GOTO3050
3030 IFPEEK(56320)=119THENY=1:GOTO3050
3040 RETURN
3050 IFPEEK(X+Y)=31THENY=Y-40:IFPEEK(X+Y)=31THEN9000
3053 IFPEEK(X+Y)=28THENPUN=PUN+1:GOTO3100
3054 IFPEEK(X+Y)=33THENRETURN
3055 IFPEEK(X+Y)=30THENRETURN
3057 IFPEEK(X+Y)=34ORPEEK(X+Y+40)=34THEN3900
3060 IFPEEK(X+Y)=37THEN3800
3065 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=32THEN:POKEX,32:X=X+Y:GOT
O9000
3070 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=102THEN3100
3100 POKEX,32
3110 X=X+Y:OL=0
3120 POKEX,35:POKEX+R,0
3130 PRINT"■ ←PUNKTE←";PUN+TR;:PRINT"←←SCHUHE←";PUN
3190 RETURN
3500 IFH=0THENG=-G:H=1
3510 RETURN
3800 FE=FE-1:IFPUN<50THENTR=TR+PUN:GOTO3810
3805 TR=TR+INT(PUN*(BO*7))
3810 X=BBB+536:D=BBB+487:G=40:BO=23:PUN=0
3830 GOTO183
3900 POKEX,32
3905 X=X+Y
3910 IFPEEK(X)=34THENX=X-40:IFPEEK(X)=34THEN9000
3920 OL=1
3930 POKEX,35:POKEX+R,0
3940 RETURN
9000 TR=TR+PUN:LI=LI-1
9010 IFBO=<1THENGOTO9020
9015 POKEX,32:X=X+40:IFPEEK(X+40)=31THEN9100
9017 POKEX,35:POKEX+R,0:GOTO9015
9020 POKEX,32
9030 BO=23:FORI=1TO23:POKEI*40+BBB+37,160:NEXT:OL=0
9035 D=BBB+487:G=40:BO=23:PUN=0
9036 IFLI=0THEN9800
9040 X=BBB+536:POKEX,35:POKEX+R,0:GOTO183
9100 POKEX,39:POKEX+R,0
9190 GOTO9020
9800 PRINT"■";:POKE53281,1
9830 PRINT"■■■■■■■■■■■■■■          DU HAST ";TR+PUN;"$ VERDIEN
T
9840 IFTR+PUN>HISTHENHIS=TR+PUN
9850 TR=0:PUN=0
9880 GETA$:IFA$<>""ORPEEK(56320)=111THEN6
9890 GOTO9880
```

Es stimmt nicht ganz, daß das Listing erst in Zeile 100 anfängt. In den ersten Zeilen stehen der Name und die Definition der Variablen. Aber dazwischen ist noch sehr viel Platz für das Anfangsbild. In jedem Programm sollten die Variablen in den vorderen Zeilen zum ersten Mal definiert werden, damit man schnell eine Übersicht über sie bekommen kann.
Tabelle der wichtigsten Variablen (nach ihrer Wichtigkeit geordnet) :

VARIABLE	BEDEUTUNG
X	Adresse, in der sich der Spieler befindet. Am Anfang enthält X den Wert 1560. Das ist etwa in der Mitte des Bildschirm-Speichers.
BBB	Adresse des Bildschirmspeicher-Anfanges. Diese Variable muß verändert werden, sobald der Zeichensatz geändert wird. Die Variable dient hier als Zeiger und ist eine Konstante, die sich nicht verändert.
R	R ist ähnlich der Variablen BBB. Sie enthält die Differenz zwischen Bildschirmspeicher-Anfang und Farbspeicher-Anfang. Diese Variable (Konstante) muß ebenfalls bei Anwendung der neuen Grafik-Zeichen verändert werden. Außer dem vereinfacht sie die Veränderung der Farbe unter einem Zeichen auf dem Bildschirm. Will man beispielsweise die Farbe unter der Spielfigur ändern, so braucht man lediglich einzugeben: POKE X+R,(Farbe des Spielers).
D	Adresse des ersten Paternosters. Die weiteren Paternoster werden mit Hilfe von D errechnet. Der zweite Paternoster ist bei D+6,der dritte bei D+12, und der vierte bei D+18. So braucht nur eine Variable erhöht zu werden und, Programmzeilen werden gespart.

BONUS Die Variable für den Bonus. In langen Spielen sollte man die Variablen-Namen so wählen, daß man sie später wiedererkennt, d.h., der Name der Variablen sollte die Bedeutung erkennen lassen,die sie darstellt.Dafür kann der Name der Variablen ruhig etwas länger sein.

Y Y ist eine Hilfs-Variable für die Koordinate des Spielers.Sie wird beispielsweise benutzt, um zu prüfen, ob das nächste Feld frei ist.

TR TR ist das erreichte Gehalt.

PUN PUN ist die Anzahl der Schuhe, die man bei sich trägt. Nach einem Fall in den Paternoster-Schacht wird diese Variable auf Null gesetzt.

LI LI stellt die Anzahl der noch vorhanden Leben dar. Bei jedem Sturz oder bei Ablauf des Bonus wird diese Variable um Eins erniedrigt. Nach Verringern dieser Variable wird überprüft, ob sie kleiner oder gleich Null ist.

FE FE ist die Arbeitsgeschwindigkeit. FE beträgt anfangs Zehn. Die Arbeitsgeschwindigleit kann kaum mehr erhöht werden, aber als Anfänger bei diesem Spiel können Sie FE erhöhen, um damit die Arbeitsgeschwindigkeit zu erniedrigen, und das Spiel zu vereinfachen.Sobald Sie beim Portier waren, erniedrigt sich FE um 5.

G G ist die Richtung für die Paternoster. Sie enthält entweder 40 für runter oder -40 für rauf. Zur Variable D für den Fahrstuhl wird immer G hinzugezählt. Daraus ergibt sich die Bewegung der Paternoster.

I — I ist eine Schleifenvariable.Es ist praktisch in verschiedenen Programmen immer die gleiche Variable für Schleifen zu benutzen.
Man spart Speicherplatz, wenn man im einem Programm immer die gleiche Schleifen-Variable benutzt. Aber passen Sie auf,daß eine Schleife schon abgeschlossen ist, bevor die gleiche Variable noch einmal verwendet wird.

HIS — HIS ist die Variable für den High-Score, die bisher höchste, erreichte Punktzahl. Obwohl der High-Score eigentlich nicht wichtig ist, wurde er doch schon (wegen der Kürze der Berechnung) in das Gerippe eingebaut.

H — Die Variable H soll verhindern, daß auf einnen Knopfdruck die Fahrtrichtung mehrmals gewechselt wird. Bei Knopfdruck wird die Fahrtrichtung geändert und H auf 1 gesetzt. H wird erst wieder auf 0 gesetzt, wenn der Joystick in Ruhe-Phase ist. Wenn H=1 ist, reagiert der Computer nicht auf den Knopfdruck. Auf die gleiche Weise kann man wie z.B. bei dem Spiel "FROGGER" die Länge der Bewegung einschränken.

OL — Wenn OL den Wert 1 hat, befindet man sich auf dem Fahrstuhl. Sonst ist OL gleich Null.

W — Zufallszahl für die Erstellung verschiedener Bilder.

BILD$(W) — Die Variable BILD$(W) kann neun verschiedene Variablen-Zuweisungen (Bilder) enthalten.Die-Variable ist dimensioniert und wählt so jedesmal ein anderes Gesamtbild aus. Je mehr verschiedene Bilder ein Spiel hat, um so interessanter ist das Programm.

Wichtig für die Übersicht über ein langes Programm ist nicht nur die Variablen-Liste, sondern auch eine Auflistung der einzelnen Programmschritte.

Zeilennummer	Aufgabe
0 - 5	Hier wird später der Zeichensatz eingelesen und aktiviert (siehe GRAPHIK).
6 - 99	Dieser Bereich ist für die Variablen-Definition vorgesehen. Außerdem liegt hier das einfache Titelbild. Später wird hier auch das verbesserte Anfangsbild zu finden sein.
100 - 180	Definition der Variable BILD$(w).
183 - 280	Hier wird das Gesamtbild erstellt.
1000 - 2990	Die Bewegung des Paternosters wird geregelt.
1050 - 1070	Das ist der Bereich für die Bewegung der Spielfigur auf dem Fahrstuhl.Diese Bewegung ist gesondert von der normalen Bewegung des Spielers.
3000 - 3190	Jetzt kommt die Lenkung der Spielfigur. Im Gegensatz zu der Bewegungsroutine im Bereich von 1050 - 1070 wird die Bewegung in dieser Routine durch den Joystick beeinflußt.
3500 - 3510	An dieser Stelle wird die Fahrtrichtung des Paternosters geändert.
3800 - 3830	Diese Zeilen werden vom Programm angesprungen, wenn die Spielfigur beim Portier

ist.

3900 - 3940 Hier steht wieder etwas für die Bewegung der Spielfigur.

9000 - 9190 Dieser Teil berücksichtigt den Sturz des in den Fahrstuhlschacht.

9800 - 9890 Hier ist die Endroutine untergebracht. Die Endroutine wird dann angesprungen, wenn man kein Leben mehr hat.

50000 - Ende Dieser Bereich ist für die DATA-Zeilen vorgesehen. Diese Daten dienen der Graphik und werden auch indem Abschnitt erst eingegeben.

Die Zeilen für die Soundeffekte sind nicht in dieser Übersicht aufgeführt. Sie werden mehreren dieser Punkte hinzugefügt. Bei dieser Aufzählung der einzelnen Programmschritte wird deutlich, wie groß die Abstände der einzelnen Bereiche sind. Man kann diese Zeilenabstände nachträglich mit einem guten RENUMBER-Programm herausnehmen.

GRAPHIK

Die erste Erweiterung des einfachen Listings ist der Abschnitt über die Graphik. Er ist wegen der vielen neuen Definitionen der Zeichen und der daraus folgenden vielen Daten die längste Ergänzung. Aber dafür werden alle Zeichen bis einschließlich zur neun, also insgesamt 58 Zeichen, geändert. Sie erhalten ein völlig neues Schriftbild. Mit der neuen Graphik wird das Spiel gleich viel besser.
Ergänzen oder ändern Sie bitte folgende Zeilen ab:

```
1 READA:IFA=-1THEN3
2 POKE57344+I,A:I=I+1:GOTO1
3 POKE53272,24:POKE56576,148:POKE648,196
4 GOTO6
5 POKE53272,21:POKE56576,151:POKE648,4:END
6 PRINT"■":POKE53281,3
7 BBB=50176:R=5120
50000 DATA127,127,81,22,20,22,56,0
50001 DATA60,60,102,102,126,126,102,0:A
50002 DATA120,124,102,124,126,103,126,0:B
50003 DATA60,126,102,96,102,126,60,0:C
50004 DATA124,126,99,99,99,126,124,0:D
50005 DATA127,126,96,124,96,126,127,0:E
50006 DATA127,126,124,96,124,120,112,0:F
50007 DATA62,121,112,115,115,63,31,0:G
50008 DATA115,115,115,127,127,115,115,0:H
50009 DATA28,28,28,28,28,28,28,0:I
50010 DATA127,127,7,7,103,63,31,0:J
50011 DATA115,119,126,126,126,119,115,0:K
50012 DATA112,112,112,112,112,126,127,0:L
50013 DATA119,127,107,107,107,99,119,0:M
50014 DATA103,115,123,127,127,119,115,0:N
50015 DATA62,127,119,119,119,127,62,0:O
50016 DATA126,127,115,127,126,112,112,0:P
50017 DATA62,127,99,111,111,127,62,0:Q
50018 DATA126,127,115,127,124,126,119,0:R
50019 DATA63,126,96,62,3,63,126,0:S
50020 DATA127,127,93,28,28,28,28,0:T
50021 DATA119,119,119,119,127,127,28,0:U
50022 DATA119,119,50,58,58,28,8,0:V
50023 DATA119,99,107,107,107,127,54,0:W
50024 DATA99,119,54,62,62,119,99,0:X
50025 DATA119,59,31,14,14,14,14,0:Y
50026 DATA127,63,7,28,112,126,127,0:Z
50027 DATA240,240,192,192,192,192,240,240:[
50028 DATA0,36,102,102,102,102,36,0:£
50029 DATA15,15,3,3,3,3,15,15:]
50030 DATA126,126,126,78,62,126,126,126:↑
```

```
50031 DATA239,239,239,0,251,251,251,0:←
50032 DATA0,0,0,0,0,0,0,0
50033 DATA255,0,255,0,255,0,255,0:!
50034 DATA255,255,255,189,60,60,60,60:"
50035 DATA0,60,60,24,255,24,60,36:#
50036 DATA101,85,87,87,85,85,85,101:$
50037 DATA24,24,24,60,60,126,255,48:%
50038 DATA127,254,127,254,127,254,127,254:&
50039 DATA0,24,24,126,126,24,24,24:'
50040 DATA24,48,96,96,96,96,48,24:(
50041 DATA24,12,6,6,6,6,12,24:)
50042 DATA48,72,72,16,16,0,16,0:*
50043 DATA0,60,60,126,126,60,60,0:+
50044 DATA0,0,0,0,0,12,24,48:,
50045 DATA0,0,0,126,126,0,0,0:-
50046 DATA0,0,0,0,0,96,96,0:.
50047 DATA0,6,12,24,24,48,96,0:/
50048 DATA0,24,60,102,102,60,24,0:0
50049 DATA0,12,28,60,12,12,12,0:1
50050 DATA0,12,18,4,8,16,30,0:2
50051 DATA0,30,2,4,2,18,30,0:3
50052 DATA0,16,16,16,20,30,4,0:4
50053 DATA0,30,16,28,2,2,28,0:5
50054 DATA0,12,18,16,28,18,12,0:6
50055 DATA0,30,18,4,8,16,16,0:7
50056 DATA0,12,18,12,18,18,12,0:8
50057 DATA0,24,36,28,4,36,24,0:9
50058 DATA-1
```

Wenn Sie jetzt das Programm starten, müssen Sie erst etwas warten, bis der Computer alle Daten eingelesen hat.
Für das Spiel werden nicht alle 59 Buchstaben gebraucht. Wir haben diese DATAs einfachheitshalber trotzdem abgedruckt. So können Sie den Zeichensatz auch für Ihre eigenen Programme benutzen.
Im Listing erscheint hinter jeder DATA-Zeile ein Zeichen. In dieser Zeile sind dann die 8 Daten des neuen Zeichens enthalten.
Die Graphik in dem Spiel "Elevator Boy" besteht nur aus den Bildschirmcode-Zeichen.
Eine weitere gute Möglichkeit, eine effektvolle Graphik zu erstellen, ist Sprites zu benutzen. Da aber in diesem Spiel keine Sprites benutzt wurden, soll in diesem Kapitel auch nichts über Spritegraphik stehen. Das gleiche gilt auch für die High-Resolution-Graphik.
Haben Sie das Spiel schon gestartet, wollen aber wieder in die normale Graphik zurück, brauchen Sie nur

```
RUN 5
```

einzugeben. Das erspart die vielen POKE-Befehle. Wenn Sie versehentlich RUNSTOP/RESTORE gedrückt haben, können Sie ebenso verfahren, müssen aber außerdem danach den Bildschirm löschen.
Wollen Sie das Programm danach wieder mit den neuen Graphik-Symbolen spielen, geben Sie bitte

```
RUN 3
```

ein. Jetzt wird der Zeichensatz nicht noch ein zweites Mal eingelesen. Dadurch entfällt die lange Wartezeit vor dem Spiel.
Man kann in Spielen auch besondere Graphikeffekte einbauen. Z.B. ist beim Sturz in den Fahrstuhlschacht ein Kreuz unten

im Schacht zu sehen. Zusätzlich wird dieser Effekt später mit einem Sound-Effekt ergänzt.
Bei der Farbe der Figuren, des Hinter- und Vordergrundes sollten Sie Ihre Phantasie spielen lassen und gute Farbkombinationen auswählen. Zu beachten wäre nur, daß die Farben auch auf einem Schwarz/Weiß-Fernseher zu unterscheiden sein müssen. Rot und blau sind beispielsweise auf einem Schwarz/Weiß-Fernseher nicht zu differenzieren.

SOUND

Wenn Sie die Graphik erfolgreich ergänzt haben, können Sie jetzt zum nächsten Abschnitt kommen - zu den Sound-Effekten. Immer dann, wenn etwas Besonderes passiert, kann man einen Sound-Effekt einbauen. Unter Sound-Effekten versteht man Melodien oder auch einzelne Töne. Wenn Sie gelegentlich einmal in einer Spielhalle waren, ist Ihnen diese Untermalung sicherlich bekannt. Auch in dem zum C-64 beigelegten Handbuch sind einige kleine Sound-Routinen abgedruckt.
Beim Spiel "Elevator Boy" soll immer dann ein Ton erklingen, wenn der Spieler ein Paar Schuhe nimmt, und wenn er in den Fahrstuhlschacht fällt. Dafür ergänzen Sie bitte wieder Ihr Programm um die nachfolgenden Zeilen:

```
250 POKE 54296,15
252 POKE 54277,31
254 POKE 54278,128
256 POKE 54272,1
260 POKE 54273,1

3053 IF PEEK(X+Y)=28 THEN POKE 54276,129
     :PUN=PUN+1:GOTO3100
3180 POKE 54276,0

9110 POKE 54277,31
```

```
9112 POKE 54278,255
9114 POKE 54272,0
9116 POKE 54273,2
9118 POKE 54276,33
9120 FOR I=1 TO 500:NEXT I:POKE 54276,0
```

In den Zeilen 250 bis 260 setzt der Computer die Werte, die für die Tonerzeugung wichtig sind. Wenn der Spieler auf ein Paar Schuhe geht, braucht nur noch die Wellenform ein- und wieder ausgeschaltet zu werden (Zeile 3053 und 3180).
Die Zeilen 9110 bis 9120 sind für den Sturz in den Fahrstuhlschacht. Damit hier ein anderer Ton zu hören ist, werden neue Werte für die Tonerzeugung festgesetzt.
Man kann zusätzlich zu den Sound-Effekten auch noch eine Melodie ertönen lassen. In dem Kapitel über Interrupt-Handling steht ein dafür geeignetes Programm. Während die Töne der Sound-Effekte über die erste der drei Stimmen zu hören ist, läuft in diesem Programm eine Melodie über die dritte Stimme. Am besten lesen Sie sich die Beschreibung zu dem Programm einmal kurz durch. Es heißt "Musik aus dem Interrupt".

DIE ANLEITUNG

Der Nutzen einer Anleitung ist offensichtlich. Der Spieler sieht sofort, worum es in dem Spiel geht. Häufig werden die Anleitungen aber auch in Form einer Beschreibung mitgeliefert. Auf diese Weise kann man seine Programme auch ein wenig schützen.
Unser Spiel ist selbsterklärend. Dazu fügen Sie bitte folgende Zeilen ein.

```
25 PRINT " !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!"
26 PRINT " WOLLEN SIE ANWEISUNGEN (J/N)
* (viermal Cursor hoch) " : GET G$
```

```
28 IF G$="N" OR PEEK(56320)=111 THEN 100
29 IF G$="J" THEN 42
38 GOTO 28
42 PRINT "(CLR-HOME)SIE SIND PAGE IN EIN
EM GROSSEN HOTEL UND";
43 PRINT "SOLLEN DIE SCHUHE AUS DEN ZIMM
ERN HOLEN.";
44 PRINT "FUER JEDES PAAR ERHALTEN SIE 1
$"
45 PRINT "TRINKGELD VOM GAST."
46 PRINT "WENN SIE UEBER 50 SCHUHE BEIM
PORTIER"
47 PRINT "ABLIEFERN, ERHALTEN SIE EIN HO
HES GEHALT."
49 PRINT : PRINT "  (Pound)  ZIMMER MIT
 EINEM PAAR SCHUHE"
50 PRINT : PRINT "  ( ↑ )     GESCHLOSSE
NE TUER"
51 PRINT : PRINT "  ( # )     Spielfigur
"
52 Print : PRINT "  ( % )     Portier"
53 PRINT : PRINT "  ("; CHR$(34) ;")
Fahrstuhl"
56 IF PEEK (56320) <> 111 THEN56
```

Außerdem müssen Sie Zeile zwölf löschen.

Die Bedienung des Programms sollte möglichst unkompliziert sein. Sie muß für den Spieler einfach und schnell zu handhaben sein. Ein Spieler, der gemütlich auf dem Sofa liegt und einen Joystick in der Hand hält, möchte nicht nach jeder Spielrunde aufstehen und das Spiel neu einstellen müssen.
Darum braucht in unserem Spiel die Frage auf Anweisungen nicht mit "n" beantwortet zu werden. Es genügt, wenn man in diesem Fall auf den Feuer-Knopf am Joystick drückt.

ANFANGSBILD

Das Anfangsbild sollte den Spieler schon ein wenig animieren. Dies erreicht man mit einer imposanten oder zumindest guten Graphik im Anfangsbild. Ein Vorschlag unsererseits ist, in das Anfangsbild eine Szene aus dem Spiel einzubauen. Ein weiterer Vorschlag ist, daß die Graphik sich bewegen könnte. Beide Vorschläge wurden beim Schreiben des Spiels berücksichtigt.
Ein Teil aus dem Anfangsbild wollen wir Ihnen ersteinmal so vorstellen. Es ist eine in BASIC geschriebene Laufschrift.

```
11 EL$ = " ELEVATOR BOY
        (37 Leerzeichen)"

27 PRINT "(1 mal Cursor hoch)"EL$

30 FOR O = O TO 150 : NEXT
31 EL$ = RIGHT$ (EL$,1) + LEFT$ (EL$,
LEN (EL$) -1 )

39 GOTO 27
```

Diese Zeilen können Sie ebenfalls in das Spiel einfügen, sie gehören ins Anfangsbild. Aber diese paar Zeilen sind gleichzeitig ein kleines, eigenständiges Programm. Sie sollten es einmal allein ausprobieren. Wenn Sie das kleine Programm gestartet haben, sehen Sie den Schriftzug "ELEVATOR BOY" über den Bildschirm laufen. Statt des Textes, können Sie ihren eigenen Text in EL$ eingeben. In Zeile 30 können Sie die Arbeitsgeschwindigkeit einstellen. Wenn Sie die Zeile 30 ganz weglassen oder die Geschwindigkeit auf O setzen, läuft die Schrift so schnell, daß sie verschwimmt. Die Laufrichtung können Sie ändern, indem Sie LEFT$ und RIGHT$ vertauschen.

Jetzt kommen weitere Zeilen, die Sie in das Gesamtlisting einfügen können. Durch diese Zeilen enthält das Spiel ein bewegtes Anfangsbild.

```
12 PRINT:PRINTTAB(14)"!←←←←← ←←←←←!"
13 PRINTTAB(14)"!↑↑↑££ £££££!"
14 PRINTTAB(14)"!←←←←← ←←←←←!"
15 PRINTTAB(14)"!↑££££ £↑↑↑↑!"
16 PRINTTAB(14)"!←←←←← ←←←←←!"
17 PRINTTAB(14)"!£££££ £££££!"
18 PRINTTAB(14)"!←←←←← ←←←←←!"
19 PRINTTAB(14)"!↑↑£££ ££££↑!"
20 PRINTTAB(14)"!←←←←← ←←←←←!"
21 PRINTTAB(14)"!↑↑↑↑£ £££↑↑!"
22 PRINTTAB(14)"!←←←←← ←←←←←!"
23 PRINT:PRINTTAB(14)"!←←←←← ←←←←←!"
32 PL=BBB+100+M
33 POKEPL-40,32
34 POKEPL,35:POKEPL+R,0
35 POKEPL+40,34:POKEPL+R+40,2
36 POKEPL+80,0
37 M=M+G:IFM>400ORM<=0THENG=-G
```

ZUSAMMENFASSUNG

Jetzt ist das Programm vollständig. Auch wenn Sie nicht mitverfolgt haben, wie das Spiel entstanden ist, können Sie das alles umfassende Listing abtippen und spielen.
Dieses abschließende Listing soll außerdem denen, die die einzelnen Schritte abgeschrieben haben, helfen, vielleicht vorhandene Fehler zu finden und korrigieren.
Hier das Gesamtlisting:

```
1 READA:IFA=-1THEN3
2 POKE57344+I,A:I=I+1:GOTO1
3 POKE53272,24:POKE56576,148:POKE648,196
4 GOTO6
5 POKE53272,21:POKE56576,151:POKE648,4:END
6 PRINT"▮▯":POKE53281,3
7 BBB=50176:R=5120
9 X=BBB+536:D=BBB+487:G=40:BO=23:PUN=0
10 FE=10:LI=3
11 EL$=" ELEVATOR BOY                        "
12 PRINT:PRINTTAB(14)"!←←←←← ←←←←←!"
13 PRINTTAB(14)"!↑↑↑££ £££££!"
14 PRINTTAB(14)"!←←←←← ←←←←←!"
15 PRINTTAB(14)"!↑££££ £↑↑↑↑!"
16 PRINTTAB(14)"!←←←←← ←←←←←!"
17 PRINTTAB(14)"!£££££ £££££!"
18 PRINTTAB(14)"!←←←←← ←←←←←!"
19 PRINTTAB(14)"!↑↑£££ £££↑!"
20 PRINTTAB(14)"!←←←←← ←←←←←!"
21 PRINTTAB(14)"!↑↑↑↑£ £££↑↑!"
22 PRINTTAB(14)"!←←←←← ←←←←←!"
23 PRINT:PRINTTAB(14)"!←←←←← ←←←←←!"
25 PRINT:PRINT" !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
26 PRINT" WILLST DU ANWEISUNGEN (J/N)*▯▯▯▯"
27 PRINT"▯ "EL$
28 GETG$:IFG$="N"ORPEEK(56320)=111THEN100
29 IFG$="J"THEN42
30 FORO=0TO150:NEXT
31 EL$=RIGHT$(EL$,1)+LEFT$(EL$,LEN(EL$)-1)
32 PL=BBB+100+M
33 POKEPL-40,32
34 POKEPL,35:POKEPL+R,0
35 POKEPL+40,34:POKEPL+R+40,2
36 POKEPL+80,32
37 M=M+G:IFM>400ORM<=0THENG=-G
39 GOTO27
```

```
42 PRINT"■SIE SIND PAGE IN EINEM GROSSEN HOTEL UND";
43 PRINT"SOLLEN SCHUHE AUSDEN ZIMMERN HOLEN."
44 PRINT"FUER JEDES PAAR SIE ERHALTEN SIE 1 $"
45 PRINT"TRINKGELD VOM GAST."
46 PRINT"WENN SIE UEBER 50 SCHUHE BEIM PORTIER"
47 PRINT"ABLIEFERN,BEKOMMEN SIE EIN EXTRA GEHALT.";
49 PRINT"■   [£] ZIMMER MIT EINEM PAAR SCHUHE"
50 PRINT"■■   [↑] GESCHLOSSENE TUER"
51 PRINT"■■   [ ] LEERES ZIMMER"
52 PRINT"■■   [#] SPIELFIGUR"
53 PRINT"■■   [%] PORTIER"
54 PRINT"■■   ["CHR$(34)"] FAHRSTUHL"
56 IFPEEK(56320)<>111THEN56
100 BILD$(1)=" !↑↑£££ ↑↑£££ ↑↑£££ ££££££ £££££!"
110 BILD$(2)=" !↑↑↑↑£ £££££ £↑£££ £££££ £££££!"
120 BILD$(3)=" !↑££££ ↑↑↑££ £££££ £££££ ££££↑!"
130 BILD$(4)=" !£££££ £££££ £↑£££ ££↑↑£ ££↑↑↑!"
140 BILD$(5)=" !↑££££ £££££ £££££ £££££ ↑↑↑↑↑!"
150 BILD$(6)=" !£££££ ££↑↑↑ ££££↑ ££££↑ ££££↑!"
160 BILD$(7)=" !£££££ £↑↑££ ££↑££ £££££ £££↑↑!"
170 BILD$(8)=" !£££££ ↑↑£££ £££££ ££↑££ ££↑↑↑!"
180 BILD$(9)=" !£££££ ↑↑£££ £££££ ↑↑↑↑£ £££££!"
183 PRINT"■";
184 PRINT" ■<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
185 PRINT" ■<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
187 PRINT" ■!■<<<<< <<<<< <<<<< <<<<< <<<<<■!"
190 FORI=1TO9
195 W=INT(RND(1)*8)+1
200 PRINTBILD$(W)
205 PRINT" ■!■<<<<< <<<<< <<<<< <<<<< <<<<<■!"
210 NEXT
215 PRINT"■ !                %               !"
217 PRINT"■ !■<<<<< <<<<< <<<<< <<<<< <<<<<■!"
220 PRINT"■ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
221 FORI=BBB+77TOBBB+917STEP40:POKEI,38:POKEI+R,6:NEXT
230 POKEX,35:POKEX+R,0:POKEX+1,32:POKEX+2,32:POKEX-1,32:POKE
X-2,32
250 POKE54296,15
260 POKE54277,31:POKE54278,128:POKE54272,1:POKE54273,1
270 PRINT"          HI-SCORE ";HIS;
280 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
1000 GOSUB3000
1003 FORI=1TOFE:NEXT
1005 POKED,32:POKED+6,32:POKED+12,32:POKED+18,32
1010 IFPEEK(D+G)=31THENG=-G
1020 D=D+G
1030 POKED,34:POKED+6,34:POKED+12,34:POKED+18,34
1040 IFOL=0THEN2000
1050 POKEX,32:X=X+G
1060 IFPEEK(X)=31THENX=X+40:GOTO9000
1070 POKEX,35:POKEX+R,0
```

```
2000 BO=BO-0.05
2010 POKEINT(BO)*40+BBB+37,32:IFBO=<1THEN9000
2990 GOTO1000
3000 IFPEEK(56320)=127THENH=0:RETURN
3010 IFPEEK(56320)=<111THEN3500
3020 IFPEEK(56320)=123THENY=-1:GOTO3050
3030 IFPEEK(56320)=119THENY=1:GOTO3050
3040 RETURN
3050 IFPEEK(X+Y)=31THENY=Y-40:IFPEEK(X+Y)=31THEN9000
3053 :IFPEEK(X+Y)=28THENPOKE54276,129:PUN=PUN+1:GOTO3100
3054 IFPEEK(X+Y)=33THENRETURN
3055 IFPEEK(X+Y)=30THENRETURN
3057 IFPEEK(X+Y)=34ORPEEK(X+Y+40)=34THEN3900
3060 IFPEEK(X+Y)=37THEN3800
3065 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=32THEN:POKEX,32:X=X+Y:GOT
O9000
3070 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=102THEN3100
3100 POKEX,32
3110 X=X+Y:OL=0
3120 POKEX,35:POKEX+R,0
3130 PRINT"■ ←PUNKTE←";PUN+TR;:PRINT"←←SCHUHE←";PUN
3180 POKE54276,0
3190 RETURN
3500 IFH=0THENG=-G:H=1
3510 RETURN
3800 FE=FE-1:IFPUN<50THENTR=TR+PUN:GOTO3810
3805 TR=TR+INT(PUN*(BO*7))
3810 X=BBB+536:D=BBB+487:G=40:BO=23:PUN=0
3830 GOTO183
3900 POKEX,32
3905 X=X+Y
3910 IFPEEK(X)=34THENX=X-40:IFPEEK(X)=34THEN9000
3920 OL=1
3930 POKEX,35:POKEX+R,0
3940 RETURN
9000 TR=TR+PUN:LI=LI-1
9010 IFBO=<1THENGOTO9020
9015 POKEX,32:X=X+40:IFPEEK(X+40)=31THEN9100
9017 POKEX,35:POKEX+R,0:GOTO9015
9020 POKEX,32
9030 BO=23:FORI=1TO23:POKEI*40+BBB+37,32:NEXT:OL=0
9035 D=BBB+487:G=40:BO=23:PUN=0
9036 IFLI=0THEN9800
9040 X=BBB+536:POKEX,35:POKEX+R,0:GOTO183
9100 POKEX,39:POKEX+R,0
9110 POKE54277,31:POKE54278,255:POKE54272,0:POKE54273,2:POKE
54276,33
9120 FORI=1TO500:NEXT:POKE54276,0
9190 GOTO9020
9800 PRINT"■";:POKE53281,1
9830 PRINT"■■■■■■■■■■■■■■■■          DU HAST ";TR+PUN;"$ VERDIEN
T
```

```
9840 IFTR+PUN>HISTHENHIS=TR+PUN
9850 TR=0:PUN=0
9880 GETA$:IFA$<>""ORPEEK(56320)=111THEN6
9890 GOTO9880
50000 DATA127,127,81,22,20,22,56,0
50001 DATA60,60,102,102,126,126,102,0:A
50002 DATA120,124,102,124,126,103,126,0:B
50003 DATA60,126,102,96,102,126,60,0:C
50004 DATA124,126,99,99,99,126,124,0:D
50005 DATA127,126,96,124,96,126,127,0:E
50006 DATA127,126,124,96,124,120,112,0:F
50007 DATA62,121,112,115,115,63,31,0:G
50008 DATA115,115,115,127,127,115,115,0:H
50009 DATA28,28,28,28,28,28,28,0:I
50010 DATA127,127,7,7,103,63,31,0:J
50011 DATA115,119,126,126,126,119,115,0:K
50012 DATA112,112,112,112,112,126,127,0:L
50013 DATA119,127,107,107,107,99,119,0:M
50014 DATA103,115,123,127,127,119,115,0:N
50015 DATA62,127,119,119,119,127,62,0:O
50016 DATA126,127,115,127,126,112,112,0:P
50017 DATA62,127,99,111,111,127,62,0:Q
50018 DATA126,127,115,127,124,126,119,0:R
50019 DATA63,126,96,62,3,63,126,0:S
50020 DATA127,127,93,28,28,28,28,0:T
50021 DATA119,119,119,119,127,127,28,0:U
50022 DATA119,119,50,58,58,28,8,0:V
50023 DATA119,99,107,107,107,127,54,0:W
50024 DATA99,119,54,62,62,119,99,0:X
50025 DATA119,59,31,14,14,14,14,0:Y
50026 DATA127,63,7,28,112,126,127,0:Z
50027 DATA240,240,192,192,192,192,240,240:[
50028 DATA0,36,102,102,102,102,36,0:£
50029 DATA15,15,3,3,3,3,15,15:]
50030 DATA126,126,126,78,62,126,126,126:↑
50031 DATA239,239,239,0,251,251,251,0:←
50032 DATA0,0,0,0,0,0,0,0
50033 DATA255,0,255,0,255,0,255,0:!
50034 DATA255,255,255,189,60,60,60,60:"
50035 DATA0,60,60,24,255,24,60,36:#
50036 DATA101,85,87,87,85,85,85,101:$
50037 DATA24,24,24,60,60,126,255,48:%
50038 DATA127,254,127,254,127,254,127,254:&
50039 DATA0,24,24,126,126,24,24,24:'
50040 DATA24,48,96,96,96,96,48,24:(
50041 DATA24,12,6,6,6,6,12,24:)
50042 DATA48,72,72,16,16,0,16,0:*
```

```
50043 DATA0,60,60,126,126,60,60,0:+
50044 DATA0,0,0,0,0,12,24,48:,
50045 DATA0,0,0,126,126,0,0,0:-
50046 DATA0,0,0,0,0,96,96,0:.
50047 DATA0,6,12,24,24,48,96,0:/
50048 DATA0,24,60,102,102,60,24,0:0
50049 DATA0,12,28,60,12,12,12,0:1
50050 DATA0,12,18,4,8,16,30,0:2
50051 DATA0,30,2,4,2,18,30,0:3
50052 DATA0,16,16,16,20,30,4,0:4
50053 DATA0,30,16,28,2,2,28,0:5
50054 DATA0,12,18,16,28,18,12,0:6
50055 DATA0,30,18,4,8,16,16,0:7
50056 DATA0,12,18,12,18,18,12,0:8
50057 DATA0,24,36,28,4,36,24,0:9
50058 DATA-1

READY.
```

Wir hoffen, daß Sie noch viel Spaß mit diesem Spiel haben.

7. *INTERRUPTS*

Laut Definition sind Interrupts Routinen, die bei einer Anforderung vom Computer von selber angesprungen werden.
Der C-64 kennt mehrere Interrupts:
Neben dem Raster-Interrupt (siehe Tips & Tricks 1) sind noch vorhanden:

Höchste Priorität:	RESET
Mittlere Priorität:	NMI
Unterste Priorität:	IRQ

Die Priorität gibt an, welche dieser Routinen zuerst angesprungen wird, wenn mehrere gleichzeitig vorliegen sollten.
Obwohl andere Autoren Gegensätzliches meinen, lassen sie sich alle verhindern.

RESET

Softwaremäßig läßt sich RESET durch SYS 64738 durchführen. Diese Adresse ergibt sich aus den in $FFFC-FFFD im LOW/HIGH-Byte angegebenen Werten.
Da es jedoch oft vorkommt, daß ein Programm abstürzt, und eine Eingabe nicht mehr möglich ist, empfiehlt es sich, einen RESET hardwaremäßig herbeizuführen.
Das erreicht man durch Kurzschließen einer Masse- und einer RESET-Leitung.
Zweckmäßig ist ein Taster, den man fest ins Gehäuse einbaut. Bewährt hat sich auch ein Summer in der Leitung, da es manchmal vorkommt, daß man unbeabsichtigt den Kontakt schließt, und dann dem Programm die Schuld gibt.

Was führt der Computer eigentlich aus, wenn ein RESET vorliegt?
Bedingt durch die Adressen $FFFC-$FFFD führt der Computer

einen Sprung zur Adresse $FCE2 aus.
Sehen wir uns das Programm ab dieser Adresse doch einmal an:

```
FCE2  LDX #$FF
FCE4  SEI
FCE5  TXS
FCE6  CLD
FCE7  JSR $FD02
FCEA  BNE $FCEF
FCEC  JSP ($8000)
FCEF  STX $D016
FCF2  JSR $FDA3
FCF5  JSR $FD50
FCF8  JSR $FD15
FCFB  JSR $FF5B
FCFE  CLI
FCFF  JMP ($A000)
```

Die ersten vier Adressen sind für uns nicht so wichtig. Sie dienen nur dem Computer. Zu sagen ist nur, daß in Adresse $FCE4 das sog. Interrupt-Disable-Flag gesetzt wird, damit der Computer im weiteren Verlauf nicht mehr gestört wird.
Probieren Sie es aus:
Ein einmal ausgelöster RESET läßt sich nicht mehr mit RUN-STOP/RESTORE verhindern.
Interessant wird es erst wieder in Adresse $FCE7.
Dort springt der Computer zu einem Unterprogramm ab Adresse $FD02.
Dieses Unterprogramm ist würdig, angeschaut zu werden:

```
FD02  LDX #$0F
FD04  LDA $FD0F,X
FD07  CMP $8003,X
FD0A  BNE $FD0F
FD0C  DEX
FD0D  BNE $FD04
FD0F  RTS
```

In diesem Unterprogramm liest der Computer der Reihe nach bestimmte Werte aus dem ROM, und vergleicht sie mit den Werten, die bei Adresse $8004-$8008 vorliegen. Sind diese Werte identisch, so wird ein indirekter Sprung ausgeführt.
Der Computer liest dabei die Werte der Adressen $8000-$8001 und faßt $8000 als LOW- und $8001 als HIGH-Byte der anzuspringenden Adresse auf.
Steht also in $8000 der Wert $00, und in $8001 $60, so geht es weiter bei Adresse $6000.
Wollen Sie also ein Programm bei RESET starten lassen, so müssen die Adressen $8000-$8008 folgendermaßen aussehen:

```
8000 LB Sprungvektor bei RESET
8001 HB
8002 LB Sprungvektor bei NMI (siehe NMI)
8003 HB
8004 195 "C" mit gesetztem Bit 7
8005 194 "B"  "      "      "  7
8006 205 "M"  "      "      "  7
8007 56  "8"
8008 48  "0"
```

Wollen Sie, daß Ihr Computer gleich nach dem Anschalten ein Programm durchführt, so müssen Sie ein EPROM in diesem Bereich adressieren und die Werte wie angegeben setzen.
Nach diesem Prinzip arbeiten auch die Module, die sich nach dem Anschalten sofort melden.

Doch weiter zum RESET.
Sollte ab $8000 kein Programm gefunden werden, macht der Computer bei Adresse $FCEF weiter. In den darauffolgenden Unterprogrammen initialisiert er neu die Pages 0,2 und 3, die E/A-Teile und den Video-Bereich.
Zuletzt führt er schließlich einen Sprung zum BASIC-Kaltstart durch.

Man kann ein BASIC-Programm auch durch RESET starten lassen. Dann muß das Programm, auf das die Adressen $8000-$8001 zeigen, folgendermaßen aussehen:

```
JSR $A659
JMP $A7AE
```

Diese zwei Zeilen sind das Maschinen-Äquivalent zum Basic-Befehl RUN.

NMI

NMI hat eine niedrigere Priorität als RESET.
Die NMI-Einsprungadresse wird durch die Adressen $0318/$0139 (792/793) festgelegt. Normalerweise ist dies die Adresse $FE47 (65095).
Da die eigentliche Routine bei $FE43 (65091) beginnt, schauen wir uns doch die Adressen ab da doch einmal an:

```
FE43   SEI
FE44   JMP($0318)
FE47   PHA
FE48   TXA
FE49   PHA
FE4A   TYA
FE4B   PHA
FE4C   LDA #$7F
FE4E   STA $DD0D
FE51   LDY $DD0D
FE54   BMI $FE72
FE56   JSR $FD02
FE59   BNE $FE5E
FE5B   JMP($8002)
```

```
FE5E   JSR $F6BC
FE61   JSR $FFE1
FE64   BNE $FE72
FE66   JSR $FD15
FE69   JSR $FDA3
FE6C   JSR $E518
FE6F   JMP($A002)
```

Als erstes fällt auf, daß in Adresse $FE43 das Interrupt-Disable-Flag gesetzt wird.
Will man also den Vektor $0318-0319 auf ein eigenes Programm zeigen lassen, so muß dieses Programm den Befehl CLI enthalten, da sonst der IRQ verhindert ist (welche Folgen das hat, erfahren Sie im nächsten Teil dieses Kapitels, IRQ).
Beispiel für ein eigenes Programm:

```
POKE792,226:POKE793,252
```

Jetzt zeigt der NMI-Vektor auf RESET.
Bei Drücken der RESTORE-TASTE (=NMI) wird ein RESET ausgelöst. Wenn Sie nach dem RESET noch einmal RESTORE drücken, ist der Effekt natürlich vorbei, da die Adressen $0318/$0319 wieder auf ihre Normalwerte gebracht worden sind.

Doch weiter zur eigentlichen NMI-Routine:
Die Adressen $FE47-$FE4B (65095-65099) retten den Akkumulator, das Y- und das X-Register.
In Adresse $FE54 (65108) wird geprüft, ob die RS 232-Schnittstelle aktiv ist. Sollte das der Fall sein, so wird zu Adresse $FE72 (65138) gesprungen, wo die NMI-Routine für die RS 232-Schnittstelle beginnt.
In Adresse $FE56 (65110) beginnt etwas, was Sie schon vom RESET her kennen:
Es wird geprüft, ob ein Modul ab $8000 vorliegt.

Sollte das der Fall sein, so wird der Inhalt von $8002 als LOW-, und der Inhalt von $8003 als HIGH-Byte der jetzt anzuspringenden Adresse interpretiert und zu dieser Adresse mit JMP($8002) verzweigt.
Da RESET und NMI auf das Vorhandensein bestimmter Werte bei $8000 reagieren, ist es mit ganz einfachen Mitteln möglich, zwei verschiedene Proramme zu starten. Eins beginnt beim Auslösen eines RESET, und das andere beim Drücken der RESTORE-Taste.
In Adresse $FE61 (65121) wird die STOP-Taste abgefragt. Ist sie gedrückt, so kommt der Computer zu Adresse $FE66 (65126), und initialisiert die I/O-Einheit neu, löscht den Bildschirm und führt schließlich einen Sprung zum BASIC-Warmstart aus.
Hier sehen Sie auch, warum das Drücken der RESTORE-Taste normalerweise nichts auslöst:
Es wird immer erst die STOP-Taste abgefragt. Ist sie nicht gedrückt, so wird zur NMI-Routine für die RS 232-Schnittstelle verzweigt.

Probieren Sie das neu erworbene Wissen doch einmal aus:

```
10 FOR X = 0 TO 8: READ A: POKE 32768+X,A :NEXT X
20 DATA 68,229,53,164,195,205,56,48

RUN
```

Nun sehen die Adressen $8000-$8008 (32768-32776) folgendermaßen aus:

32768	68	32772	195
32769	229	32773	194
32770	53	32774	205
32771	164	32775	56
		32776	48

Der Inhalt der Adressen 32768-32769 ergibt die Adresse, zu der der Computer im Falle eines RESET springt. Diese Adresse (58692) ist ein Unterprogramm das den Bildschirm löscht.
Der Inhalt der Adressen 32770/32771 ergibt die Adresse, zu der der Computer im Falle einer NMI-Anforderung springt. Hier ist es die Adresse 42037. Die Unterroutine, die ab dieser Adresse beginnt, gibt die Fehlermeldung "out of memory error" aus.
Die Adressen 32772-32776 stellen die Codes da, die der Computer braucht, um ein ROM ab $8000 zu erkennen.
Wenn Sie nun nach Eingabe dieser 2 Zeilen und dem Starten RESTORE drücken, so wird die Meldung "out of memory error" ausgegeben. Es kann vorkommen, daß der Cursor auf seiner Stelle bleibt. Das ist ganz normal, da die IRQ-Routine den Cursor verwaltet. Da NMI aber eine höhere Priorität als IRQ hat, wird die NMI-Routine angesprungen, bevor die IRQ-Routine ihre Arbeit beendet hat.
Wenn Sie einen RESET auslösen, wird der Bildschirm gelöscht. Damit haben Sie eine gute Möglichkeit gefunden, ein Programm zu starten.
Denkbar wäre z.B. die Möglichkeit, ein Programm im Bereich $8000 so abzuSAVEn, daß das Programm nach dem Laden durch Drücken der RESTORE-Taste oder durch RESET gestartet wird.
Da man dann keine BASIC-Zeile hat, die auf den Anfang des Programms hinweist, ist es für fremde Benutzer schwer, das Programm zu starten, sollten sie nicht durch Zufall darauf kommen. Noch sicherer wäre diese Lösung, wenn man nach Drücken von RESET bzw. RESTORE noch ein Code-Wort eingeben muß (ohne daß danach gefragt wird).
Für Leute, die sich mit diesen Dingen nicht so gut auskennen, dürfte dieser Schutz nicht leicht zu knacken sein. Noch schwerer wird das Ganze, wenn man noch andere Arten des Programmschutzes dazunimmt (z.B Autostart).

Dieser Interrupt hat zwar die niedrigste Priorität, sie ist aber ebenso wichtig wie die anderen. Da sie weit am häufigsten angesprungen wird, halten wir sie sogar für die Wichtigste.
Die Start-Adresse der IRQ-Routine ist in den Adressen $0314/$0315 (788/789) verankert. Dort kann sie leicht verändert werden.
Probieren Sie es aus:

```
POKE788,226:POKE789,252
```

Sobald Sie RETURN gedrückt haben, führt der Computer einen RESET aus.
Wieso das?
Nun, 226+252*256 ergibt 64738, und 64738 ist die Adresse für RESET.
Und nun das Wichtigste:
Die IRQ-Routine wird nicht wie NMI oder RESET erst angesprungen, wenn ein bestimmter Schalterkontakt vorliegt, sondern alle 1/60 sec!
Deswegen ist die IRQ-Routine auch so wichtig:
Sie erledigt alle Aufgaben, die immer wieder gemacht werden müssen. Sie bringt z.B. die Uhr immer auf den neuesten Stand, läßt den Cursor blinken und fragt die Stop-Taste ab.
Schauen wir uns die IRQ-Routine doch einmal an:

```
EA31   JSR $FFEA
EA34   LDA $CC
EA36   BNE $EA61
EA38   DEC $CD
EA3A   BNE $EA61
EA3C   LDA #$14
EA3E   STA $CD
EA40   LDY $D3
```

```
EA42   LSR $CF
EA44   LDX $0287
EA47   LDA($D1),Y
EA49   BCS $EA5C
EA4B   INC $CF
EA4D   STA $CE
EA4F   JSR $EA24
EA52   LDA($F3),Y
EA54   STA $0287
EA57   LDX $0286
EA5A   LDA $CE
EA5C   EOR #$80
EA5E   JSR $EA1C
EA61   LDA $01
EA63   AND #$10
EA65   BEQ $EA71
EA67   LDY #$00
EA69   STY $C0
EA6B   LDA $01
EA6D   ORA #$20
EA6F   BNE $EA79
EA71   LDA $C0
EA73   BNE $EA7B
EA75   LDA $01
EA77   AND #$1F
EA79   STA $01
EA7B   JSR $EA87
EA7E   LDA $DC0D
EA81   PLA
EA82   TAY
EA83   PLA
EA84   TAX
EA85   PLA
EA86   RTI
```

In Adresse $EA31 (59953) wird zu einem Unterprogramm ab Adresse F69B (63131) gesprungen, das die Zeit auf den neuesten Stand bringt und die Stop-Taste abfragt. Überspringt man den Aufruf dieses Unterproramms in der IRQ-Routine, so wird die Zeit nicht mehr weitergestellt, und RUN-STOP funktioniert auch nicht mehr (außer während einer Cassetten-Operation, da da RUN-STOP nicht über den IRQ abgefragt wird).
Probieren Sie es aus:

```
POKE788,52
```

setzt den IRQ-Anfang auf $EA34 (59956).
Geben Sie nun folgendes kleines Programm ein:

```
10 PRINT TI$
20 GOTO 10
```

Nach dem Starten werden Sie merken, daß TI$ wie erwartet nicht mehr weitergesetzt wird, sondern auf dem Stand stehenbleibt, auf dem es vor der Eingabe von POKE 788,52 war.
Auch die STOP-Taste funktioniert nicht mehr.
Wieso geht aber RUN-STOP/RESTORE?
Schauen Sie sich dazu noch einmal den Teil über NMI an:
Sobald die RESTORE-Taste gedrückt ist, wird zu der NMI-Routine gesprungen. Innerhalb dieser Routine wird geprüft, ob die STOP-Taste gedrückt ist. Die IRQ-Routine hat also damit garnichts zu tun, im Gegenteil, sie wird sogar noch verhindert. In Adresse $EA36 (59958) wird geprüft, ob der Cursor "angeschaltet" ist. Sollte dies nicht der Fall sein, so wird sofort zur Adresse $EA61 (60001) gesprungen. Ist er "angeschaltet", so geht es normal weiter. Aber Moment mal! Wenn die Adresse $CC (204) darüber entscheidet, ob der Cursor an oder aus ist, dann kann man sie vielleicht ja auch beeinflussen.
Ja, es geht:

```
10 POKE204,0:REM CURSOR EIN
20 GETA$:IFA$=""THEN20
30 POKE204,1:REM CURSOR AUS
```

Wenn Sie dieses Programm ausprobieren, ist ein Cursor da. So kann man also ganz einfach den Benutzer auf eine GET-Anweisung aufmerksam machen. Das Programm hat jedoch einen Nachteil: Drücken Sie eine Taste (verlassen Sie also die Zeile 20), wenn der Cursor gerade auf dem Bildschirm erscheint, so wird er da bleiben, und ein neuer Cursor erscheint erst, beim Drücken einer weiteren Taste.
Fügen Sie also noch folgende Zeile ein:

```
25 IFPEEK(207)=1THEN25
```

Aus Speicherstelle 207 geht hervor, ob der Cursor gerade auf dem Bildschirm ist (=1) oder nicht (=2).

Weiter in der IRQ-Routine:
In Adresse $EA38 (59960) wird der Timer für den Cursor um Eins erniedrigt. Ist er ungleich Null, so wird zur Adresse $EA61 (60001) gesprungen. Ist er Null, dann wird er auf $14 (#20) gesetzt (da der Computer die IRQ-Routine alle 1/60 sec anspringt, und dies 20mal tun muß, um den Cursor zu verändern, können Sie leicht nachrechnen, daß er in 1/3 sec Intervallen blinkt).
Durch einfaches Ändern der IRQ-Routine kann man die Blinkzeit verändern:

```
   JSR $FFEA    ;Zeit,STOP-Taste
   LDA $CC      ;Cursor an?
   BNE  *       ;nein
   DEC $CD      ;Timer erniedrigen
   BNE  *       ;ungleich Null, also weiter
   LDA $FF      ;Timer neu setzen
   JMP $EAE     ;IRQ weiter
 * JMP $EA61    ;normal weiter
```

Als BASIC-Loader:

```
10 FORX=0TO18:READA:POKE24625+X,A:NEXTX
20 DATA 32,234,255,165,204,208,9,198,205,208,5,165,255,76,
       62,234,76,97,234
```

Nach RUN müssen Sie nur noch

```
POKE788,96
```

eingeben, da die IRQ-Routine auf die neu erstellte Routine verändert werden muß.
Nun können Sie in Speicherstelle 255 beliebige Werte von 0 (kein Blinken des Cursors) über 1 (sehr schnelles Blinken) bis 255 (sehr langsames Blinken) POKEn. Da die Adresse 255 dauernd gelesen wird, wirkt sich das Eingeben eines anderen Wertes in diese Adresse sofort auf die Blinkzeit des Cursors aus.

In den nächsten Adressen, bis zu $EA5E (59998), führt der Computer die Cursorfunktionen zu Ende, d.h. er gibt den Cursor aus, merkt sich das Zeichen unter dem Cursor und dessen Farbe usw.
Interessant ist auch der Effekt, der sich ergibt, wenn man den ganzen Teil der IRQ-Routine, der den Cursor betrifft, überspringt:

```
POKE788,97
```

Nun wird

a) Keine Zeit mehr berechnet
b) Die STOP-Taste nicht mehr abgefragt
c) Kein Cursor mehr ausgegeben

Die Zeichen erscheinen aber immer noch, da die Tastatur erst später in der IRQ-Routine abgefragt wird. Von Adresse $EA61-$EA79 (60001-60025) geht es um den Rekorder. Sollte eine Taste am Rekorder gedrückt sein, so wird der Motor angestellt. Ist keine gedrückt, wird er ausgestellt (sofern er an war).
Probieren Sie es aus:

a) Geben Sie ein: POKE788,123 (IRQ-Anfang = $EA7B)
   Wenn Sie jetzt auf PLAY drücken, wird der Motor nicht laufen.
b) Drücken Sie RUN-STOP-RESTORE, um wieder in den normalen Modus zu kommen.
   Drücken Sie jetzt die PLAY-Taste am Rekorder und geben Sie nun POKE788,123 ein.
   Wenn Sie jetzt die STOP-Taste am Rekorder drücken, so wird der Motor immer noch laufen.

In Adresse $EA7B (60027) wird zu einem Unterprogramm gesprungen, das die Tastatur-Abfrage erledigt, und die entsprechenden Zeichen ausgibt.
Wenn Sie auch noch diese Adresse mit

```
POKE788,126
```

übergehen, so ist der Computer völlig hilflos.
RUN-STOP/RESTORE funktioniert aber immer noch. Das liegt daran, daß RESTORE nicht wie die anderen Tasten abgefragt wird. Von der RESTORE-Taste geht eine Leitung fast direkt zum Pin 4 (NMI) der CPU. RESTORE wird also direkt abgefragt.

In den Adressen $EA81-$EA85 (60033-60037) werden die originalen Werte der Register wieder hergestellt.
Schließlich wird mit dem Befehl RTI (Return From Interrupt) die IRQ-Routine abgeschlossen.

Da die IRQ-Routine leicht zu verbiegen ist, und so regelmäßig angesprungen wird, ist sie für viele Zwecke zu gebrauchen. Auch in diesem Buch wird sie oft benutzt. Beispiele finden Sie auf den folgenden Seiten. Vielleicht bekommen Sie dort eine Idee, wie Sie die IRQ-Routine nutzen können. Sie dürfen nur nie vergessen, am Ende Ihrer eigenen Routine wieder zur ursprünglichen IRQ-Routine zurück zu springen da sonst

a) die STOP-Taste
b) die UHR
c) der Cursor
d) der Rekorder
e) die Tastatur

lahmgelegt sind (es sei denn, Sie wollen diesen Effekt erzielen).

. . . *UND WIE MAN DEN IRQ PROGRAMMIERT !*

Nun einige Programmier-Tips, betreffend der IRQ und was man damit machen kann:
Wenn man selbst eine Interrupt-Routine programmiert, muß man den Vektor der Adressen 788/789 ändern. Wie diese Änderung vorgeht, wird später erklärt. Der größte Vorteil der Interrupt-Routine besteht darin, daß sie alle 1/60 sec. aus dem Direkt- sowohl wie aus dem Programm-Modus angesprungen wird.
Da man dadurch eine Vielzahl von Möglichkeiten abdecken kann, stellen wir Ihnen nun einige diesbezügliche Programme vor:

Immer Aktiv

Unser erstes Beispiel macht deutlich, welche Möglichkeiten sich durch die IRQ-Programmierung eröffnen, und in welcher Geschwindigkeit der IRQ arbeitet.

```
10 DATA 120,169,15,141,20,3,169
20 DATA 159,141,21,3,133,56,88
30 DATA 96,238,32,208,76,49,234
40 FOR I= 40704 TO40724
50 READ A
60 POKE I,A
70 S=S+A
80 NEXT I
90 IF S <> 2171 THEN PRINT "FEHLER IN DATAS!":END
100 PRINT "DATAS OK": SYS 40704
110 NEW
```

Die Routine macht eigentlich nichts weiter, als bei jedem Interrupt - Ansprung den Wert der Bildschirmrahmenfarbe um eins zu erhöhen. Doch der Interrupt wird so schnell

hintereinander angesprungen, so daß man keine einheitliche Bildschirmrahmenfarbe mehr erkennen kann.
Erkennen kann man, daß die Arbeitsgeschwindigkeit durch eine gut programmierte Interrupt - Routine nicht vermindert wird. Sie können, während diese Routine läuft, weiterhin BASIC-Zeilen schreiben oder bearbeiten lassen.
Wenn Sie das Flackern des Bildschirmrahmens stört, drücken Sie einfach Runstop/Restore. Starten können Sie die Routine mit

SYS 40704

Man kann diese Routine grundsätzlich in zwei Teile teilen. Im ersten Teil werden der IRQ Vektor geändert und die Routine vor dem Überschreiben geschützt. Das Ändern der Vektoren ist in BASIC nicht ohne Weiteres möglich. Denn wenn die Anfangsadresse wie bei der Routine "Immer aktiv" bei 40719 anfängt (Anfang des zweiten Teils), und Sie geben

Poke 788, 15 : Poke 789, 159

ein, dann passiert folgendes:
Sobald die Zeile den Wert (788) geändert hat, den Wert (789) aber noch nicht, springt der Computer nicht nach $EA31, sondern nach $EA0F. Der Computer hängt sich dann meistens auf.
Diese Routinen ändern den Wert automatisch im 1.Teil der Maschinensprache-Routine. Auf einer der nächsten Seite wird beschrieben, wie man die Vektoren auch in BASIC ändern kann.
In Maschinensprache gibt es den Befehl SEI (SEt Interrupt), der verhindert, daß ein Interrupt ausgeführt wird. Dann können Sie ungehindert den Vektor ändern. Nach CLI (CLear Interrupt),einem weiteren Maschinensprachebefehl, führt der Computer den Interrupt mit den neuen Werten aus. Dann wird die Routine im ersten Teil noch abgeblockt, d.h., daß sie weder von Variablen noch vom BASIC-Programm ohne Weiteres

überschrieben wird. Dafür setzen wir einfach das BASIC-Speicherende auf den Anfang der Routine.
Jetzt den ersten Teil zur besseren Übersicht mit allen Maschinensprachebefehlen:

```
SEI                 ; verhindert Interrupt
LDA #               ; hier wird das Low-Byte
                      der Anfangsadresse des 2.
                      Teils bestimmt.
STA $ 0314          ; speichert dieses Low-Byte
                      in dem IRQ-Zeiger ab.
LDA #               ; bestimmt das High-Byte
                      des 2.Teils.
STA $ 0315          ; speichert das High-Byte
                      im IRQ-Zeiger ab.
STA $ 38            ; setzt Speicherende High-
                      Byte auf den Anfang der
                      gesamten Routine, blockt
                      sie auf diese Weise ab.
CLI                 ; Interrupt wird wieder
                      ausgeführt.
```

Außerdem kann man im ersten Teil wie zum Beispiel bei der Piepton-Routine, die auf einer der folgenden Seiten abgedruckt ist, Werte setzen, die als Grundlage für den zweiten Teil dienen.

Der zweite Teil:
Dies ist die eigentliche Routine. Nur dieser Teil wird vom Computer angesprungen. Hier kann der Programmierer seine Fantasie spielen lassen und irgendetwas hineinschreiben. Die Interrupt-Routinen in diesem Buch sollen ja nur Anregungen und Beispiele sein. Nur, zu lang sollte dieser Teil nicht werden, damit sich die Arbeitsgeschwindigkeit des Direktmodus nicht verringert.

Das Wichtigste an diesem Teil ist, daß am Ende der Routine der Sprung zur Adresse $EA31 erfolgt, sonst würde sich der Computer aufhängen.

TASTATUR-PIEP

Nach soviel grauer Theorie endlich wieder einmal ein Programm. Bei einigen Computern ertönt beim Betätigen einer Taste ein Piepton. Beim Eingeben von Programmen und Tabellen ist das sehr nützlich. Der C-64 hat diese Einrichtung nicht. Es ist jedoch möglich, diese akustische Hilfe durch eine IRQ-Routine zu erzeugen:

```
0 REM PIEPTON-ROUTINE
10 FOR I=0 TO 61
20 READ A
30 S=S+A
40 POKE 40704+I,A
50 NEXT I
60 IF S<>6973 THEN PRINT "FEHLER IN DATAS"
   : END
70 PRINT "DATAS OK"
80 SYS 40704
90 DATA 169,255,141,6,212,141,24,212,169
100 DATA 9,141,5,212,169,103,141,1,212
110 DATA 169,33,141,0,212,120,169,38,141
120 DATA 20,3,169,159,141,21,3,133,56,88
130 DATA 96,72,165,203,201,64,208,9,169
140 DATA 0,141,4,212,104,76,49,234,169
150 DATA 17,141,4,212,76,50,159
```

Zum besseren Verständnis hier auch das Maschinensprache-Listing:

```
9F00 LDA #$FF
9F02 STA $D406
```

```
9F05 STA $D418
9F08 LDA #$09
9F0A STA $D405
9F0D LDA #$67
9F0F STA $D401
9F12 LDA #$21
9F14 STA $D400
9F17 SEI
9F18 LDA #$26
9F1A STA $0314
9F1D LDA #$9F
9F1F STA $0315
9F22 STA $38
9F24 CLI
9F25 RTS
9F26 PHA
9F27 LDA $CB
9F29 CMP #$40
9F2B BNE $9F36
9F2D LDA #$00
9F2F STA $D404
9F32 PLA
9F33 JMP $EA31
9F36 LDA #$11
9F38 STA $D404
9F3B JMP $9F32
```

Den Teil 1b von $9F17 bis $9F25 kennen wir bereits aus der Routine "Immer Aktiv". Hier werden wieder die Vektoren geändert. Dagegen werden im Teil 1a von $9F00 bis $9F16 fast alle Werte für die Tonerzeugung gesetzt, ausgenommen die Wellenform. Diese Werte werden in der eigentlichen Routine nicht mehr benötigt.
Teil 2 der Routine fragt die Adresse 203 ab, ob irgendeine Taste gedrückt wurde. Liegt hier der Wert 64 vor, wurde keine Taste gedrückt, und die Routine setzt die Wellenform

auf den Wert 0, d.h. es ist kein Ton zu hören. Im Falle, daß eine Taste gedrückt wurde, ist der Wert ungleich 64, das Programm setzt die Wellenform auf 17 und der Piepton wird hörbar. Nach dieser Abfrage springt das Programm zur Adresse $EA31 (59953) weiter.

NEBENBEI MUSIK

Sind Sie ein so grosser Musikfan, daß Sie beim Programmieren nicht auf Musik verzichten wollen, aber keine Musikkiste in der Nähe haben, und Ihre kleine Schwester Ihnen nichts vorsingen will (oder Sie es nicht wollen), dann ist dieses Programm genau das RichtiAe für Sie. Es sieht sehr lang aus. Das kommt durch die Noten, denn jede Note braucht 3 Werte: Frequenz low, Frequenz high und den Notenwert. Der BASIC-Loader ist in zwei Abschnitte zu fassen:
Im ersten wird die Routine geladen, im zweiten werden die Noten und ihre Werte eingelesen.
Saven Sie das Programm ab, bevor Sie es ausprobieren !!!

```
0 REM MUSIK AUS DEM IRQ
10 DATA120,169,38,141,20,3,169,144
20 DATA141,21,3,88,133,56,169,0
30 DATA141,243,159,169,1,141,241,159
40 DATA169,0,141,20,212,169,31,141
50 DATA24,212,141,19,212,96,206,241
60 DATA159,208,48,72,138,72,169,0
70 DATA141,18,212,174,243,159,189,0
80 DATA145,141,14,212,189,0,146,141
90 DATA15,212,189,0,147,141,241,159
100 DATA169,33,141,18,212,232,56,224
110 DATA60,144,2,162,0,142,243,159
120 DATA104,170,104,76,49,234
130 FORI=36864TO36957
140 READA
150 POKEI,A
160 S=S+A
170 NEXT
180 IFS<>11531THENPRINT"?FEHLER IN DATAS":END
190 INPUT"WIEVIELE NOTEN";N:POKE36944,N
200 REM NOTEN : ↑=60 NOTEN !!
```

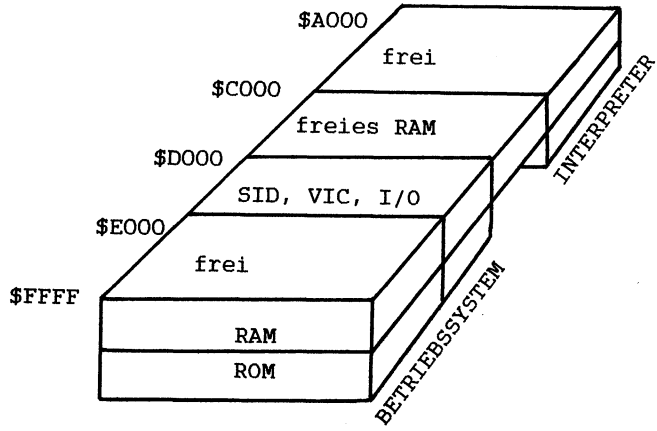
```
210 FORI=0TO 59
220 READL
230 POKE37120+I,L
240 READH
250 POKE37376+I,H
260 READW
270 POKE37632+I,W
280 NEXT
290 SYS36864
300 REM MUSIK-DATEN
310 DATA196, 9,10, 10,13,10, 10,13,10
320 DATA162,14,10,109,16,10, 10,13,10
330 DATA  109,16,10,162,14,10,196, 9,10
340 DATA   10,13,10, 10,13,10,162,14,10
350 DATA  109,16,10, 10,13,20,158,11,10
360 DATA  196, 9,10, 10,13,10, 10,13,10
370 DATA  162,14,10,109,16,10,103,17,10
380 DATA  109,16,10,162,14,10, 10,13,10
390 DATA  158,11,10,196, 9,10,247,10,10
400 DATA  158,11,10, 10,13,20, 10,13,10
410 DATA    0, 0,10,247,10,15,158,11, 5
420 DATA  247,10,10,196, 9,10,247,10,10
430 DATA  158,11,10, 10,13,20,196, 9,15
440 DATA  247,10, 5,196, 9,10,180, 8,10
450 DATA   55, 8,10,180, 8,10,196, 9,20
460 DATA  247,10,15,158,11, 5,247,10,10
470 DATA  196, 9,10,247,10,10,158,11,10
480 DATA   10,13,10,247,10,10,196, 9,10
490 DATA   10,13,10,158,11,10,162,14,10
500 DATA   10,13,20, 10,13,10,  0, 0,99
```

Nach dem Start des Programmes werden Sie zuerst gefragt, wieviele Noten das Musikstück hat. Bei unserem Lied müssen Sie 59 eingeben. Dann liest der Computer die Noten ein und startet die Routine.
Wenn bis dahin alles richtig verlaufen ist und auch kein Fehler in den DATAs ist, hören Sie nun den "Yankee Doodle". Wie oben schon erwähnt, können Sie jetzt programmieren und die Musik spielt nebenbei weiter. Sie können diese Routine aber auch während eines Spiels zusätzlich zu den Spezial-Effekten laufen lassen, denn die Musik läuft nur über die dritte Stimme.

Diese Routine können Sie selber benutzen und die Musikwerte ändern, ohne daß Sie selbst Maschinensprache können. Sie müssen dafür nur die Noten Ihres Musikstückes in der Form Frequeny low, Frequeny high und Notenwert statt der Noten des "Yankee Doodles" in die DATAs ab Zeile 300 schreiben. Wenn Sie dann das Programm starten, müssen Sie nur noch die Anzahl der Noten angeben. Das Programm kann bis zu 256 Noten verarbeiten.
Was die Notenwerte betrifft, können Sie die Werte selber wählen. Bei dem Lied "Yankee Doodle" haben wir für die Viertelnote "10", für die halbe Note "20" und für die ganze Note "40" als Notenwert gewählt. Sie brauchen sich aber nicht nach diesen Werten zu richten.

8. BETRIEBSSYSTEM: ROM IN RAM

Der Commodore 64 bietet eine gegenüber vielen anderen Heimcomputern außergewöhnliche und zugleich sehr interessante Eigenschaft:
Das Betriebssystem kann ins RAM des selben Adreßbereiches verlegt werden.
Um diesen Vorgang verstehen zu können, sehen Sie sich bitte einmal die folgende Skizze an. Sie zeigt die Speicherorganisation ab $A000:



Gewisse Speicherbereiche sind doppelt belegt. Das bedeutet, in demselben Adreßbereich befinden sich sowohl ein RAM-, als auch ein ROM-Speicher. Oder anders ausgedrückt, es gibt beispielsweise die Adresse $A000 "zweimal": Einmal im ROM und ein weiteres Mal im RAM des selben Speicherbereichs.
Der Computer kann jedoch nur einen der beiden Speicherbereiche ansprechen (adressieren), entweder ROM oder RAM.
Wir werden uns im Verlauf dieses Kapitels näher mit den beiden Bereichen $A000-$BFFF (BASIC-Interpreter) sowie $E000 bis $FFFF (E/A-Einheit) befassen.

Das ROM der beiden Bereiche enthält Interpreter und Ein/Ausgabe-Teil (E/A), das "darunterliegende" RAM hingegen "nichts" ,es wird wie beispielsweise der BASIC-Speicher bei jedem Ausschalten des Computers gelöscht.
Es liegt also nichts näher, als Interpreter und/oder E/A-Teil in dieses RAM zu kopieren und anschließend auf die kopierten Bereiche umzuschalten. Dies ist nicht weiter schwer, da der PEEK-Befehl das ROM ausliest, während der POKE-Befehl ins RAM (wohin sonst ?) schreibt.
Eine Kopierroutine für den Interpreter könnte in BASIC folgendermaßen aussehen:

```
10 FOR A= 10*4096 TO 12*4096 -1
20 POKE A,PEEK(A)
30 NEXT A
40 END
```

Entsprechend die BASIC-Kopierroutine für das Betriebssystem:

```
10 FOR A= 14*4096 TO 16*4096 -1
20 POKE A,PEEK (A)
30 NEXT A
40 END
```

Erst wenn der Inhalt des betreffenden ROM-Bereiches ins darunterliegende RAM kopiert worden ist, darf der Speicherbereich ins RAM umgeschaltet werden,Sonst kommt es zum Absturz des Computers.

Int.ROM	Int.RAM	BetrSys.ROM	BetrSys.RAM	Umschaltung
I	0	I	0	POKE 1,55
0	I	I	0	POKE 1,54
0	I	0	I	POKE 1,53
I	0	0	I	---------

Es ist von BASIC aus nicht möglich, das Betriebssystem allein im ROM arbeiten zu lassen !

Übrigens: Sicherlich wird Ihnen die Länge aufgefallen sein, die die BASIC-Kopier-Routinen benötigen, um die entsprechenden Speicherbereiche zu kopieren.
Das folgende Maschinenprogramm erledigt den gleichen Vorgang allerdings erheblich schneller: Es braucht weniger als eine Sekunde !

Zunächst das Assembler-Listing:

```
033C  LDX#$20
033E  LDA#$A0
0340  LDY#$00
0342  STY$22
0344  STA$23
0346  LDA($22),Y
0348  STA($22),Y
034A  INY
034B  BNE$0346
034D  INC$23
034F  DEX
0350  BNE$0346
0352  RTS
```

Und hier der BASIC-Loader:

```
10 FOR I=828 TO 851
20 READ X: POKE I,X: NEXT
30 DATA 162,32,169,160,160,0,132,34,133,35,177,34
40 DATA 145,34,200,208,249,230,35,202,208,244,96
50 SYS 828
```

Die Kopierroutine wird mit "SYS 828" aufgerufen.

Wollen Sie sowohl den Interpreter als auch den E/A-Teil mit dieser Routine kopieren, so ändern Sie im BASIC-Loader Zeile 50:

```
50 SYS 828: POKE 831,224: SYS 828
```

Was läßt sich jetzt aber mit dem im RAM befindlichen Betriebssystem anfangen ?

Sofern Sie das DATA BECKER-Buch "64 INTERN" besitzen, finden Sie das komplette Assembler-Listing beider Bereiche. Sie können nun dieses Ihren Wünschen entsprechend ändern. Es bietet sich weiterhin an, die BASIC-Fehlermeldungen oder Befehle zu verändern, BASIC-Befehls-Routinen abzuändern, etc.
Ihnen stehen nun alle Türen offen, bis hin zum eigenen Interpreter!

Wer in Maschinen-Sprache programmiert, steht oft vor großen Problemen, da die Assembler-Befehle nicht so vielseitig sind wie die BASIC-Befehle. PRINT in Maschinen-Sprache ist schon ein ganz schönes Problem. Doch vieles ist ganz einfach, wenn man sich im Betriebssystem auskennt, denn viele Routinen sind schon benutzerfertig vorhanden.
Das Betriebssystem teilt sich in zwei Teile auf:

a) Der Interpreter
b) Der Ein-Ausgabeteil

Der Interpreter ist der Teil, der die BASIC-Befehle in für den Computer verständliche Maschinen-Befehle übersetzt. Der E/A - Teil ist für alle Operationen mit der Außenwelt bestimmt. Sehen Sie sich die nächsten Seiten ruhig einmal an, vielleicht finden auch Sie einige Routinen, die für Sie von Nutzen sind.

Blockverschiebe-Routine
Einsprungadresse: $A3B8 (41912)
Mit Hilfe dieser Routine kann man schnell und problemlos einen Speicherbereich in einen anderen Bereich verschieben. Dabei muß der alte Blockanfang in den Adressen $5F/$60 (95/96), das alte Blockende (+1!) in den Adressen $5A/$5B (90/91) und das neue Blockende (+1!) in den Adressen $58/$59 (88/89) stehen.

Prüfung auf Platz im Stapel
Einsprungadresse: $A3FB (41979)
Diese Routine prüft nach, ob noch genug Platz im Stapel

vorhanden ist. Sollte nicht mehr genug vorhanden sein, so erfolgt die Fehlermeldung "out of memory".

Ausgabe von "out of memory"
Einsprungadresse: $A435 (42037)
Diese Routine veranlaßt den Computer zur Ausgabe von "out of memory error". Der Computer springt danach in den Ready-(Direkt-)Modus.

Fehlermeldung ausgeben
Einsprungadresse: $A43A (42042)
Mit Hilfe dieser Routine kann man leicht eine Fehlermeldung ausgeben. Der Computer geht nach der Ausgabe wieder in den Direkt-Modus.
Die Fehlernummer muß dabei im X-Register übergeben werden.
Folgende Nummern führen zu folgender Fehlermeldung:

NUMMER dez.	NUMMER hex.	FEHLERMELDUNG
1	1	too many files
2	2	file open
3	3	file not open
4	4	file not found
5	5	device not present
6	6	not input file
7	7	not output file
8	8	missing file name
9	9	illegal device number
10	A	next without for
11	B	syntax
12	C	return without gosub
13	D	out of data
14	E	illegal quantity
15	F	overflow
16	10	out of memory

17	11	undef/d statement
18	12	bad subscript
19	13	redim/d array
20	14	devision by zero
21	15	illegal direct
22	16	type mismatch
23	17	string too long
24	18	file data
25	19	formula too complex
26	1A	can/t continue
27	1B	undef/d function
28	1C	verify
29	1D	load
30	1E	break

Diese Routine kann man auch von BASIC aus nutzen:

```
POKE781,21:SYS42042
```

führt z.B. zur Ausgabe von "illegal direct error".

BASIC-Programmzeilen neu binden
Einsprungadresse: $A533 (42291)
Diese Routine bindet eine BASIC-Zeile neu, d.h. die Zeilen werden wieder in ihre richtige Reihenfolge gebracht.

NEW
Einsprungadresse: $A644 (42564)
Diese Routine ist der BASIC-Befehl NEW. Ein BASIC-Programm wird dadurch gelöscht (bzw "versteckt").

CLR
Einsprungadresse: $A660 (42592)
Diese Routine kommt dem BASIC-Befehl CLR gleich. Es werden also alle Variablen gelöscht.

RUN

RUN kann man nicht mit einem Befehl erledigen.
In Maschinen-Sprache sieht der BASIC-Befehl RUN folgendermaßen aus:

```
JSR $A659
JSR $A7AE
```

STOP-Taste

Einsprungadresse: $A82C (43052)

Bei dieser Routine wird die Stop-Taste abgefragt, und sollte sie gedrückt sein, so wird ein laufendes BASIC-Programm abgebrochen.
Sollte ein STOP-Befehl in einem BASIC-Programm auftauchen, so wird direkt zur Adresse $A82F (43055) gesprungen. Möchten Sie also ein BASIC-Programm von der Maschinen-Sprache aus sofort unterbrechen, so müssen Sie diese Adresse anspringen.

GOTO

Einsprungadresse: $A8A3 (43171)

Die eigentliche Goto-Routine beginnt ab der Adresse $A8AO (43168). In Adresse $A8AO folgt jedoch ein Sprung zu einem Unterprogramm, das die gültige Zeilennummer in die Adressen $14-$15 (2O-21) holt. Springen Sie also die Adresse $A8A3 an, so können Sie die Zeilennummer, die angesprungen werden soll, selbst bestimmen, indem Sie sie in Low- und High-Byte zerlegen und in den Adressen $14-$15 ablegen.

String ausgeben.

Einsprungadresse: $AB1E (43806)

Ein String wird ausgegeben. Dieser String muß eine Null als Endkennzeichen aufweisen; seine Adresse muß in das Y-Register und in den Akkumulator gegeben werden.

Akkumulator : Low-Byte
Y-Register : High-Byte

Zum Verständnis ein Beispiel:

```
LDA #$41
LDY #$03
JSR $AB1E
RTS
```

Mit diesem Programm wird der Kassettenpuffer bis zur ersten Null ausgegeben. Damit läßt sich leicht der Name eines Programms erfahren.
Das geht übrigens auch in BASIC:

```
POKE780,65:POKE782,3:SYS43806
```

Einige Strings, die im ROM untergebracht sind:

MELDUNG	AKKU(LOW)	Y-REG(HIGH)	ADRESSE
O.K.	100	163	$A364
ERROR(ohne CR)	105	163	$A369
IN	113	163	$A371
READY	118	163	$A376
BREAK(ohne CR)	129	163	$A381
BASIC BYTES FREE	96	228	$E460
(Einschaltmeldung)	115	228	$E473

Bei geeignetem Suchen werden Sie garantiert noch weitere Meldungen finden. Sie können natürlich auch einen eigenen Sring konstruieren.

Ausgabe eines Leerzeichens (bzw. Cursor right)
Einsprungadresse : $AB3B (43835)
Diese Routine gibt ein Leerzeichen oder Cursor right aus. Ein Leerzeichen wird ausgegeben, wenn ein File vorliegt (Adresse $13 (19) ungleich Null).

In dieser Routine sind noch zwei andere Routinen verborgen:

a) $AB45 (43845)
   Ausgabe eines Fragezeichens
b) $AB47 (43847)
   Ausgabe des im Akku stehenden Zeichens

Zeichen auf Klammer zu prüfen
Einsprungadresse: $AEF7 (44791)
Diese Routine prüft das Zeichen, auf das der Vektor $7A-$7B (122-123) zeigt, auf "Klammer zu". Falls es nicht Klammer zu ist, wird SYNTAX ERROR ausgegeben, und der Computer springt in den Ready-Modus.
In dieser Routine sind noch andere Routinen verborgen:

a) $AEFA (44794)
   Wie $AEF7, nur auf "KLAMMER AUF"
b) $AEFD (44797)
   Wie $AEF7, nur auf "KOMMA"
c) $AEFF (44799)
   Wie $AEF7, nur auf das im Akku stehende Zeichen

SYNTAX ERROR
Einsprungadresse: $AF08 (44808)
Diese Routine gibt "SYNTAX ERROR" aus. Der Computer geht danach in den Direkt-Modus.

Zeichen auf Buchstabe prüfen
Einsprungadresse: $B113 (45331)
Das Zeichen, das sich beim Ansprung dieser Routine im Akku befindet, wird daraufhin überprüft, ob es ein Buchstabe ist. Sollte es einer sein, so wird das Carry-Flag gelöscht.

FAC-Zahl in Integer-Zahl
Einsprungadresse: $B1AA (45482)
Die Zahl, die gerade im FAC steht, wird in eine Integer-Zahl

umgewandelt. Diese Integer-Zahl wird in Low- und High-Byte zerlegt und in die Adressen $64/$65 (100/101) gespeichert.

BAD SUBSCRIPT ERROR
Einsprungadresse: $B245 (45637)
Ausgabe von "bad subscript error" und Sprung in den Ready-Modus.

ILLEGAL QUANTITY ERROR
Einsprungadresse: $B248 (45640)
Ausgabe von "illegal quantity error" und Sprung in den Ready-Modus.

Test auf Direkt-Modus
Einsprungadresse: $B3A6 (45990)
Diese Routine prüft, ob sich der Computer im Direkt-Modus befindet. Ist er es, so wird "illegal direct error" ausgegeben.
Auch in dieser Routine sind wieder zwei andere verborgen:

a) $B3AB (45995)
   illegal direct error
b) $B3AE (45998)
   undef/d function error

FORMULA TOO COMPLEX ERROR
Einsprungadresse: $B4D0 (46288)

FAC in LOW-HIGH-BYTE
Einsprungadresse: $B7F7 (47095)
Diese Routine wandelt die Zahl, die im FAC steht (sie muß positiv und darf nicht größer als 65536 sein), in eine Sechzehn-Bit-Zahl um. Diese wird in Low- und High-Byte zerlegt und in Adresse $14/$15 (20/21) und Y-Register-Akku abgelegt.

OVERFLOW-ERROR
Einsprungadresse: $B97E (47486)

DEVISION BY ZERO ERROR
Einsprungadresse: $BB8A (48010)

LOW-HIGH-BYTE in Integer-Zahl
Einsprungadresse: $BDCD (48589)

Mit Hilfe dieser Routine kann man Low/High-Byte in eine Integer-Zahl verwandeln und ausgeben lassen.
Das Low-und das High-Byte muß im Akkumulator bzw. im X-Register stehen.
Diese Routine kann man auch gut von BASIC aus nutzen:

```
10 INPUT"LOW-BYTE";LO
20 INPUT"HIGH-BYTE";HI
30 POKE781,LO:POKE780,HI:SYS48589
40 PRINT:GOTO10
```

In dieser Routine werden übrigens zwei andere Routinen benutzt:

a) Die Integer-Zahl wird in eine Fließkommazahl umgewandelt ($BC49)
b) Die Fließkommazahl wird in einen ASCII-String umgewandelt ($BDDF)

BREAK ERROR
Einsprungadresse: $E107 (57607)

Auf weitere Zeichen prüfen
Einsprungadresse: $E211 (57873)
Hier wird geprüft, ob noch weitere Zeichen folgen.
Sollte dies nicht der Fall sein, so wird "syntax error" ausgegeben.
Möchte man vorher prüfen, ob ein Komma vorliegt, so muß die Einsprungadresse $E20E (57870) heißen.

BASIC-Kaltstart
Einsprungadresse: $E394 (58260)
Diese Routine kann man als Teil-Reset bezeichnen:

a) BASIC-Programme und Variablen werden gelöscht.
b) Der RAM wird wieder auf den Anschaltzustand gebracht.
c) Der Bereich $0300-$030B (768-779) wird wieder original gesetzt.
d) Das Anfangsbild erscheint.
e) Es wird zum BASIC-Warmstart gesprungen.

Manches dagegen wird nicht verändert, z.B. die Farben und der Bereich $0314/$0333 (788/819), in dem unter anderem der IRQ- und der NMI-Sprungvektor liegt.

Warten auf Commodore-Taste
Einsprungadresse : $E4E0 (58592)
Der Computer wartet darauf, daß die Commodore-Taste gedrückt wird. Wird sie nach einer gewissen Zeit nicht gedrückt, so springt der Computer von alleine wieder zurück.

Bildschirm-Reset
Einsprungadresse : $E518 (58648)
Diese Routine stellt den Bildschirm wieder neu her, d.h. die Farben werden wieder original gesetzt, der Bildschirm wird gelöscht, und der Cursor wird wieder in den Anschaltzustand versetzt.
Diese Routine kann man jetzt differenzieren:

a) Bildschirm-Reset ohne Verändern des Video-Controlers
   Einsprungadresse : $E51B (58651)
b) Bildschirm löschen
   Einsprungadresse : $E544 (58692)
c) Cursor Home
   Einsprungadresse : $E566 (58726)
d) Cursor Home und Neu-Initialisieren des Video-Controlers
   Einsprungadresse : $E59A (58778)
e) Nur Video-Controller initialisieren
   Einsprungadresse : $E5A0 (58784)

Rückschritt in vorhergehende Zeile
Einsprungadresse : $E701 (59137)
Bei Ansprung dieser Routine geht der Cursor eine Zeile nach oben.

Bildschirm scrollen
Einsprungadresse : $E8EA (59626)

Bei Ansprung dieser Routine wird der gesamte Bildschirm um eine Zeile nach oben geschoben, d.h. die oberste Zeile verschwindet und unten kommt eine Leerzeile dazu.

Bildschirmzeile löschen I
Einsprungadresse : $E9FF (59903)

Mit dieser Routine kann man eine Bildschirmzeile löschen. Die Nummmer dieser Zeile (die oberste Zeile hat die Nummer Null) muß man im X-Register übergeben.
Auch von BASIC her kann diese Routine genutzt werden:

```
POKE 781,(Zeilennummer):SYS 59903
```

Bildschirmzeile löschen II
Einsprungadresse : $E9FF (59905)

Diese Routine löscht, wie die oben aufgeführte Routine, die Bildschirmzeile, deren Nummer im X-Register steht. Zusätzlich kann dem Computer aber auch noch mitgeteilt werden, bis zu welcher Stelle diese Zeile gelöscht werden soll (0-39).
Diese Zahl muß im Y-Register übergeben werden.
Beispiel:

```
    LDA #$0A
    LDX #$00
    JSR $E9FF
    RTS
```

```
POKE781,0:POKE782,10:SYS 59905
```

Dieses Maschinen-Programm und das BASIC-Äquivalent löschen die 1. Bildschirmzeile (Nummer Null) bis zur 11. Stelle (Nummer 10).

Verzögerung von einer Millisekunde
Einsprungadresse : $EEB3 (60958)

Dieses Unterprogramm läßt den Computer eine Millisekunde warten.

Systemmeldungen ausgeben
Einsprungadresse : $F12B (61739)

Mit Hilfe dieser Routine kann man alle Strings ausgeben, die bei dem Umgang mit der Floppy & Datasette auftreten.
Zuerst wird allerdings getestet, ob man sich im Programm-Modus befindet. Sollte das der Fall sein, so wird nichts ausgegeben.
Der Offset der Meldung muß im Y-Register übergeben werden.

Y-REGISTER dez.	hex.	MELDUNG
0- 1	00-01	i-o error #
12- 13	0C-0D	searching
23	17	for
27- 28	1B-1C	press play on tape
46	2E	press record & play on tape
73- 74	49-4A	loading
81- 82	51-52	saving
89- 90	59-5A	verifying
99-100	63-64	found
106-107	6A-6B	o.k.

Diese Routine kann auch benutzt werden, ohne daß geprüft wird, ob man im Programm-Modus ist. Dann muß die Einsprung-Adresse allerdings $F12F (61743) heißen.

Searching (for filename) ausgeben
Einsprungadresse: $F5AF (62895)

Bei Ansprung dieser Routine wird zuerst geprüft, ob man im Programm-Modus ist. Sollte das der Fall sein, so wird sofort wieder zurückgesprungen.
Im anderen Fall wird "searching" ausgegeben. Nun wird

getestet, ob die Länge des Filenamens (abgespeichert in $B7) gleich Null ist. Wenn ja, so wird die Routine jetzt beendet. Wenn nein, so wird "for" ausgegeben und dann der Filename (abgespeichert im Low-und High-Byte in $BB-$BC).
Beispiel:

```
POKE183,2:POKE187,39:POKE188,241:SYS 62895
```

Nun wird "searching for o.k." ausgegeben, da der Zeiger 187-188 ($BB-$BC) auf die Systemmeldung "o.k." (Adresse $F127) gesetzt wurde.
Diese Routine kann auch benutzt werden, ohne daß vorher auf den Programm-Modus getestet wird. Die Einsprungadresse hieße in diesem Fall $F5B3 (62899).

Loading-Verifying ausgeben
Einsprungadresse: $F5D2 (62930)
Bei dieser Adresse wird "loading" bzw. "verifying" ausgegeben. Das hängt von der Adresse $93 (147) ab. Bei $93=0 wird "loading", und bei $93=1 "verifying" ausgegeben.

"saving" ausgeben
Einsprungadresse: $F68F (63119)
Diese Routine gibt "saving" aus. Wollen Sie, daß nicht getestet wird, ob man im Programm-Modus ist, so müssen Sie die Adresse $F693 (63123) anspringen.

TOO MANY FILES
Einsprungadresse: $F6FB (63227)

FILE OPEN
Einsprungadresse: $F6FE (63230)

FILE NOT FOUND
Einsprungadresse: $F701 (63233)

DEVICE NOT PRESENT
Einsprungadresse: $F707 (63239)

NOT INPUT FILE
Einsprungadresse: $F70A (63242)

NOT OUTPUT FILE
Einsprungadresse: $F70D (63245)

MISSING FILE NAME
Einsprungadresse: $F710 (63248)

ILLEGAL DEVICE NUMBER
Einsprungadresse: $F713 (63251)

Programm-Header von Band lesen
Einsprungadresse: $F72C (63276)
Dieses Programm liest den Header von Band (siehe Kapitel LOAD-SAVE)

Programm-Header generieren und auf Band schreiben
Einsprungadresse : $F76A (63338)
Mit diesem Unterprogramm kann man einen Programm-Header auf Band schreiben.
Folgende Adressen müessen folgende Merkmale enthalten:

$C1-$C2 (193-194) : Startadresse des Programms
$AE-$AF (174-175) : Endadresse des Programms
$B7 (183) : Anzahl der Zeichen des Filenamens
$BB-$BC (187-188) : Low-High-Byte-Zeiger auf den Filenamen

Diese Werte müssen vor Ansprung der Routine gesetzt werden.

Band nach Filenamen absuchen
Einsprungadresse : $F7EA (63466)
Diese Routine sucht auf dem Band nach einem bestimmten

Filenamen. Dieser Filename muß folgendermaßen bestimmt werden:

$B7	(183)	: Länge des gesuchten Filenamens (soll das nächste Programm gesucht werden, so muß diese Adresse auf Null gesetzt werden)
$BB-$BC	(187-188)	: Adresse des gesuchten Filenamens

Bei Ansprung dieser Routine sucht der Computer solange das Band ab, bis er den Header gefunden, das Bandende erreicht hat (EOT-Signal) oder unterbrochen wird.

Bandtaste abfragen I
Einsprungadresse : $F817 (63511)
Bei dieser Routine wird die Datasette abgefragt. Ist eine Bandtaste gedrückt, so wird sofort wieder zurückgesprungen. Im anderen Falle wird "press play on tape" ausgegeben und gewartet, bis eine Taste gedrückt wird. Daraufhin wird "ok" ausgegeben und zurückgesprungen.
Stop-Taste wird in dieser Routine abgefragt.

Bandtaste abfragen II
Einsprungadresse : $F82E (63534)
Auch diese Routine fragt die Datasette daraufhin ab, ob eine Taste gedrückt ist. Bei dieser Routine wird jedoch nichts ausgegeben, sondern sofort wieder zurückgesprungen. Es wird jedoch das Y-Flag folgendermaßen gesetzt:

a) Taste gedrückt : Y=1
b) Taste nicht gedrückt : Y=0

Bandtaste abfragen III
Einsprungadresse : $F838 (63544)
Diese Routine arbeitet wie $F817. Es wird nur statt "press play on tape" "press record & play on tape" ausgegeben.

STOP-Taste

Einsprungadresse : $F8DO (63696)

Diese Routine wird sofort wieder verlassen, wenn die Stop-Taste nicht gedrückt ist.

Ist sie gedrückt, so wird der Band-Motor ausgestellt, die IRQ-Routine wieder hergestellt, das Carry-Flag gesetzt und dann erst zurückgesprungen. Wichtig ist noch, daß in dieser Routine zwei PLA-Befehle auftauchen, d.h. daß bei gedrückter Stop-Taste die erste Rücksprung-Adresse gelöscht wird.

Rekorderbetrieb beenden

Einsprungadresse : $FC93 (64659)

Diese Routine beendet die Kommunikation mit dem Rekorder, d.h. der Bildschirm wird wiederhergestellt, der Rekorder ausgeschaltet und der Video-Controler wieder auf die Standardwerte gesetzt.

Rekordermotor ausschalten

Einsprungadresse $FCCA (64714)

Wenn Sie sich das Betriebssystem einmal anschauen, wird Ihnen vielleicht auffallen, daß die Adressen $FF81-$FFF3 (65409-65523) lauter Aufrufe verschiedener Unterprogramme sind. Doch wieso haben die Programmierer des Betriebssystems diese Sprungtabelle geschaffen, da man die Routinen doch auch direkt anspringen könnte?

Da diese Tabelle bei allen Commodore-Computer identisch ist ($FF81 z.B. ist immer der Aufruf zum Video-Reset, nur die Adresse, zu der gesprungen wird, ist verschieden), kann man Programme, die diese Routinen benutzen, ganz einfach von einem Computer auf ein anderes Modell übernehmen. Es kommt ja auf die Ansprungadresse in der Tabelle an, nicht, wohin der Computer dann tatsächlich springt.
Sie sollten diese Routinen oft benutzen, da es damit einerseits einfacher ist, Ihr Programm auf einen anderen Rechner von Commodore umzustellen und andererseits, da diese Routinen, wie Sie noch sehen werden, vielseitig verwendbar sind.
Diese Tabelle heißt "Kernal".
In der folgenden Aufzählung sämtlicher Adressen ist immer als erstes die Kernal-Adresse, also die Adresse die Sie anspringen müssen, und dann die Adresse, die schließlich vom Computer angesprungen wird, aufgeführt.

Kernal-Adresse: $FF81 (65409)
Funktion : Video-Reset
Tatsächliche Sprungadresse : $FF5B
Diese Routine stellt den Bildschirm wieder auf seine Standard-Werte. Es wird zu einem Programm ab Adresse $FF5B (66571) gesprungen:

```
FF5B  JSR $E518
FF5E  LDA $D012
FF61  BNE $FF5E
FF63  LDA $D019
FF66  AND #$01
FF6B  JMP $EDDD
```

In Adresse $FF5B wird zu einem Unterprogramm gesprungen, das

a) den Cursor wieder auf seine Standard-Werte bringt,
b) Klein-Groß-Umschaltung wieder ermöglicht,
c) die Länge des Tastatur-Puffers auf 10 setzt,
d) den Bildschirm löscht und
e) den Cursor an den Bildschirm-Anfang springen läßt.

Von $FF5E-$FF61 wird geprüft, ob die Rasterzeile, die Zeile, die gerade auf dem Bildschirm geschrieben wird, zu Ende ist. Ist das nicht der Fall, so wird gewartet, bis dieser Zustand eintritt.
Sodann wird das Bit 0 von Speicherstelle $D019(53273) in die Adresse $02A6 (678) geschoben (durch AND #$01 werden Bits 1-7 gelöscht, nur Bit 0 wird entweder 1 oder 0). Dann wird zu einem Programm gesprungen, das prüft, ob eine PAL- oder eine NTSC-Version des Fernsehers vorliegt. Darüber entscheidet die Speicherstelle $02A6, die eben gesetzt wurde.
Probieren Sie es aus:

```
PRINT PEEK(53273) AND 1
```

Wenn Sie 1 erhalten, so haben Sie eine PAL-Version (16421 Zyklen). Bei Null liegt eine NTSC-Version (17048 Zyklen) vor.
Und wenn Sie etwas anderes bekommen, so haben Sie etwas falsch gemacht. Sie können aber davon ausgehen, daß bei

Ihnen eine 1 erscheint, da in Deutschland und den anderen westeuropäischen Ländern (außer Frankreich mit SECAM) PAL-Norm vorherrscht.
Die NTSC-Version (ein Vorläufer von PAL) dagegen ist in den USA verbreitet. Da der C-64 in beiden Ländern vertreten ist, wurden Routinen in dem Betriebssystem eingebaut, die ein problemloses Umstellen ermöglichen.

Kernal-Adresse: $FF84 (65412)
Funktion : CIA/s initialisieren
Tatsächliche Sprungadresse : $FDA3
Diese Routine setzt die CIA/s (die Ausgabe-IC/s) wieder auf die Standard-Werte. Auch dieses Unterprogamm stellt fest, ob eine PAL- oder eine NTSC-Version vorliegt, und legt die Zyklen danach fest.

Kernal-Adresse: $FF87 (65415)
Funktion : RAM löschen bzw. testen
Tatsächliche Sprungadresse : $FD50
Diese Routine löscht die Zero-Page (außer die Adressen $00 -$01), Page 2 und Page 3.
Außerdem wird der Zeiger des Kassetten-Puffers ($B2-$B3:178-179) auf seinen normalen Wert gebracht, so daß er bei $033C (828) beginnt. Als Weiteres wird (ab $0400 :1024) auf das RAM-Ende geprüft. Diese Adresse wird in Low- und High-Byte zerlegt, und in die Adressen $0283-$0284 (643-644) geschrieben. Dann werden noch der RAM-Anfang auf $0800 (2048) und der Video-RAM-Start auf $0400 (1024) gelegt. In dieser Routine ist eine andere verborgen, die gut benutzt werden kann. Mit ihrer Hilfe kann man leicht die BASIC-Ram-Obergrenze festlegen.
eee : $FE20
X-Register: Low-Byte
Y-Register: High-Byte

Auch ein Lesen der Obergrenze ist möglich:
eee : $FE27
X-Register: Low-Byte
Y-Register: High-Byte

Kernal-Adresse: $FF8A (65418)
Funktion : I/O initialisieren
Tatsächliche Sprungadresse : $FD15
Diese Routine setzt die I/O-Einheit wieder auf die Standardwerte.

Kernal-Adresse: $FF8D (65421)
Funktion : I/O-Vektoren initialisieren
Tatsächliche Sprungadresse : $FD1A
Bei Ansprung dieser Routine werden die Adressen $0314-$0333 (788-819) wieder auf die Normal-Werte gebracht.

Kernal-Adresse: $FF90 (65424)
Funktion : Status setzen
Tatsächliche Sprungadresse : $FE18
Diese Routine setzt den Status:

```
FE18  STA $9D  ;Flag für Direkt-Modus ( $80=Direkt, $00=
               Programm-Modus)
FE1A  LDA $90
FE1C  ORA $90
FE1E  STA $90
FE20  RTS
```

Kernal-Adresse: $FF93 (65427)
Funktion: Sekundär-Adresse nach Listen senden

Tatsächliche Sprungadresse : $EDB9

Diese Routine gibt die Sekundär-Adresse auf den IEC-Bus (serielle Ausgabe) aus. Der IRQ wird dabei unterbrochen. Die Sekundäer-Adresse muß im Akku mit übergeben werden. Mit Listen ist ein Floppy gemeint, das Daten empfangen soll. Wie man das einstellt, erfahren Sie später.

Kernal-Adresse: $FF96 (65430)

Funktion: Sekundär-Adresse nach Talk senden

Tatsächliche Sprungadresse : $EDC7

Diese Routine funktioniert wie $FF93, nur es wird die Sekundär-Adresse zu einem Floppy gesendet, das Daten schicken soll (Talk).

Kernal-Adresse: $FF99 (65433)

Funktion: RAM-Ende setzen-holen

Tatsächliche Sprungadresse : $FE25

Bei dieser Routine wird das RAM-Ende gesetzt (Carry-Flag gleich Null) oder gelesen (Carry-Flag gleich Eins). Beide Male werden folgende Register folgendermaßen benutzt.

X-Register : Low-Byte
Y-Register : High-Byte

Wie Sie benutzt werden (Lesen oder Schreiben), hängt vom Carry-Flag ab.

Kernal-Adresse: $FF9C (65436)

Funktion : RAM-Anfang setzen-holen

Tatsächliche Sprungadresse : $FE34

Diese Routine hat die arbeitet auf die gleiche Weise wie die Routine $FF99. In diesem Fall wird nur statt das RAM-Endes der RAM-Anfang behandelt.

Kernal-Adresse: $FF9F (65439)
Funktion : Tastatur-Abfrage
Tatsächliche Sprungadresse : $EA87

In diesem Unterprogramm werden die Tasten abgefragt, entschlüsselt und das entsprechende Zeichen wird ausgegeben. Diese Routine (allerdings nicht über die Kernal-Adresse, sondern direkt $EA87) wird auch von der IRQ-Routine benutzt.

Kernal-Adresse: $FFA2 (65442)
Funktion : Time-out Flag für IEC-Bus setzen
Tatsächliche Sprungadresse : $FE21

Dieses Unterprogramm setzt das Time-out-Flag ($0285 : 645):

```
FE21  STA $0285
FE24  RTS
```

Wie Sie sehen, muß das Flag im Akku mit übergeben werden.

Kernal-Adresse: $FFA5 (65445)
Funktion : Eingabe vom IEC-Bus (IECIN)
Tatsächliche Sprungadresse : $EE13

Diese Routine holt ein Zeichen von der Floppy. Es müssen allerdings andere Routinen, die die Verbindung ers herstellen, vorher angesprungen worden sein.

Kernal-Adresse: $FFA8 (65448)
Funktion : Ausgabe vom IEC-Bus (IECOUT)
Tatsächliche Sprungadresse : EDDD

Diese Routine gibt ein Zeichen (mit ATN-Signal) auf dem IEC-Bus aus. Wie bei der Routine $FFA5 (IECIN) müssen allerdings andere Routinen vorher angesprungen werden.

Kernal-Adresse: $FFAB (65451)
Funktion : UNTALK senden
Tatsächliche Sprungadresse : $EDEF

Diese Routine sendet ein UNTALK-Signal. Daraufhin wird jegliche Kommunikation mit einem aktiven Ein-Ausgabe-Gerät, daß gerade Daten zum Computer sendet, unterbrochen.

Kernal-Adresse: $FFAE (65454)
Funktion : UNLISTEN senden
Tatsächliche Sprungadresse : $EDFE

Diese Routine hat die gleiche Wirkung wie die Routine UNTALK (Adresse $FFAB), nur daß ein Gerät angesprochen wird, das gerade Daten vom Computer empfängt.

Kernal-Adresse: $FFB1 (65457)
Funktion : LISTEN senden
Tatsächliche Sprungadresse : $EDOC

Diese Routine ist das Gegenstück zur Routine UNLISTEN (Adresse $FFAE), da hier eine Kommunikation mit einem Gerät begonnen wird. Dabei muß vorher die Geräte-Nummer der Floppy in den Akkumulator geladen und in Adresse $BA(186) abgespeichert worden sein. Wollen Sie also zu einer Floppy Daten senden, so müssen die ersten 3 Zeilen folgendermaßen aussehen:

```
LDA #$08    ;8=Device-Number für Floppy
STA $BA     ;abspeichern
JSR $FFB1   ;LISTEN-Routine
```

Kernal-Adresse: $FFB4 (65460)
Funktion : TALK senden

Tatsächliche Sprungadresse : $ED09

Diese Routine arbeitet ähnlich wie LISTEN, nur wird hir der Floppy mitgeteilt, daß es Daten senden soll. Beide Routinen (TALK u. LISTEN) benutzen die gleiche Routine:

```
ED09  ORA #$40
EDOB  .BYTE $2C
EDOC  ORA #$20
EDOE  JSR $FOA4
```

In $ED09 ist der Einsprung für TALK.
IN $EDOC ist der Einsprung für LISTEN.
Wird TALK angesprungen, so wird die Geräte-Nummer mit 64 logisch-oder verknüpft, d.h. zu der Geräte-Nummer (die Bit 0-3 belegen kann) wird Bit 6 gesetzt. Danach wird durch einen Programmier-Trick $EDOC übersprungen, da sonst auch noch Bit 5 gesetzt worden wäre.
Wird LISTEN angesprungen, so wird die Geräte-Nummer mit 32 logisch oder-Verknüpft, so daß Bit 5 gesetzt wird.
Ab Adresse $EDOE geht es für beide Routinen gemeinsam weiter.
Sie erkennen, daß das Floppy bei gesetztem Bit

a) Bit 5 der Geräte-Adresse Daten empfängt (LISTEN)
b) Bit 6 der Geräte-Adresse Daten sendet

Kernal-Adresse: $FFB7 (65463)
Funktion : Status holen
Tatsächliche Sprungadresse : $FEO7

Diese Routine holt den Status in den Akkumulator und setzt ihn daraufhin auf Null. Diese Routine kann auch für die RS 232-Schnittstelle benutzt werden, wenn die Geräte-Adresse ($BA : 186) gleich zwei ist.

Kernal-Adresse: $FFBA (65466)
Funktion : Fileparameter setzen
Tatsächliche Sprungadresse : $FE00

Diese Routine legt alle Parameter für einen File fest. Der Unterroutine muß die logische File-Nummer, die Geräte-Nummer und die Sekundär-Adresse übergeben werden.

PARAMETER	REGISTER	WIRD ABGESPEICHERT IN
Logische Filenummer	Akku	$B8 (184)
Geräte-Nummer	X-Register	$BA (186)
Sekundär-Adresse	Y-Register	$B9 (185)

Kernal-Adresse: FFBD (65469)
Funktion : Filenamenparameter setzen
Tatsächliche Sprungadresse : $FDF9

In dieser Routine werden alle Parameter festgelegt, die einen Filenamen betreffen.

PARAMETER	REGISTER	WIRD ABGESPEICHERT IN
Länge des Namens	Akku	$B7 (183)
Adresse low	X-Register	$BB (187)
Adresse high	Y-Register	$BC (188)

Wie bei der Kernal-Routine $FFBA müssen die entsprechenden Werte im Akku, dem X- und dem Y-Register übergeben werden.

Kernal-Adresse: $FFCO (65472)
Funktion: OPEN
Tatsächliche Sprungadresse: Ergibt sich aus $031A-$031B
Normalerweise $F34A
Da dieser Befehl sehr wichtig ist, wird er hier aufgelistet und erklärt:

```
F34A   LDX $B8    : lädt die logische Filenummer
F34C   BNE $F351 ; ungleich null, also weiter
F34E   JMP $F70A ; gibt "not input file" aus
F351   JSR $F30F ; logische Filenummer schon vorhanden?
F354   BNE $F359 ; nein
F356   JMP $F6FE ; gibt "file open" aus
```

```
F359   LDX $98      ;Anzahl der offenen Files
F35B   CPX #$0A     ;mit 10 vergleichen
F35D   BCC $F362    ;kleiner als 10, alo weiter
F35F   JMP $F6FB    ;zu groß, also "too many files"
F362   INC $98      ;Anzahl um Eins erhöhen
F364   LDA $B8      ;logische Filenummer
F366   STA $0259,x  ;speichert sie in Tabelle ab
F369   LDA $B9      ;Sekundär-Adresse
F36B   ORA #60      ;Bit 6 und 5 für Floppy setzen
F36D   STA $B9      ;wieder abspeichern
F36F   STA $026D,X  ;Sekundär-Adresse in Tabelle speichern
F372   LDA $BA      ;Geräte-Nummer
F374   STA $0263,X  ;in Tabelle abspeichern
F377   BEQ $F3DB    ;Tastur, also Rücksprung
F379   CMP #$03     ;Bildschirm?
F37B   BEQ $F3D3    ;ja, also Rücksprung
F37D   BCC $F384    ;kein File auf IEC-Bus
F37F   JSR $F3D5    ;File auf IEC-Bus eröffnen
F382   BCC $F3D3    ;fertig
F384   CMP #$02     ;RS 232-Schnittstelle?
F386   BNE $F38B    ;nein, also Band
F388   JMP $F409    ;RS 232 OPEN
F38B   JSR $F7D0    ;Kassetten-Puffer-Anfang holen
F38E   BCS $F393    ;Band-OPEN weiter
F390   JMP $F713    ;gibt "illegal device number" aus
F393   LDA $B9      ;Sekundär-Adresse
F395   AND #$0F     ;Bits 4-7 löschen
F397   BNE $FBB8    ;Sekundär-Adresse; 0, also schreiben
F399   JSR $F817    ;Play-Taste abfragen
F39C   BCS $F3D4    ;Stop-Taste gedrückt, also Abbruch
F39E   JSR $F5AF    ;"searching (for name)" ausgeben
F3A1   LDA $B7      ;Länge des Filenamens
F3A3   BEQ $F3AF    ;kein Filename, also weiter
F3A5   JSR $F7EA    :sucht gewünschten Tape-Header
F3A8   BCC $F3C2    ;gefunden
F3AA   BEQ $F3D4    ;Abbruch
```

```
F3AC   JMP $F704    ;EOT, also "file not found"
F3AF   JSR $F72C    ;weiter suchen
F3B2   BEQ $F3D4    ;EOT, also Ende
F3B4   BCC $F3C2    ;gefunden
F3B6   BCS $F3AC    ;weiter suchen
F3B8   JSR $F838    ;wartet auf Record & Play-Taste
F3BB   BCS $F3D4    ;Stop-Taste gedrückt, also Abbruch
F3BD   LDA #$04     ;Kontroll-Byte für Tape-Header
F3BF   JSR $F76A    ;Header auf Band schreiben
F3C2   LDA #$BF     ;Zeiger auf Ende des Kassetten-Puffers
F3C4   LDY $B9      ;Sekundär-Adresse
F3C6   CPY #$60     ;mit 96 (Bit 5 und 6) vergleichen
F3C8   BEQ $F3D1    ;Sek.-Adresse gleich null, also weiter
F3CA   LDZ #$00
F3CC   LDA #$02     ;Kontroll-Byte für Datenblock
F3CE   STA($B2),Z   ;in Kassetten-Puffer schreiben
F3D0   TZA          ;Akku gleich Null
F3D1   STA $A6      ;Zeiger in Kassetten-Puffer
F3D3   CLC
F3D4   RTS
F3D5   File auf IEC-Bus eröffnen
```

Wie Ihnen beim Durchsehen der Routine vielleicht aufgefallen ist, wird von der OPEN-Routine vorausgesetzt, daß einige Parameter schon gesetzt worden sind.

1. Für den Filenamen:

   a) Länge ($B7 ; 183)
   b) Adresse low ($BB ; 187)
   c) Adresse high ($BC ; 188)

2. Für das File:

   a) Logische Filenummer ($B8 ; 184)
   b) Sekundär-Adresse ($B9 ; 185)
   c) Geräte-Nummer ($BA ; 186)

Für das Setzen dieser Parameter sind in Kernal schon zwei Routinen vorgesehen:

a) Fileparameter setzen : $FFBA
b) Filenamenparameter setzen : $FFBD

Diese beiden Routinen müssen also vor dem Aufruf der OPEN-Routine angesprungen werden.
Ein Beispiel:
Sie wollen ein File mit dem Namen "$" (Directory) auf dem Floppy eröffnen:

```
     LDA #$01   ;Länge des Filenamens
     LDX #$D0   ;Adresse low
     LDY #$FF   ;Adresse high
     JSR $FFBD  ;Filenamenparameter festlegen
-----------------------------------------------
     LDA #$01   ;logische Filenummer
     LDY #$00   ;Sekundär-Adresse
     LDX #$08   ;Geräte-Nummer
     JSR $FFBA  ;Fileparameter festlegen
-----------------------------------------------
     JSR $FFC0  ;OPEN
```

Zur Erklärung:
Die Adresse des Filenamens ist $FFD0. Auf diese Adresse sind wir gekommen, da der Inhalt dieser Speicherstelle den Dezimalwert 36 enthält, was dem ASCII-Wert von "$" entspricht.

Kernal-Adresse: $FFC3 (65475)
Funktion : CLOSE
Tatsächliche Sprungadresse : Vektor $031C-$031D (796-797)

Normalerweise: $F291

Auch diese Routine ist sehr wichtig. Sie ist das Gegenstück zur OPEN-Routine. Bei der CLOSE-Routine muß aber nur ein Wert angegeben werden: Die logische Filenummer. Sie kennen das wahrscheinlich von BASIC her. Da heißt es auch nur "CLOSE 1", wenn das File mit der logischen Filenummer 1 geschlossen werden soll. Vor dem Anspringen der OPEN-Routine muß der Akkumulator mit der logischen Filenummer des Files, das geschlossen werden soll, geladen worden sein. Wollen Sie also das File mit der Nummer 10 schließen, so muß das in Maschinensprache heißen:

```
LDA #$0A
JSR $FFC3
RTS
```

Zu der Routine ist weiter nichts zu sagen. Für den Anwender ist nur wichtig, daß sie funktioniert. Kurz erklärt werden soll nur noch, wie der Computer die anderen Werte (Geräte-Adresse etc.) bekommt:
Er merkt sich immer die Anzahl der gerade offenen Files (in Adresse $98 : 158). Alle anderen Werte sind alle in Tabellen untergebracht.

601-610 : Tabelle für logische Files
611-620 : Tabelle für Geräte-Nummer
621-630 : Tabelle für Sekundär-Adresse

Zu Anfang sind in allen drei Tabellen die Werte Null.
Nehmen wir jetzt an, es wird ein File mit der Nummer 1, Sekundär-Adresse 0 und Geräte-Nummer 8 eröffnet. Automatisch wird dann auch die Anzahl der logischen Files um einen erhöht, so daß in Adresse $98 (158) der Wert 1 steht. Die drei Tabellen sehen dann folgendermaßen aus:

	1	2	3	4	5	6	7	8	9	10	
601	1	0	0	0	0	0	0	0	0	0	Filenummer
611	0	0	0	0	0	0	0	0	0	0	Sekundärnummer
621	8	0	0	0	0	0	0	0	0	0	Gerätenummer

Nehmen wir an, es wird jetzt auch noch ein File mit der logischen Nummer 3, Sekundärnummer 1 und Gerätenummer 1 eröffnet:

	1	2	3	4	5	6	7	8	9	10	
601	1	3	0	0	0	0	0	0	0	0	Filenummer
611	0	1	0	0	0	0	0	0	0	0	Sekundärnummer
621	8	1	0	0	0	0	0	0	0	0	Gerätenummer

In dem gleichen Sinne werden alle anderen Files abgespeichert.
Will der Computer jetzt die anderen Werte zu dem File mit der logischen Nummer 3 haben, so geht er zuerst einmal die Tabelle 601-610 durch. Ein Zähler wird jedesmal erhöht, wenn ein Platz weitergesprungen wird. Hat der Computer die logische Filenummer gefunden, so merkt er sich den Zähler. Da dieser Zähler auch auf die anderen Tabellen zutrifft (d.h. die anderen Tabellen haben den Eintrag zu diesem File am gleichem Platz), können die entsprechenden Werte ganz einfach aus den entsprechenden Tabellen gelesen werden.

Kernal-Adresse: $FFC6 (65478)
Funktion : Eingabegerät setzen (CHKIN)
Tatsächliche Sprungadresse : Vektor $031E-$031F (798-799)
Normalerweise: $F20E

Diese Routine setzt ein Eingabegerät fest. Die Festlegung (Gerät, etc.) erfolgt durch ein File. Die logische Filenummer muß im X-Register mit übergeben werden. Ist kein File mit dieser Nummer eröffnet, so wird "file not open"

ausgegeben. Der Computer sucht sich dann die entsprechende Gerätenummer aus der Tabelle und schreibt sie in die Speicherstelle $99 (153).
Sollte ein Floppy, also der IEC-Bus, angesprochen werden, so wird zusätzlich noch ein TALK-Signal gesendet. Daraufhin wird der Status getestet. Ist er nicht o.k., so wird "device not present" ausgegeben, da das Gerät nicht geantwortet hat und so folglich auch nicht da ist.
Wird das Band als Eingabe-Gerät gesetzt (Gerätenummer Eins) und die Sekundär-Adresse ist ungleich Null, wird "not input file" ausgegeben, da ein Output-File (Sekundär-Adresse Eins meint schreiben) ja kein Eingabegerät festsetzen kann.
Sie können diese Routine auch anspringen, ohne daß ein File eröffnet wurde. Dann müssen Sie nur schon die entsprechenden Werte (Gerätenummer, Sekundär-Adresse) in die entsprechenden Speicherstellen ($BA;186 : $B9;187) geschrieben haben. Die Einsprungadresse heißt dann F219 (61977).
Die Nummer des Eingabegeräts kommt in $99 (153).

Kernal-Adresse: $FFC9 (65481)
Funktion: Ausgabegerät setzen (CKOUT)
Tatsächliche Sprungadresse: Vektor $0320-$0321 (800-801)
Normalerweise: $F250

Auch bei dieser Routine muß die logische Filenummer wieder im X-Register mit übergeben werden.
Ist kein File mit dieser Nummer eröffnet, so gibt der Computer "file not open" aus. Soll ein File, das die Datasette anspricht, das Ausgabegerät bestimmen, hat aber die Sekundär-Nummer Null (Lesen), so wird "not output file" ausgegeben.
Auch diese Routine können Sie so verwenden, daß ein Ausgabegerät gesetzt wird, ohne daß Sie ein File öffnen müssen.
Geben Sie die gewünschten Parameter (Geräte-Adresse und Sekundär-Adresse) in $BA (186) und $B9 (185) und springen

Sie die Routine ab $F25B (62043) an.
Wird bei dieser Routine ein Floppy angesprochen, so wird ein LISTEN-Signal gesendet, der Status abgefragt und ggf. "device not present" ausgegeben.
Die Nummer des Ausgabegeräts kommt in $9A (154).

Kernal-Adresse: $FFCC (65484)
Funktion: I-O zurücksetzen (CLRCH)
Tatsächliche Sprungadresse : Vektor $0322-$0323 (802-803)
Normalerweise: $F333

Durch diese Routine kann man einen gerade aktiven I/O-Kanal auf dem IEC-Bus schließen.

```
F333   LDX #$03    ;3=Bildschirm
F335   CPX $9A     ;Nummer des Ausgabegerätes
F337   BCS $FB3C   ;kleiner als 3
F339   JSR $EDFE   ;UNLISTEN zur Floppy senden
F33C   CPX $99     ;Nummer des Eingabegerätes
F33E   BCS #F343   ;kleiner als 3
F340   JSR $EDEF   ;UNTALK zur Floppy senden
F343   STX $9A     ;Ausgabe wieder auf Bildschirm
F345   LDA #$00    ;0=Tastatur
F347   STX $99     ;Eingabe wieder von Tastatur
F349   RTS
```

Diese Routine arbeitet folgendermaßen:
Sie schaut zuerst nach, ob das Ausgabegerät der IEC-Bus ist. Ist dies der Fall, so wird zu einer Routine gesprungen, die den Datenverkehr abbricht (UNLISTEN). Ist dies nicht der Fall, so wird geschaut, ob das Eingabegerät das Floppy ist. Wenn ja, so wird ein UNTALK-Signal zur Floppy gesendet, damit das Floppy weiß, daß es aufhöhren soll, Daten zum Computer zu senden.
Schließlich wird noch bei beiden Möglichkeiten (Floppy als Ausgabegerät oder Floppy als Eingabegerät) der Bildschirm

wieder als Ausgabegerät und die Tastatur wieder als Eingabe gerät gesetzt.

Kernal-Adresse: $FFCF (65487)
Funktion: Eingabe eines Zeichens (BASIN)
Tatsächliche Sprungadresse : Vektor $0324-$0325 (804-805)
Normalerweise: $F157

Mit Hilfe dieser Routine kann man ein Zeichen von einem beliebigen Eingabegerät holen. Das geholte Zeichen befindet sich nach Rücksprung im Akku. Da diese Routine nicht so einfach ist, einige Beispiele:
Sie wollen einen String in einem Bereich (wir nehmen den Bereich ab $033C) ablegen. Der String soll über die Tastatur eingegeben werden. Das Programm legen wir ab dem Bereich $6000 (24576) in den Speicher.

```
    LDX #$00      ;Zähler auf Null setzen
  * JSR $FFCF     ;BASIN (Zeichen holen)
    STA $033C,X   ;Zeichen ablegen
    INX           ;Zähler erhöhen
    CMP #$0D      ;Return?
    BNE  *        ;nein
    RTS           ;ja
```

Als BASIC-Loader:

```
10 FORY=0TO13:READA:POKE24576+Y,A:NEXTY:SYS24576
20 DATA162,0,32,207,255,157,60,3,232,201,13,208,245,96
```

Sobald Sie das Programm eingegeben und RUN getippt haben, erscheint ein Cursor auf dem Bildschirm. Nun können Sie Ihre Zeichen eingeben. Abgebrochen wird das Programm durch Eingabe von RETURN.
Testen Sie nun, ob Ihr String auch richtig abgelegt wurde:
L ist die Länge des von Ihnen eingegebenen Strings. Diese

Länge darf 192 nicht überschreiten, da dann der Kassetten-Puffer voll ist.

```
FORY=0TOL:PRINTCHR$(PEEK(828+Y));:NEXTY
```

Wenn Sie alles richtig gemacht haben, so erscheinen Ihre eingegebenen Zeichen (auch Cursorfunktionen) auf dem Fernseher. Doch diese Routine kann nicht nur im Zusammenhang mit der Tastatur benutzt werden. Genauso gut können Daten von der Floppy, vom Band oder der RS 232-Schnittstelle übernommen werden. Dabei müssen Sie aber zuerst das Eingabegerät verändern.
Gehen Sie so vor:

Nehmen wir an, Sie wollen Daten von der Floppy (die schon wartet) übernehmen:

```
a) Parameter setzen:              LDA #$01  ;logische File-Nr.
                                  LDX #$08  ;Gerätenummer
                                  LDY #$00  ;Sekundär-Adresse
                                  JSR $FFBA ;Parameter setzen

b) Filenamenparameter setzen: LDA #$xx  ;Länge des Namens
                                  LDX #$y1  ;Adresse low
                                  LDY #$y2  ;Adresse high
                                  JSR $FFBD ;Parameter setzen

c) OPEN:                          JSR $FFC0 ;OPEN-Routine

d) Setzen des Eingabegeräts: LDX #$01  ;logische File-Nr.
                                  JSR $FFC6 ;CHKIN

e) Holen der Daten aus der geöffneten Datei:
   BASIN

f) Falls nicht alle Daten übernommen sind, e)

g) Standard-Werte einschalten:JSR $FFCC  ;CLRCH

h) Datei schließen :          LDA #$01  ;logische File-Nr.
                                  JSR $FFC3 ;CLOSE
```

Wenn Sie es ersteinmal verstanden haben, können Sie dieses System für alle Arten von Dateien einsetzen.
Ein Beispiel dafür finden Sie in dem Kapitel FLOPPY-TRICKS.

Kernal-Adresse: $FFD2 (65490)
Funktion : Ausgabe eines Zeichens (BSOUT)

Tatsächliche Sprungadresse : Vektor $0326-$0327 (806-807)
Normalerweise : $F1CA

Genauso wie ein Zeichen eingegeben wird (BASIN-Routine), kann auch ein Zeichen ausgegeben werden.
Das geschieht durch diese Routine.
Am besten gleich ein Beispiel:

```
    LDX #$00     ;Zähler auf Null
  * LDA $FOBE,X  ;Zeichen lesen
    JSR $FFD2    ;und ausgeben (BSOUT)
    INX          ;Zähler erhöhen
    CPX #$09     ;schon 9?
    BNE  *       ;nein
    RTS          ;ja
```

Diese Routine holt den Bereich $FOBE-$FOC6 (61630-61638 (Betriebssystem-Meldung "I/O ERROR")) der Reihe nach in den Akkumulator und gibt ihn mit Hilfe der BSOUT-Routine auf dem Bildschirm aus.
Als Basic-Loader:

```
10 FORX=0TO13:READA:POKE24576+X,A:NEXTX:SYS24576
20 DATA162,0,189,190,240,32,210,255,232,224,9,208,245,96
```

Bei RUN erhalten Sie die Meldung "I/O ERROR" auf dem Bildschirm. Wie Sie sehen, ist auch die BSOUT-Routine leicht zu bedienen. Natürlich kann auch sie, wie die BASIN-Routine, für andere Geräte genutzt werden. Die Schritte dazu sind identisch zur BASIN-Routine, nur

a) die Sekundär-Adresse muß Eins sein.
b) statt der CHKIN-Routine muß die CKOUT-Routine angesprungen werden, da ja ein Ausgabe-Gerät festgelegt werden soll.
c) statt der BASIN-Routine muß natürlich die BSOUT-Routine angesprungen werden.

Diese beiden Routinen sind sehr leistungsstark und können oft benutzt werden, da sie für alle Arten der Ein/Ausgabe geeignet sind.

Kernal-Adresse: $FFD5 (65493)
Funktion: LOAD
Tatsächliche Sprungadresse: $F49E

Dies ist die Load-Routine. Um Load auszuführen, ist es aber nötig, mehrere Parameter zu übergeben.
Vor dem Anspringen der Load-Routine muß folgendes gesetzt worden sein:

a) $BA (186) : Gerätenummer
b) $B7 (183) : Länge des Filenamens (ist bei Gerätenummer 8 unbedingt erforderlich, kann sonst Null sein)
c) $B9 (185) : Sekundär-Adresse
d) $BB-$BC : Low-High-Byte auf den Filenamen (wenn $B7=0 nicht erforderlich)
e) X-Register: Low-Byte der Startadresse(kommt in $C3ö195Ä)
f) Y-Register: High-Byte der Startadresse (kommt in $C4)
g) Akku : Load-Verify-Flag (0=Load,1=Verify); kommt in $93 (147)

In Adresse $F4A2 der Load-Routine wird ein indirekter Sprung durchgeführt:

```
JMP ($0330)
```

Normalerweise ergibt das $F4A5. Dieser Vektor kann aber auch auf eigene Load-Routinen gesetzt werden (siehe KAPITEL Tips & Tricks).

Kernal-Adresse: $FFD8 (65496)
Funktion: SAVE
Tatsächliche Sprungadresse : $F5DD

Auch für die Save-Routine müssen wieder einige Werte gesetzt werden:

a) $AE-$AF (174-175) : Endadresse
b) $C1-$C2 : Startadresse
c) $BA : Gerätenummer
d) $B9 : Sekundär-Adresse (wird bei IEC-Save automatisch auf Eins gesetzt)
e) $B7 : Länge des Filenamens (kann bis auf bei IEC-Save Null sein)
f) $BB-$BC : Adresse des Filenamens (wenn $B7=0 irrelevant)

In einer Schleife werden dann der Reihe nach alle Bytes von Startadresse bis zur Endadresse zum jeweiligen Gerät gebracht.

Kernal-Adresse: $FFDB (65499)
Funktion: Zeit setzen
Tatsächliche Sprungadresse: $F6E4

Der Akkumulator, das X- und das Y-Register setzen die Uhrzeit. Wie Sie sicherlich wissen, ist die Uhrzeit in den Adressen $A0-$A2 (160-162) untergeracht (und zwar ist TI PEEK(160) * 65525 + PEEK(161) * 256 + PEEK(162)).
Durch diese Routine können Sie nun die Uhrzeit beliebig stellen.

Der Akku kommt in Adresse $A2 (162, Low-Byte der Uhr)
Das X-Register kommt in Adresse $A1 (161, Mid-Byte der Uhr)
Das Y-Register kommt in Adresse $A0 (160,High-Byte der Uhr)

Kernal-Adresse: $FFDE (65502)
Funktion: Zeit holen
Tatsächliche Sprungadresse: $F6DD

Nach dem Rücksprung enthalten die verschiedenen Register folgende Werte:

REGISTER	ADRESSE	FUNKTION
Akku	$A2 (162)	Niedrigstes Zeit-Byte
X	$A1 (161)	Mittleres Zeit-Byte
Y	$A0 (160)	Höchstes Zeit-Byte

Kernal-Adresse: $FFE1 (65505)
Funktion: STOP-Taste abfragen
Tatsächliche Sprungadresse: Vektor $0328-$0329 (808-809)
Normalerweise: $F6ED

Diese Routine funktioniert nur mit Hilfe der IRQ-Routine. In einem Unterprogramm, das von der IRQ-Routine aus angesprungen wird, wird bei gedrückter STOP-Taste ein Flag ($91 (145)) gesetzt. In diesem Kernal-Unterprogramm wird nun das Flag abgefragt. Ist der richtige Inhalt vorhanden ($7F (127)), dann wird eine andere Kernal-Routine, CLRCH ($FFCC) durchgeführt. Außerdem wird der Tastaturpuffer geleert, indem eine Null in die Adresse $C6 (198) geschrieben wird. Wird der Teil der IRQ-Routine, der die Stop-Taste abfragt, durch POKE 788,52 außer Betrieb gesetzt, so funktioniert auch diese Routine nicht mehr, außer man führt den Sprung zu dem Unterprogramm, das von der IRQ-Routine aus angesprungen wird, vor dem Sprung zu dieser Kernal-Routine durch.

Kernal-Adresse: $FFE4 (65508)
Funktion: GETIN
Tatsächliche Sprungadresse: Vektor $032A-$032B (810-811)
Normalerweise: $F13E

Diese Routine holt ein Zeichen.
Anhand der Speicherstelle $99 (aktives Eingabegerät (153)) springt der Computer zu verschiedenen Routinen:

$99 (153)	Eingabegerät	Sprungadresse
0	Tastatur	$E5B4
2	RS 232-Schnittstelle	$F086
anders	Floppy-Band etc.	$F166

Wird ein Zeichen von der Tastatur geholt, so sieht die Routine folgendermaßen aus:

```
E5B4  LDY $0277    ;erstes Zeichen in Y-Register
E5B7  LDX #$00     ;Zähler
E5B9  LDA $0278,X  ;Zeichen holen
E5BC  STA $0277,X  ;und eine Stelle vorher ablegen
E5BF  INX          ;Zähler erhöhen
E5C0  CPX #$C6     ;Anzahl der Zeichen
E5C2  BNE $E5B9    ;noch nicht alle, also weiter
E5C4  DEC $C6      ;Zeichenanzahl um Eins erniedrigen
E5C6  TYA          ;1.Zeichen in Akku holen
E5C7  CLI          ;Interrupt wieder erlauben
E5C8  CLC
E5C9  RTS
```

Diese Routine holt sich das erste Zeichen aus dem Tastatur-Puffer (631-640), schiebt die anderen alle eins auf und erniedrigt die Zahl um einen.
Der Interrupt wurde verhindert, damit der Tastatur-Puffer nicht noch kurzfristig geändert wird (in der IRQ-Routine wird die Tastatur abgefragt, siehe Kapitel INTERRUPTS).

Kernal-Adresse: $FFE7 (65511)
Funktion: Schließen aller offenen Files (CLALL)
Tatsächliche Sprungadresse: Vektor $032C-$032D (812-813)
Normalerweise: $F32F

Diese Routine benutzt eine andere Kernal-Routine, die CLRCH-Routine ($FFCC). Es werden nur zwei Befehle

vorweggenommen:

```
F32F   LDA #$00
F331   STA $98
```

Damit werden alle offenen Files geschlossen ($98 (158) = Anzahl der offenen Files).
Ab Adresse $F333 geht es mit der CLRCH-Routine weiter.

Kernal-Adresse : $FFEA (65514)
Funktion : Zeit erhöhen-STOP-Taste abfragen
Tatsächliche Sprungadresse : $F69B

Diese Routine wird auch von der IRQ-Routine benutzt, allerdings wird direkt die Adresse $F69B angesprungen.
In dieser Routine wird die Uhrzeit erhöht.
Außerdem wird nachgeschaut, ob der Wert für 24h erreicht worden ist. Ist das der Fall, so wird die Uhr wieder auf Null gestellt.
Weiterhin wird getestet, ob die Stop-Taste gedrückt ist. Ist sie es, so wird ein Flag ($91 (145)) gesetzt.
Da diese Routine wie gesagt von der IRQ-Routine genutzt wird, fällt auf, daß in der IRQ-Routine nur die Stop-Taste abgefragt, nicht aber zu einer Routine gesprungen wird. Das hat einen großen Vorteil:
Je nachdem, was gerade ausgeführt wird, kann zu einer anderen Routine gesprungen werden. Ist der Computer z.B. gerade im Lade-Modus, und die Stop-Taste wird gedrückt, so wird zu einer anderen Routine gesprungen, als wenn der Computer im Programm-Modus gewesen wäre.

Kernal-Adresse: $FFED (65517)
Funktion: Zeilen-Spalten-Anzahl holen
Tatsächliche Sprungadresse: $E505

Diese Routine holt die Spalten-Anzahl (40) in das X-Register und die Zeilen-Anzahl (25) in das Y-Register:

```
E505   LDX #$28    ;Spalten-Anzahl
E507   LDY #$19    ;Zeilen-Anzahl
E509   RTS
```

Kernal-Adresse : $FFF0 (65520)
Funktion : Cursor setzen-holen
Tatsächliche Sprungadresse : $E50A

Ist das Carry-Flag gesetzt, so wird die Cursorposition geholt, sonst wird sie gesetzt.

```
E50A   BCS $E513    ;setzen
E50C   STX $D6      ;Zeile
E50E   STY $D3      ;Spalte
E510   JSR $E56C    :Cursor setzen
E513   LDX $D6
E515   LDY $D3
E517   RTS
```

Wie Sie sehen, ist

a) beim Holen der Cursor-Position die Zeile ins X-Register und die Spalte ins Y-Register.
b) beim Setzen der Cursor-Position die Zeile ins X-Register und die Spalte ins Y-Register zu übergeben.

Kernal-Adresse : $FFF3 (65523)
Funktion : Startadresse I/O-Baustein holen
Tatsächliche Sprungadresse : $E500

Nach Ansprung dieser Routine ist im X-Register das Low-Byte ($00) der Startadresse des I/O-Bausteins und im Y-Register das High-Byte ($DC).
Die gesamte Routine sieht folgendermaßen aus:

```
E500   LDX #$00    ;Low-Byte
E502   LDY #$DC    ;High-Byte
E504   RTS
```

Diese Routine und die Routine zum Holen der Zeilen-/ Spalten-Anzahl mag auf den ersten Blick unsinnig erscheinen. Man sieht den Sinn aber schnell ein, wenn man noch einmal überdenkt, wozu der Kernal denn geschaffen ist: Zum Umsetzen von Programmen auf einen anderen Computer.
Wie Sie sicherlich schnell einsehen werden, kann man kein vernünftiges Programm schreiben, ohne daß Bildschirmformat des Computers zu kennen. Mit Hilfe dieser Routinen kann man das Programm so schreiben, daß es sich selbsttätig nach dem Format erkundigt.

Wir hoffen, daß dieser Ausflug in den Bereich des Kernal lehrreich für Sie war, und daß Sie als Maschinensprache-Programmierer doch die ein oder andere Routine einmal benutzen können.

11. DER SPEICHER

WIE SPEICHERT DER COMPUTER EINE BASIC-ZEILE ?

Auf den folgenden Zeilen wollen wir Ihnen einige Einzelheiten die Abspeicherung von BASIC-Zeilen betreffend aufzeigen.
Um die BASIC-Zeilen überhaupt verarbeiten zu können, muß das erste Byte des BASIC-Speichers "0" enthalten. Im Normalfall fängt der BASIC-Speicher bei Adresse 2049 an. Also muß Adresse 2048 eine Null enthalten. POKEn wir in diese Adresse eine 1, dann druckt der Computer bei NEW oder bei RUN die Fehlermeldung SYNTAX ERROR aus (aber Vorsicht!! NEW führt der Computer trotzdem aus!).
Wir haben ein kleines Programm geschrieben, das die Speicherung einer BASIC-Zeile verdeutlichen soll. Es zeigt in der oberen Zeile des Bildschirms ständig an, wieviele Bytes des BASIC-Speichers von der letzten Zeile verbraucht wurden.
Nachdem Sie es eingegeben und mit RUN gestartet haben, erscheint in der oberen Zeile des Bildschirms folgender Ausdruck:

```
00 BYTES BENUTZT
```

Dies bedeutet, daß die letzte Eingabe (seit dem letzten RETURN) kein Platz im BASIC-Speicher mehr verbraucht hat. BASIC-Speicherplatz wird nur bei Programm-Zeilen belegt.
Geben Sie jetzt ein:

```
100 PRINT
```

Nun zeigt das Programm folgende Aussage an:

```
06 BYTES BENUTZT
```

Von diesen 6 Bytes sind fünf sogenannte Verwaltungs-Bytes.

Das letzte Byte ist für den Befehlscode (Token) für PRINT. Die 5 Verwaltungs-Bytes unterteilen sich wiederum in:

a) zwei Bytes (die ersten beiden) für die Abfangsadresse der nächsten BASIC-Zeile (in Low-High-Byte).Die letzte Zeile zeigt auf den Wert Null.
b) zwei Bytes (die zweiten beiden) für die Zeilennummer. Auch diese beiden Bytes sind in Low - High-Byte unterteilt, so daß die Zeilen-Nummer 1000 die Werte 232/3 ergibt.
c) ein Byte für die Kennzeichnung des Zeilen-Endes. Dieses Byte muß den Wert Null haben, und es steht am Ende der entsprechenden Zeile.

Geben Sie nun

```
20 PRINT"WERNER"
```

ein. Hier werden 14 Bytes benötigt. Davon wieder die 5 bekannten Verwaltung-Bytes, ein Byte für den Token (PRINT) und die übrigen 8 für den Text (Werner gleich 6 und die Anführungszeichen jeweils ein Byte).
Geben Sie nun noch etwas ein:

```
30 PRINTCHR$(48)
```

Diesmal sind 11 Bytes erforderlich:

a) 5 Bytes für die Verwaltung
b) 1 Byte für den BASIC-Token "PRINT"
c) 1 Byte für den BASIC-Token "CHR$"
d) 4 Bytes für den "Text-Teil" "(48)".

Wenn Sie die Funktionen TAB und SPC verwenden, ist zu beachten, daß diese Befehle bereits das Zeichen "(" enthalten.

Die Zusammensetzung der Bytes einer BASIC-Zeile ist bei allen BASIC-Befehlen gleich. Als letztes Beispiel wollen wir dafür folgende Zeile zeigen:

```
40 POKE198,0
```

Und noch einmal eine Auflistung der benutzten Bytes:

a) 5 Bytes für die Verwaltung
b) 1 Byte für den BASIC-Token "POKE"
c) 5 Bytes für den Text (hier 198,0)

Und nun die Routine, zuerst den BASIC-Loader:

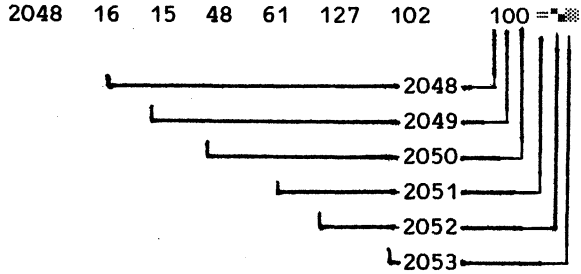
```
10 FOR I=0 TO 87
20 READ A
30 POKE 40704+I,A
40 S=S+A
50 NEXT I
60 IF S<>7590 THEN PRINT "FEHLER IN
   DATAS":END
70 PRINT"DATA O.K."
80 SYS 40704
100 DATA120,169,15,141,20,3,169,159,141,
21,3,133,56,88,96,169,48
110 DATA141,0,4,141,1,4,165,11,201,76,
240,18,56,201,10,144,8,238
120 DATA0,4,233,10,76,30,159,105,48,141,
1,4,162,2,189,69,159,157,0,4
130 DATA232,224,18,208,245,169,0,202,157
,0,216,208,250,76,49,234,32,2,25,20,5
140 DATA19,32,2,5,14,21,20,26,20,32,32,
32
```

Für Assembler-Freaks:

```
9F00   SEI
9F01   LDA #0F
9F03   STA $0314
9F06   LDA #9F
9F08   STA $0315
9F0B   STA $38
9F0D   CLI
9F0E   RTS
9F0F   LDA #$30
9F11   STA $0400
9F14   STA $0401
9F17   LDA $0B
9F19   CMP #$4C
9F1B   BEQ $9F2F
9F1D   SEC
9F1E   CMP #$0A
9F20   BCC $9F2A
9F22   INC $0400
9F25   SBC #$10
9F27   JMP $9F1E
9F2A   ADC #$30
9F2C   STA $0401
9F2F   LDX #$02
9F31   LDA $9F45,X
9F34   STA $0400,X
9F37   INX
9F38   CPX #$12
9F3A   BNE $9F31
9F3C   LDA #$00
9F3E   DEX
9F3F   STA $D800,X
9F42   BNE $9F3E
9F44   JMP $EA31
```

Die restlichen Daten stellen die Bildschirm-Code-Werte für den String "BYTES BENUTZT" dar.

Vorhin haben wir schon gesehen, wie der Computer eine BASIC-Zeile abspeichert. Das nun folgende, kurze BASIC-Programm ermöglicht es, die Speicherinhalte des Computers zu sichten. So kann man sich ganze (Basic-)Programme anschauen. Es ist auch sehr interessant und aufschlußreich, auf diese Art Teile des ROMs zu untersuchen.
Wenn Ihnen Maschinensprache-Monitore nicht fremd sind, kennen Sie sicher den Befehl "M" (Memory-Display). In Deutsch heißt das "Speicher-Anzeige". Dieses Programm führt genau diesen Befehl aus.
Zuerst werden Sie nach der Anfangs- und Endadresse gefragt. Nun werden auf dem Bildschirm die Anfangs-Adresse, und die PEEKs dieser Adresse und der fünf folgenden angezeigt. Am Ende der Zeile stehen noch die Bildschirm-Code-Zeichen dieser Speicher-Inhalte.
Beispielsweise die folgende (fiktive) ausgegebene Zeile:



Die Zeilen, die auf dem Bildschirm zu sehen sind, haben das Muster der Zeile mit der Zahl 2048 am Anfang. Das Gebilde unter dieser Zeile soll verdeutlichen, wie die Zahlen und Zeichen zueinander stehen.
Wie Sie sehen, erfolgt die Zahlenausgabe im dezimalen Zahlen-System, da dieser Monitor zum durchforsten von BASIC-Programmen gedacht ist.

Da das Programm nur eine Funktion auszuführen braucht, ist die Bedienung denkbar einfach. Laden Sie das Programm herein und starten Sie es dann. Der Computer meldet sich mit

BASIC-MONITOR

ANFANGSADRESSE?

Geben Sie nun die gewünschte Adresse (in dezimal) ein.

ENDADRESSE?

Geben Sie wieder die gewünschte Adresse ein. Der Computer gibt jetzt nach dem oben schon beschreibenen Muster die entsprechenden Werte aus. Wie Ihnen beim Ausprobieren vielleicht aufgefallen ist, druckt der Computer in der letzten Zeile auch dann 6 Werte aus, wenn eigentlich nach der eingegebenen Endadresse weniger Werte ausgegeben werden müßten.
Wenn Sie während eines längeren Durchlaufs den Wunsch haben das Programm zu stoppen, brauchen Sie nur auf F-1 zu drücken.

Einige interessante Start-Adressen:

2048 : Anfang des BASIC-Speichers. Hier befindet sich (normalerweise) der Monitor.
40964 : BASIC-Befehle

Nun erstmal der Monitor:

```
10 print"■■"chr$(14):poke53281,6:poke53280,6
20 print" ■          Basic-Monitor on           ■":print:print
30 input" Anfangsadresse ";a
40 input" Endaddresse    ";e
50 fori=atoestep6
60 printi;
70 foro=0to5
80 xp=peek(i+o):gosub200: rem print using
90 u=peek(214)*40+1024+33+o
100 pokeu,peek(i+o):poke54272+u,1
110 next o
120 print
130 nexti
140 input" ■       noch einmal (j/n)       "
150 geta$:ifa$="" then 150
160 ifa$="j"then run
170 end
190 :
200 rem print formatieren
210 xp$=str$(xp)
220 iflen(xp$)<4thenxp$=" "+xp$:goto220
240 printxp$;:return
```

Obwohl der Monitor in BASIC geschrieben ist, ist es möglich, ein anderes BASIC-Programm damit zu untersuchen. Dazu müssen

Sie aber einige Werte verändern.
Gehen Sie folgendermaßen vor:

- Versetzen Sie Ihren Computer in den Anschaltzustand
- Geben Sie ein:

```
POKE 44,150
```

  Statt der 150 können Sie auch eine andere Zahl zwischen 8 und 159 nehmen. Dadurch wird der BASIC-Anfang hochgesetzt, so daß sich der Monitor und das Programm, daß Sie untersuchen wollen nicht überschneiden.
- Geben Sie

```
POKE 256*PEEK(44),0
```

  ein. Dadurch wird eine Null an den BASIC-Anfang gesetzt, was unbedingt erforderlich ist.
- Laden Sie nun den BASIC-Monitor.
- Geben Sie nun

```
POKE 44,8
```

  ein. Dadurch wird der BASIC-Anfang wieder auf den Normalwert gebracht.
- Laden Sie nun das Programm, das Sie untersuchen wollen.
- Geben Sie nun wieder

```
POKE 44,(Ihren Wert)
```

  ein, um wieder zum Monitor zu gelangen.
- Starten Sie nun den Monitor mit RUN. Nun können Sie das Programm ganz einfach untersuchen (ab Adresse 2048).

Da der Monitor nicht lang und so einfach zu verstehen ist, können Sie Ihn ja mal durch andere Befehle ergänzen, um so z.B. BASIC-Programme mit dem Monitor verändern zu können.

Im Folgenden finden Sie ein kommentiertes Zeropage-Listing. Es soll Ihnen helfen, die Bedeutung unbekannter Adressen in fremden Listings zu klären. Beim Durchschmökern finden aber auch Sie vielleicht so manche interessante Anregung !
Wir wünschen Ihnen viel Spaß!

```
0000   0       6510 DATEN RICHTUNGSREGISTER FÜR PROZESSORPORT
0001   1 055   6510 PROZESSOR-PORT
----   -       Bit 0: RAM oder ROM von $A000 bis $BFFF
----   -       (BASIC-Interpreter) * POKE 1,54
----   -       Bit 1: RAM oder ROM von $E000 bis $FFFF
----   -       (Betriebssystem, KERNAL) * POKE 1,53
----   -       Bit 2: I/O oder ROM von $D000 bis $DFFF
----   -       (Charactergen.-Zugriff) * POKE 1,51
0002   2       NICHT BENUTZT
0003   3       VEKTOR LOW (Umrechnung Float-Fixed)
0004   4       VEKTOR HIGH
0005   5       VEKTOR LOW (Umrechnung Fixed-Float)
0006   6       VEKTOR HIGH
0007   7       SUCHZEICHEN
0008   8       FLAG (Hochkomma-, Gänsefüßchen-Mode)
0009   9       SPEICHER FÜR SPALTE BEIM TAB-BEFEHL
000A  10       FLAG (0=LOAD; 1=VERIFY)
000B  11       ZEIGER (Eingabepuffer); Anzahl der Elemente
000C  12       FLAG (Standard-DIM)
000D  13       TYPFLAG ($FF(255)=String, 00=numerisch)
000E  14       TYPFLAG ($80(128)=Integer, 00=Fließkomma)
000F  15       FLAG (DATA/LIST); Hochkommaflag bei LIST
0010  16       FLAG (FNx, Element)
0011  17       EINGABEFLAG (00=INPUT; 64=GET; 136=READ)
0012  18       VORZEICHEN BEI ATN  ( Flag für  Gleichheit bei
----  --       Vergleich)
```

```
0013  19      AKTIVES I/O-GERÄT
----  --      POKE 19,1:INPUT"OHNE FRAGEZEICHEN";A$:POKE19,0
----  --      (Ausschalten des Fragezeichens bei INPUT)
0014  20      INTEGER-ADRESSE LOW
0015  21      INTEGER-ADRESSE HIGH
0016  22      ZEIGER AUF TEMPORÄREN STRINGSTAPEL
----  --      POKE 22,35: LIST
----  --      (Listen ohne Zeilennummern)
0017  23      VEKTOR LOW (Letzter temporärer String)
0018  24      VEKTOR HIGH
0019  25      STAPEL FÜR TEMPORÄRE STRINGS
001A  26                  "
001B  27                  "
001C  28                  "
001D  29                  "
001E  30                  "
001F  31                  "
0020  32                  "
0021  33                  "
0022  34      BEREICH FÜR HILFSZEIGER (nutzbar für M-Prg.)
0023  35                  "
0024  36                  "
0025  37                  "
0026  38      BEREICH FÜR PRODUKT BEI MULTIPLIKATION
0027  39                  "
0028  40                  "
0029  41                  "
002A  42                  "
002B  43  01  VEKTOR LOW (Basic-Anfang)
----  --  --  An neuen BASIC-Anfang POKE (newadd)-1,0:NEW !!
002C  44  08  VEKTOR HIGH
002D  45      VEKTOR LOW (Variablen-Start)
002E  46      VEKTOR HIGH
002F  47      VEKTOR LOW (Beginn der Arrays (Felder) )
0030  48      VEKTOR HIGH
0031  49      VEKTOR LOW (Ende der Arrays)
```

```
0032  50      VEKTOR HIGH
0033  51      VEKTOR LOW (Beginn der Strings)
0034  52      VEKTOR HIGH
0035  53      VEKTOR LOW (Hilfszeiger für Strings)
0036  54      VEKTOR HIGH
0037  55      VEKTOR LOW (BASIC-Speicher-Ende)
----  --      Herabsetzen des verfügbaren BASIC-Speichers,um
----  --      beispielsweise  dort abgelegte  Assemblerprog-
----  --      gramme vor dem Überschreiben zu schützen.
0038  56      VEKTOR HIGH
0039  57      LOW-BYTE (aktuelle Zeilennummer)
----  --      PRINT PEEK(57) +256* PEEK(58) ergibt die aktu-
----  --      elle BASIC-Zeilennummer aus (Abfrage  funktio-
----  --      niert nur im Programm-Mode!)
003A  58      HIGH-BYTE
003B  59      LOW-BYTE (vorherige Zeilennummer)
----  --      PRINT PEEK(59) +256*PEEK(60) ergibt die voran-
----  --      gegangene BASIC-Zeilennummer (auch im  Direkt-
----  --      de)
003C  60      HIGH-BYTE
003D  61      LOW-BYTE (Nächstes Statement für CONT)
----  --      Zeiger auf Einsprungadresse bei CONT
003E  62      HIGH-BYTE
003F  63      LOW-BYTE (Augenblickliche DATA-Zeilennummer)
----  --      Enthält die augenblickliche Zeilennummer   für
----  --      DATA (In Verbindung mit READ;kann zum Ausgeben
----  --      der Zeilennummer bei eigenen   Fehlermeldungen
----  --      benutzt werden !)
0040  64      HIGH-BYTE
0041  65      VEKTOR LOW (Adresse des aktuellen DATA-Elemen-
----  --      tes. Zeigt auf die Anfangsadresse des nächsten
----  --      DATA-Elementes im BASIC-Speicher
0042  66      VEKTOR HIGH
0043  67      VEKTOR LOW (Zeiger auf Herkunft der Eingabe)
0044  68      VEKTOR HIGH
```

```
0045  69      REG 1 Variablenname (siehe  "Grundsätzliches
0046  70      REG 2 Variablenname  zur Variablenspeicherung
0047  71      VEKTOR LOW Variablenadresse
0048  72      VEKTOR HIGH
0049  73      LOW-BYTE (Variablenwert)
004A  74      HIGH-BYTE
004B  75      LOW-BYTE (Zwischenspeicher für Programmzeiger)
004C  76      HIGH-BYTE
004D  77      Maske für Vergleichsoperationen
004E  78      VEKTOR LOW (Zeiger für FN)
004F  79      VEKTOR HIGH
0050  80      Stringdescriptor  (verschieden genutzter   Ar-
0051  81            "            beitsbereich)
0052  82            "
0053  83            "
0054  84  76  Konstante $4C (JMP-Befehl für Funktionen)
0055  85      Sprungvektor für Funktionen (LOW)
0056  86                  "              (HIGH)
0057  87      Register für Arithmetik, Akku 3
0058  88            "
0059  89            "
005A  90            "
005B  91            "
005C  92      Register für Arithmetik, Akku 2
005D  93            "
005E  94            "
005F  95            "
0060  96            "
0061  97      Fließkomma-Akku 1, FAC
0062  98            "
0063  99            "
0064 100            "
0065 101            "
0066 102      Vorzeichen des FAC
0067 103      Zähler für Polynomberechnung (z.B. ab $E059)
0068 104      Rundungsbyte für FAC
```

0069	105	Fließkomma-Akku 2, ARG
006A	106	"
006B	107	"
006C	108	"
006D	109	"
006E	110	Vorzeichen des ARG
006F	111	Vergleich der Vorzeichen FAC/ARG
0070	112	Rundungsbyte für FAC

Kleine Anmerkung zu den vorangegangenen Speicherstellen:

Die "Formelauswertungsroutine" befindet sich im BASIC-Interpreter ab $AD9E.
Sie holt einen beliebigen Ausdruck und wertet ihn aus. Diese Auswertung geschieht in reellen Zahlen, eventuell vorkommende Integer (Ganzzahl) Variablen werden zunächst ins Fließkommaformat umgewandelt.
Aber nicht nur Zahlen, auch Stringparameter können von dieser Routine bearbeitet werden.
Um Stringparameter von anderen numerischen Variablen zu unterscheiden, wird ein entsprechendes Zeichen (Typflag) gesetzt (Adresse 13).
Numerischer Werte werden im FAC 1 (ab $61) zwischengespeichert. Da dies aber nicht für arithmetische Verknüpfungen (Subtraktion, Addition, etc.) ausreicht, gibt es einen weiteren Fließkomma-Akku (ab $69), ARG.
Das Ergebnis nach dem Aufruf der entsprechenden Routine steht wie das Argument im FAC.
Bei Stringauswertungen werden die Adressen $64 / $65 hinzugezogen, die einen Zeiger auf den Stringdescriptor bilden.
Dort stehen Informationen über Länge und Adresse des bearbeiteten Strings.
Die im Folgenden aufgeführten Routinen des Interpreters dürften gerade bei der Maschinensprache-Programmierung für die Auswertung beliebiger Ausdrücke interessant sein:

```
FAC = ARG + FAC (Addition)          $B86A  (47210)
FAC = ARG - FAC (Subtraktion)       $B853  (47187)
FAC = ARG * FAC (Multiplikation)    $BA28  (47656)
FAC = ARG / FAC (Division)          $BB12  (47890)

FAC = ARG ^ FAC (Potenzieren)      $BF7B  (49019)
```

Die Routine $B475 (46197) holt die Stringlänge in den Akku.
Die Adresse liegt im Low-High-Format im X- und Y-Register.

```
0071 113      VEKTOR LOW (Zeiger für Polynomauswertung)
0072 114      VEKTOR HIGH  Sie dienen als Zeiger auf den Po-
---- ---      lynomkoeffizienten ($E043) und als Zeiger  für
---- ---      Polynomgrad ($E059).
0073 115      CHRGET-ROUTINE (Assembler-ROM-Listing):
0074 116
0075 117      0073 INC $7A
0076 118      0075 BNE $0079 Zeiger in BASIC-Text erhöhen
0077 119      0077 INC $7B
0078 120      0079 LDA $HHLL
0079 121
007A 122      Programmzeiger LOW
007B 123      Programmzeiger HIGH
007C 124
007D 125      007C CMP "$3A  ":"
007E 126      007E BCS $008A
007F 127      0080 CMP "$20  " " Space übergehen
0080 128      0082 BEQ $0073
0081 129      0084 SEC
0082 130      0085 SBC "$30
0083 131      0087 SEC
0084 132      0088 SBC "$D0
0085 133      008A RTS
0086 134
0087 135      Die CHRGET-Routine liegt eigentlich im  ROM ab
```

```
0088 136      $E3A2, wird aber während des RESETS ins RAM ab
0089 137      $0073 kopiert. Die Routine funktioniert nur im
008A 138      RAM-Bereich. X- u. Y-Register werden nicht an-
---- ---      getastet.
---- ---      Carry=0 : Inhalt des gelesenen Bytes liegt  im
---- ---      Bereich $30 - $39 (Ziffern im ASCII-Code)
---- ---      Zero=1  : Zeichen = $00 oder $3A (":")
008B 139      letzter RND-Wert
008C 140                "
008D 141                "
008E 142                "
008F 143                "
0090 144      Statuswort ST (bei z.B. IEC-Bus-Routinen)
0091 145 10   Flag des Systems für STOP-Taste
0092 146      Zeitkonstante für Kassette (von $F92C gesetzt)
0093 147      LOAD/VERIFY-Flag ($00=LOAD; $01=VERIFY)
---- ---      Wird von LOAD-Routine gesetzt, während Adresse
---- ---      $0A (10) vom LOAD-Befehl gesetzt wird.
0094 148      Flag für zurückgestelltes Zeichen bei  IEC-Out
0095 149      Zurückgestelltes Zeichen (Puffer)
0096 150      End-Of-Tape (EOT) gefunden (Flag)
0097 151      Zwischenspeicher für Register
0098 152      Anzahl der offenen Dateien
---- ---      POKE 152,0 schließt alle Files,
---- ---      POKE 152,12 verhindert das öffnen von Files
0099 153 01   aktives Eingabegerät (1=Tastatur)
---- --- --   POKE 153,2 verhindert Tastatur-Eingaben
009A 154 03   aktives Ausgabegerät (3=Bildschirm)
---- --- --   POKE 154,1 hat dieselbe Bedeutung wie bei
---- --- --   Adresse 19 in bezug auf INPUT
009B 155 00   Paritätsbyte von Band
009C 156      Flag für Byte empfangen
009D 157      Direkt-Mode-Kontrolle ($00=Programm, $80=RUN)
---- ---      Ein mit GOTO gestartetes Programm bewirkt $80!
009E 158      Band Pass 1 Checksumme
009F 159      Band Pass 2 Korrektur
```

```
00A0 160      TIME 3 (Übertrag TIME 2)
00A1 161      TIME 2 (Übertrag TIME 1)
00A2 162      TIME 1 (Zähler für TI und TI$)
00A3 163      Bitzähler. Zählt bei $FB97 die acht Bits eines
---- ---      auszugebenden Bytes
00A4 164      Zyklen-Zähler ($A3=8 dann $A4=0)
00A5 165      Abwärtszähler schreiben auf Kassette
00A6 166      Zeiger in Bandpuffer (wird ein Zeichen in  den
---- ---      Kassettenpuffer geschrieben, erhöht sich diese
---- ---      Adresse um 1 und zeigt auf die nächste  freie
---- ---      Stelle. Ist der Puffer voll (192 Zeichen),wird
---- ---      das Zero-Flag gesetzt ($F80D).
00A7 167      Zwischenspeicher für LOAD/SAVE bei Kassette
00A8 168                  "
00A9 169                  "
00AA 170                  "
00AB 171                  "
00AC 172      Zeiger für Bandpuffer und Scrolling
00AD 173      HIGH-BYTE          "
00AE 174      Lowbyte der Endadresse des geladenen Programms
00AF 175      High-Byte          "
00B0 176 00   Zeitkonstanten für Band
00B1 177                  "
00B2 178      VEKTOR LOW (Verschieben des Kassettenpuffers)
00B3 179      VEKTOR HIGH
00B4 180      Bitzähler für Band
```

```
00B5 181      nächstes Bit für RS 232
00B6 182      Puffer für auszugebendes Byte
00B7 183      Länge des Programm-Namens
00B8 184      Zuletzt benutzte Dateinummer
00B9 185      Zuletzt benutzte Sekundäradresse
00BA 186      Zuletzt benutzte Gerätenummer
00BB 187      VEKTOR LOW  auf abgespeicherten Programmnamen
00BC 188      VEKTOR HIGH bei Floppy Disk
00BD 189      Zwischenspeicher für serielle Ein-/ Ausgabe
00BE 190      Anzahl der noch zu lesenden/schreibenden
---- ---      Blocks (Routine ab $FBCD)
00BF 191      Serieller Wortpuffer. Puffer für Adresse 189
00C0 192      Kassettenmotorflag
00C1 193      VEKTOR LOW  Eingabe/Ausgabe-Startadresse
00C2 194      VEKTOR HIGH (zeigt auf $A000);für SAVE-Routine
00C3 195      Endadresse für Ein/Ausgabe vom Bildschirm
00C4 196      High-Byte                "
00C5 197      Nummer der gedrückten Taste (64=keine Taste)
---- ---      10 IF PEEK(197)=64 THEN 10 wartet auf Tasten-
---- ---      druck (siehe auch Tabelle im Anhang).
00C6 198      Anzahl der Tasten, die aus dem Tastaturpuffer
---- ---      ausgegeben werden sollen(siehe Tastaturpuffer)
00C7 199      Revers(RVS)Flag  1=revers, 0=normal
---- ---      POKE 199,1: PRINT "DEMO"
00C8 200      Zeiger auf Zeilenende für Eingabe
00C9 201      Cursorzeile für Eingabe (dient nur als Puffer)
00CA 202      Cursorspalte für Eingabe          "
00CB 203      Nummer der gedrückten Taste  (64=keine Taste)
---- ---      scheinbar dieselbe Funktion wie Adresse  197.
---- ---      Für den Computer jedoch zeitverschobene Abfra-
---- ---      ge auf zwischenzeitlich gedrückte Tasten.
00CC 204      Cursor-Flag   0=Cursor ein  1=Cursor aus
---- ---      Gestattet das Einschalten des Cursors im Prg.
00CD 205      Zähler für Cursor blinken
00CE 206      Zeichen in Cursorposition
00CF 207      Einschaltflag 1=Cursor sichtbar 0=unsichtbar
```

```
---- ---     10IFPEEK(207)THEN10 wartet, bis Cursor sich in
---- ---     Aus-Phase befindet.
00D0 208     Eingabeflag  (z.B. $E65F, $F16A)
00D1 209     VEKTOR LOW  in aktuelles Video-RAM
00D2 210     VEKTOR HIGH
00D3 211     Eingabe der Cursorspalte für Eingabe-Cursor
---- ---     Cursor-Zeile siehe Adresse 214. Aufruf der
---- ---     Set-Routine: SYS 58732)
00D4 212     Hochkomma(")Flag  Benutzung beispielsweise bei
---- ---     ESCAPE-Routine. POKE 212,1: PRINT... gibt auch
---- ---     eventuell vorkommende Steuerzeichen aus.
00D5 213     Länge der Bildschirmzeile (39/79)
00D6 214     Eingabe der Cursorzeile (siehe Adresse 211 !)
00D7 215     div. Zwecke (letzte Taste, Puffer, Prüfsumme)
00D8 216     Anzahl der ausstehenden Inserts
00D9 217     MSB der Bildschirmzeilen-Anfänge
  bis
00F0 240                  "
00F1 241     Unechte Bildschirmzeile
00F2 242     Bildschirmzeilen-Marke
00F3 243     VEKTOR LOW  Zeiger ins aktuelle Farb-RAM
00F4 244     VEKTOR HIGH (ab $D800)
00F5 245     VEKTOR LOW   Tastatur-Dekodiertabelle
00F6 246     VEKTOR HIGH  Zeiger: $EB81 (60289)
00F7 247     VEKTOR LOW  RS 232 Eingabepuffer
00F8 248     VEKTOR HIGH
00F9 249     VEKTOR LOW  RS 232 Ausgabepuffer
00FA 250     VEKTOR HIGH
00FB 251     Zeropageraum zur eigenen Verwendung(unbenutzt)
00FC 252                                  "
00FD 253                                  "
00FE 254                                  "
00FF 255     Anfang des Puffers Umwandlung Fließkomma-ASCII

Ende Page 0
```

WICHTIGE ADRESSEN DER FOLGENDEN PAGES

0277 - 0280 631-640 TASTATURPUFFER

In diesem Bereich können bis zu 10 Zeichen zwischengespeichert ("gepuffert") werden.
Dies passiert beispielsweise immer dann, wenn Tasten gedrückt werden, jedoch momentan nicht weiterverarbeitet werden können (weil der Computer beispielsweise noch an anderer Stelle beschäftigt ist).
Der Tastaturpuffer läßt sich jedoch auch für eigene Zwecke verwenden: Die in den Tastaturpuffer gebrachten Zeichen können ausgegeben werden, sobald Adresse 198 die gewünschte Anzahl bekommt.
Die Besonderheit des Tastaturpuffers besteht darin, daß die Zeichen erst nach Beendigung des Programmes ausgegeben werden (also bereits im Direktmode).
Auf diese Weise lassen sich BASIC-Zeilen in ein bereits laufendes Programm einfügen (siehe DATA-Generator Kapitel 1).

0286 646 ZEICHENFARB-SPEICHER

In dieser Adresse ist die augenblickliche Zeichenfarbe abgespeichert. Sie kann von Ihnen geändert werden, indem Sie den Inhalt dieser Speicherstelle verändern.

```
10 A=INT(RND(1)*15)
20 POKE 646,A: REM ZUFÄLLIGE FARBE
30 PRINT"*";
40 GOTO 10
```

0 = schwarz	1 = weiß	2 = rot
4 = türkis	5 = violett	6 = grün
7 = blau	8 = gelb	9 = orange
10 = braun	11 = hellrot	12 = grau 1

13 = grau 2 14 = hellgrün 15 = hellblau
15 = grau 3

0288 648 HIGH-BYTE DES VIDEO-RAMs

Die Abfrage

```
PRINT PEEK(648)*256
```

ergibt die aktuelle Anfangsadresse des Bildschirmspeichers. Sie beträgt normalerweise 1024.
Verschoben werden kann der Bildschirmspeicher mit Hilfe des 4. - 7. Bits der Adresse 53272 und des 0. und 1. Bits der Adresse 56576 (siehe auch Kapitel "Grafik" !)

028A 650 REPEAT-FUNKTION FÜR ALLE TASTEN

Diese Adresse steuert die Prellfähigkeit der Tastatur:

```
POKE 650,0    Repeat nur für Steuertasten
POKE 650,64   Repeat off
POKE 650,128  Repeat für alle Tasten
```

028C 652 ZÄHLER FÜR REPEAT-VERZÖGERUNG

Sind alle Tasten durch Ändern der Adresse 650 mit einer Wiederholfunktion ausgestattet, so geht der Repeatvorgang folgendermaßen vor sich:
Es wird zunächst nur ein Zeichen ausgegeben. Nach einer gewissen Zeit erst werden dauernd weitere Zeichen ausgegeben.
Der Inhalt dieser Speicherstelle bestimmt nun die Dauer dieser kleinen Pause zwischen dem Ausgeben des ersten und der weiteren Zeichen.
Einen für Ihren Zweck passenden Wert müssen Sie durch etwas Herumprobieren selbst ermitteln!

028D;028E 653;654 FLAG FÜR SHIFT, C=- UND CTRL-TASTE

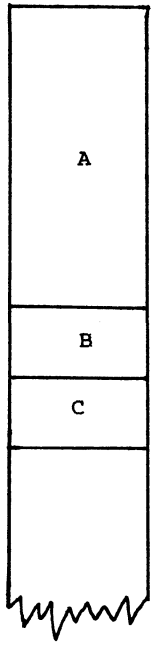
Bit 0, 1 und 2 dieser Adresse werden jeweils bei Betätigung einer dieser Tasten gesetzt.

0291 657 SPERRFLAG FÜR SHIFT/C=

Der Inhalt dieser Adresse entscheidet darüber, ob die Umschaltung von einem auf den anderen Zeichensatz durch Drücken der Shift- und C=-Taste gestattet wird oder nicht.

ALLGEMEINES ZUR VARIABLENSPEICHERUNG

Im Folgenden finden Sie eine Übersicht, aus der hervorgeht, wo welche Variablen im Speicher abgelegt werden:

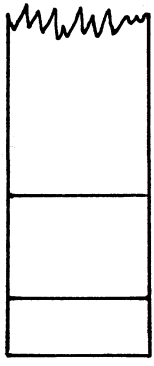


-Anfang BASIC-Speicher ($2B, $2C) $0800

-Bereich A: BASIC-Programm

-Variablen-Anfang ($2D, $2E)
-Bereich B: Variablenspeicherung
-Beginn der Arrays ($2F, $30)
-Bereich C: Speicherung der Array-Variablen
-Ende der Arrays ($31, $32)

-freier BASIC-Speicher



-freier BASIC-Speicher

-Stringbeginn (bewegt sich abwärts, $33, $34)
-gespeicherte Strings

-BASIC-Speicher-Ende ($37, $38)
-$9FFF

Schema der Variablenspeicherung

Im Bereich A befindet sich das BASIC-Programm. Direkt dahinter werden die Fließkomma- und Integervariablen sowie DEF FN gespeichert (Bereich B).
Im Bereich C sind die sogenannten Arrays gespeichert, also die Variablenfelder. Vom Ende des BASIC-Speichers her abwärts werden die Strings gespeichert.

Sämtliche Bereiche werden durch Zeiger bestimmt, die normalerweise automatisch durch den Interpreter gesetzt werden.
Diese Zeiger können jedoch auch von Ihnen verändert werden (insbesondere BASIC-Start und -Ende):

43/44	BASIC-Anfang
45/46	Variablen-Anfang
47/48	Array-Anfang
49/50	Array-Ende
51/52	String-Anfang
53/54	Hilfszeiger
55/56	BASIC-Ende

Beim Verlegen des Variablenanfanges ist folgendes zu beachten: Wird dieser Vektor höhergesetzt, so wird das Programm beim anschließenden AbSAVEn scheinbar länger. Die SAVE-Routine SAVEd blind alles zwischen der in Adresse 43/44 und der in Adresse 45/46 bestimmten Grenze!

Zur Speicherung:

Da die Arrays über einen eigenen Speicherplatz verfügen, gibt es insgesamt noch vier unterschiedliche

Variablenformen:

a) Fließkomma-Variablen (Variablennamen wie A, B, CD, etc.)
b) Stringvariablen (Variablennamen durch $ gekennzeichnet)
c) Integervariablen (Variablenname durch % gekennzeichnet)
und
d) Funktionen (FN)

Diese vier verschiedenen Variablentypen müssen von einander unterschieden werden können. Dazu sind zwei Bits (=4 Möglichkeiten) nötig.
Diese zwei Bits befinden sich im Namenszeichen.
Da es keine reversen Variablennamen gibt, steht schon einmal das 7.Bit zur Verfügung. Und da ein Variablenname grundsätzlich aus zwei Zeichen besteht (auch wenn nur eins angegeben wird), kommt man auf die zwei Bits.

Im Bereich B der Skizze wird für jede benutzte Variable gleich welcher Art ein Platz von sieben Bytes reserviert.
Zwei Bytes werden für die Namenszeichen gebraucht. Diese beiden Bytes werden somit automatisch reserviert, egal ob Variable A oder DR benutzt wird.
Nun zur oben erwähnten Unterscheidungsmöglichkeit mit den zwei 7. Bits der Namenszeichen:

7. Bit des...

Namenszeichen 1	Namenszeichen 2	Variablentyp
0	0	Fließkomma
0	1	String
1	0	Funktion
1	1	Integer

Restliche Bedeutung aller sieben Bytes:

BYTE	1	2	3	4	5	6	7	Variable
	NZ1	NZ2	gespeicherter Wert			binär		Fließkomma
	NZ1	NZ2	Länge	LO	HI	0	0	String
	NZ1	NZ2	HI	LO	0	0	0	Integer

NZ1 = erstes Namenszeichen
NZ2 = zweites Namenszeichen
LO und HI = Low- und High-Adresse des Variableninhaltes

Eine Ausnahme bildet FN. Diese Funktion benötigt 14 Bytes:

BYTE	1	2	3	4	5	6	7	8	9
	NZ1F	NZ2F	LO	HI	LOA	HIA	ASCD	NZ1V	NZ2V

BYTE	10	11	12	13	14
	Wert der Argumentvariable				

NZ1F/NZ2F = Erstes und zweites Namenszeichen Funktion
LO/HI = Low- und High-Adresse der Funktion
LOA/HIA = Low- und High-Adresse des Argumentes
NZ1V/NZ2V = Erstes und zweites Namenszeichen des Argum.

LISTE INTERESSANTER ZEIGER

Die folgenden Seiten enthalten alle wichtigen Zeiger des C-64.
Gerade mit Hilfe dieser Zeiger lassen sich äußerst wirkungsvolle Manipulationen hervorrufen.
Manche ROM-Routinen werden über Zeiger, die im RAM liegen, angesprungen. Durch Verändern dieser Zeiger können eigene Routinen angesprungen werden, ohne daß das ROM erst mühsam ins RAM kopiert werden muß.
Doch hier die Liste:

Low-, High-Byte		Sprungadd. Routine
3	4	Umwandlung Fließ-in Festkomma
5	6	Umwandlung Fest-in Fließkomma
23	24	Zeiger auf zuletzt verw. String
34	35	Zeiger zur freien Verwendung
36	37	"
43	44	BASIC-Start(1.Byte Bas.Start=0)
45	46	Variablenstart
47	48	Array-Start
49	50	Ende der Arrays
51	52	Beginn der Strings (bewegt sich abwärts)
53	54	Hilfszeiger für Strings
55	56	BASIC-Ende
61	62	Nächstes Statement für "CONT"
65	66	Nächstes DATA-Element
85	86	Sprungvektor für Funktionen
122	123	Programmzeiger der CHRGET-Rout.
178	179	Anfangsadresse Bandpuffer
187	188	Zeiger auf Programm-Name
209	210	Zeiger auf aktuelle BS-Zeile
243	244	Zeiger ins aktuelle Farb-RAM

245	246		Tastatur-Dekodiertabelle
247	248		RS 232 Eingabepuffer
249	250		RS 232 Ausgabepuffer
655	656		Tastatur-Decodierung
768	769	$E38B	BASIC-Warmstart
770	771	$A483	Zeilen-Eingabe
772	773	$A57C	Umwandlung in Interpreter-Code
774	775	$A71A	LIST (Umwandlung in Klartext)
776	777	$A7E4	BASIC-Befehlsadresse holen
778	779	$AE86	Ausdruck auswerten
785	786	$B248	USR-Zeiger
788	789	$EA31	Interrupt(IRQ)-Vektor
790	791	$FE66	BREAK-Vektor
792	793	$FE47	Nicht maskierbarer Interr. NMI
794	795	$F34A	OPEN
796	797	$F291	CLOSE
798	799	$F20E	CHKIN
800	801	$F250	CKOUT
802	803	$F333	CLRCH
804	805	$F157	INPUT
806	807	$F1CA	OUTPUT
808	809	$F6ED	STOP
810	811	$F13E	GET
812	813	$F32F	CLALL
814	815	$FE66	Warmstart
816	817	$F4A5	LOAD
818	819	$F5ED	SAVE

Die Abfrage der Zeiger geschieht mit:

```
PRINT PEEK(low)+256*PEEK(high)
```

Manche Zeiger lassen sich nur bei verhindertem Interrupt verändern:

```
POKE 56334,0     Interrupt off
POKE 56334,1     Interrupt on
```

ALLGEMEINES ZU DEN TABELLEN

Dieser Teil des Anhangs soll die Tabellensammlung, die im C-64 Handbuch steht, ergänzen. Wir sind nämlich der Meinung, daß dort wichtige Tabellen vergessen wurden. Damit man die zu den einzelnen Zahlen gehörenden Bedeutungen und Werte schneller findet, haben wir mehrere kleine Tabellen zu einer großen zusammen gefaßt. Nur die Übersicht für di Joystick-Abfrage stellt eine eigene Tabelle dar.
In der ersten Tabelle finden Sie eine Übersicht über die drei Zahlensysteme, die am meisten benutzt werden. Es sind Dezimal, Hexadezimal und Binär. Außerdem finden Sie hier folgende Bereiche: Maschinensprache-Befehle (MNEMONICs), BASIC-Befehle (Token) und die Tastaturdecodier-Tabelle.
Zu den ersten drei Spalten der Tabelle braucht eigentlich nichts gesagt zu werden. Als Computerbesitzer kommt man um diese Zahlen-Systeme nicht herum. In BASIC werden Zahlen fast nur dezimal dargestellt, hexadezimale Zahlen werden in der Maschinensprache verwendet. Binäre Zahlen muß man können, um die Arbeitsweise des Computers zu verstehen.
In der Spalte über die MNEMONICs wird das Format eines Maschinensprache-Befehls dargestellt. Jedes "z" bedeutet eine hexadezimale Zahl.
Wie Sie bestimmt wissen, wird jedes BASIC-Befehlswort im Speicher in eine Zahl umgewandelt. In der fünften Spalte der Tabelle sind diese Befehlsworte (Tokens) zu finden.
Die letzte Spalte stellt die Tastatur-Decodierungs-Tabelle (Tast. Deko.) dar. Dieser Wert wird z.B. in Adresse 197 oder 203 ausgegeben.

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
0	$00	%00000000	BRK		Inst Del
1	$01	%00000001	ORA (zz,X)		RETURN
2	$02	%00000010			Cursor right
3	$03	%00000011			F-7
4	$04	%00000100			F-1
5	$05	%00000101	ORA zz		F-3
6	$06	%00000110	ASL zz		F-5
7	$07	%00000111			Cursor down
8	$08	%00001000	PHP		3
9	$09	%00001001	ORA #zz		W
10	$0A	%00001010	ASL		A
11	$0B	%00001011			4
12	$0C	%00001100			Z
13	$0D	%00001101	ORA zzzz		S
14	$0E	%00001110	ASL zzzz		E
15	$0F	%00001111			SHIFT links
16	$10	%00010000	BPL zzzz		5
17	$11	%00010001	ORA (zz),y		R
18	$12	%00010010			D
19	$13	%00010011			6
20	$14	%00010100			C
21	$15	%00010101	ORA zz,X		F
22	$16	%00010110	ASL zz,X		T
23	$17	%00010111			X
24	$18	%00011000	CLC		7
25	$19	%00011001	ORA zzzz,y		Y
26	$1A	%00011010			G
27	$1B	%00011011			8
28	$1C	%00011100			B
29	$1D	%00011101	ORA zzzz,X		H

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
30	$1E	%00011110	ASL zzzz,X		U
31	$1F	%00011111			V
32	$20	%00100000	JSR zzzz		9
33	$21	%00100001	AND (zz,X)		I
34	$22	%00100010			J
35	$23	%00100011			O
36	$24	%00100100	BIT zz		M
37	$25	%00100101	AND zz		K
38	$26	%00100110	ROL zz		O
39	$27	%00100111			N
40	$28	%00101000	PLP		+
41	$29	%00101001	AND #zz		P
42	$2A	%00101010	ROL		L
43	$2B	%00101011			-
44	$2C	%00101100	BIT zzzz		.
45	$2D	%00101101	AND zzzz		:
46	$2E	%00101110	ROL zzzz		ü
47	$2F	%00101111			,
48	$30	%00110000	BMI zzzz		PFUND
49	$31	%00110001	AND (zz),y		*
50	$32	%00110010			;
51	$33	%00110011			CLR HOME
52	$34	%00110100			SHIFT right
53	$35	%00110101	AND zz,X		=
54	$36	%00110110	ROL zz,X		^
55	$37	%00110111			-
56	$38	%00111000	SEC		1
57	$39	%00111001	AND zzzz,y		PFEIL
58	$3A	%00111010			CTRL
59	$3B	%00111011			2

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko
60	$3C	%00111100			SPACE
61	$3D	%00111101	AND zzzz,X		commodore
62	$3E	%00111110	ROL zzzz,X		Q
63	$3F	%00111111			RUN STOP
64	$40	%01000000	RTI		KEINE TASTE
65	$41	%01000001	EOR (zz,x)		
66	$42	%01000010			
67	$43	%01000011			
68	$44	%01000100			
69	$45	%01000101	EOR zz		
70	$46	%01000110	LSR zz		
71	$47	%01000111			
72	$48	%01001000	PHA		
73	$49	%01001001	EOR #zz		
74	$4A	%01001010	LSR		
75	$4B	%01001011			
76	$4C	%01001100	JMP zzzz		
77	$4D	%01001101	EOR zzzz		
78	$4E	%01001110	LSR zzzz		
79	$4F	%01001111			
80	$50	%01010000	BVC zzzz		
81	$51	%01010001	EOR(zz),y		
82	$52	%01010010			
83	$53	%01010011			
84	$54	%01010100			
85	$55	%01010101	EOR zz,X		
86	$56	%01010110	LSR zz,X		
87	$57	%01010111			
88	$58	%01011000	CLI		
89	$59	%01011001	EOR zzzz,y		

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
90	$5A	%01011010			
91	$5B	%01011011			
92	$5C	%01011100			
93	$5D	%01011101	EOR zzzz,X		
94	$5E	%01011110	LSR zzzz,X		
95	$5F	%01011111			
96	$60	%01100000	RTS		
97	$61	%01100001	ADC (zz,X)		
98	$62	%01100010			
99	$63	%01100011			
100	$64	%01100100			
101	$65	%01100101	ADC zz		
102	$66	%01100110	ROR zz		
103	$67	%01100111			
104	$68	%01101000	PLA		
105	$69	%01101001	ADC #zz		
106	$6A	%01101010	ROR		
107	$6B	%01101011			
108	$6C	%01101100	JMP (zzzz)		
109	$6D	%01101101	ADC zzzz		
110	$6E	%01101110	ROR zzzz		
111	$6F	%01101111			
112	$70	%01110000	BVS zzzz		
113	$71	%01110001	ADC (zz),y		
114	$72	%01110010			
115	$73	%01110011			
116	$74	%01110100			
117	$75	%01110101	ADC zz,X		
118	$76	%01110110	ROR zz,X		
119	$77	%01110111			

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
120	$78	%01111000	SEI		
121	$79	%01111001	ADC zzzz,y		
122	$7A	%01111010			
123	$7B	%01111011			
124	$7C	%01111100			
125	$7D	%01111101	ADC zzzz,X		
126	$7E	%01111110	ROR zzzz,X		
127	$7F	%01111111			
128	$80	%10000000		END	
129	$81	%10000001	STA(zz,X)	FOR	
130	$82	%10000010		NEXT	
131	$83	%10000011		DATA	
132	$84	%10000100	STy zz	INPUT#	
133	$85	%10000101	STA zz	INPUT	
134	$86	%10000110	STX zz	DIM	
135	$87	%10000111		READ	
136	$88	%10001000	DEy	LET	
137	$89	%10001001		GOTO	
138	$8A	%10001010	TAX	RUN	
139	$8B	%10001011		IF	
140	$8C	%10001100	STy zzzz	RESTORE	
141	$8D	%10001101	STA zzzz	GOSUB	
142	$8E	%10001110	STX zzzz	RETURN	
143	$8F	%10001111		REM	
144	$90	%10010000	BCC zzzz	STOP	
145	$91	%10010001	STA (zz),y	ON	
146	$92	%10010010		WAIT	
147	$93	%10010011		LOAD	
148	$94	%10010100	STy zz,X	SAVE	
149	$95	%10010101	STA zz,X	VERIFy	

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
150	$96	%10010110	STX zz,y	DEF	
151	$97	%10010111		POKE	
152	$98	%10011000	TyA	PRINT#	
153	$99	%10011001	STA zzzz,y	PRINT	
154	$9A	%10011010	TXS	CONT	
155	$9B	%10011011		LIST	
156	$9C	%10011100		CLR	
157	$9D	%10011101	STA zzzz,X	CMD	
158	$9E	%10011110		SYS	
159	$9F	%10011111		OPEN	
160	$A0	%10100000	LDy #zz	CLOSE	
161	$A1	%10100001	LDA (zz,X)	GET	
162	$A2	%10100010	LDX #zz	NEW	
163	$A3	%10100011		TAB(	
164	$A4	%10100100	LDy zz	TO	
165	$A5	%10100101	LDA zz	FN	
166	$A6	%10100110	LDX zz	SPC(	
167	$A7	%10100111		THEN	
168	$A8	%10101000	TAy	NOT	
169	$A9	%10101001	LDA #zz	STEP	

170	$AA	%10101010	TAX	+
171	$AB	%10101011		-
172	$AC	%10101100	LDy zzzz	*
173	$AD	%10101101	LDA zzzz	-
174	$AE	%10101110	LDX zzzz	^
175	$AF	%10101111		AND
176	$B0	%10110000	BCS zzzz	OR
177	$B1	%10110001	LDA (zz),y	:
178	$B2	%10110010		=
179	$B3	%10110011		;

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
180	$B4	%10110100	LDY zz,X	SGN	
181	$B5	%10110101	LDA zz,X	INT	
182	$B6	%10110110	LDX zz,Y	ABS	
183	$B7	%10110111		USR	
184	$B8	%10111000	CLV	FRE	
185	$B9	%10111001	LDA zzzz,Y	POS	
186	$BA	%10111010	TSX	SQR	
187	$BB	%10111011		RND	
188	$BC	%10111100	LDY zzzz,X	LOG	
189	$BD	%10111101	LDA zzzz,X	EXP	
190	$BE	%10111110	LDX zzzz,Y	COS	
191	$BF	%10111111		SIN	
192	$C0	%11000000	CPY #zz	TAN	
193	$C1	%11000001	CMP (zz,X)	ATN	
194	$C2	%11000010		PEEK	
195	$C3	%11000011		LEN	
196	$C4	%11000100	CPY zz	STR$	
197	$C5	%11000101	CMP zz	VAL	
198	$C6	%11000110	DEC zz	ASC	
199	$C7	%11000111		CHR$	
200	$C8	%11001000	INY	LEFT$	
201	$C9	%11001001	CMP #zz	RIGHT$	
202	$CA	%11001010	DEX	MID$	
203	$CB	%11001011		GO	
204	$CC	%11001100	CPY zzzz		
205	$CD	%11001101	CMP zzzz		
206	$CE	%11001110	DEC zzzz		
207	$CF	%11001111			
208	$D0	%11010000	BNE zzzz		
209	$D1	%11010001	CMP (zz),Y		

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
210	ö$D2	%11010010			
211	ö$D3	%11010011			
212	ö$D4	%11010100			
213	ö$D5	%11010101	CMP zz,X		
214	ö$D6	%11010110	DEC zz,X		
215	$D7	%11010111			
216	$D8	%11011000	CLC		
217	$D9	%11011001	CMP zzzz,Y		
218	$DA	%11011010			
219	$DB	%11011011			
220	$DC	%11011100			
221	$DD	%11011101	CMP zzzz,X		
222	$DE	%11011110	DEC zzzz,X		
223	$DF	%11011111			
224	$E0	%11100000	CPX #zz		
225	$E1	%11100001	SBC (zz,X)		
226	$E2	%11100010			
227	$E3	%11100011			
228	$E4	%11100100	CPX zz		
229	$E5	%11100101	SBC zz		
230	$E6	%11100110	INC zz		
231	$E7	%11100111			
232	$E8	%11101000	INX		
233	$E9	%11101001	SBC #zz		
234	$EA	%11101010	NOP		
235	$EB	%11101011			
236	$EC	%11101100	CPX zzzz		
237	$ED	%11101101	SBC zzzz		
238	$EE	%11101110	DEC zzzz		
239	$EF	%11101111			

Dezi.	Hex.	Binär	Mnemonic	Token	Tast. Deko.
240	$F0	%11110000	BEQ zzzz		
241	$F1	%11110001	SBC (zz),Y		
242	$F2	%11110010			
243	$F3	%11110011			
244	$F4	%11110100			
245	$F5	%11110101	SBC zz,X		
246	$F6	%11110110	INC zz,X		
247	$F7	%11110111			
248	$F8	%11111000	SED		
249	$F9	%11111001	SBC zzzz,Y		
250	$FA	%11111010			
251	$FB	%11111011			
252	$FC	%11111100			
253	$FD	%11111101	SBC zzzz,X		
254	$FE	%11111110	INC zzzz,X		
255	$FF	%11111111		PI	

TABELLE DER GERÄTENUMMERN

Aus der folgenden Übersicht geht hervor, welche Gerätenummern welchem Gerät entsprechen:

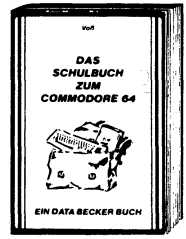
NUMMER	zugehöriges GERÄT
0	Tastatur
1	Tape (Kassette)
2	RS 232
3	Bildschirm
4	Drucker
5	(optionell) Drucker
6	-
7	-
8	Floppy Disk
9	(optionell) Floppy Disk
10	"
11	"
12	"
13	"
14	"
15	"

Hardwaremäßiger Betriebsstop

Sicherlich haben Sie es auch schon erlebt:
Gerade haben Sie eine sehr gute Runde in einem Action-Spiel erwischt - da klingelt das Telefon oder es klingelt an der Haustür, und Sie müssen Ihren Computer verlassen. Damit ist die Runde und die gute Laune verloren, da die wenigsten Spiele eine Taste haben, mit der Sie den Computer anhalten können.
Doch zum Glück gibt es eine Möglichkeit, den Computer hardwaremäßig zu stoppen. Verbinden Sie dazu die IRQ-Leitung am Expansionsport mit einer GND(Masse)-Leitung, und setzen Sie einen Schalter dazwischen.
Sobald Sie den Kontakt schließen, wird der Computer anhalten, und erst wieder weitermachen, wenn die Leitung wieder unterbrochen wird.
Doch den Schalter können Sie nicht nur zum Stoppen von Programmen verwenden. Denken Sie auch einmal an eine andere Sache, die oft viel zu schnell vor sich geht: Das Listen von Programmen.
Geht Ihnen wieder einmal CTRL beim Listen zu schnell, und ist es Ihnen zu mühselig dauernd LIST einzutippen, so setzen Sie doch einen Taster zwischen die Leitung, und halten Sie den Computer an.
Eines muß allerdings noch gesagt werden:
Durch Schließen des Kontaktes wird die IRQ-Routine öfter als sonst angesprungen, so daß der Cursor schneller blinkt, und die Uhr noch ungenauer als vorher ist. Beim Listen und Spielen dürfte das aber nicht von Bedeutung sein.

DATA BECKER'S NEUE BÜCHER UND PROGRAMME FÜR COMMODORE

Spickzettel ade.



Ein neues DATA BECKER BUCH, das den Einsatz des COMMODORE 64 in der Schule entscheidend mitprägen dürfte, wurde von Professor Voß geschrieben. Besonders für Schüler der Mittel- und Oberstufe geschrieben, enthält das Buch viele interessante Problemlösungs- und Lernprogramme, die besonders ausführlich und leicht verständlich beschrieben sind. Sie ermöglichen ein intensives und anregendes Lernen, unter anderem mit folgenden Themen: Satz des Pythagoras, quadratische Gleichungen, geometrische Reihen, Pendelbewegungen, mechanische Hebel, Molekülbildung, exponentielles Wachstum, Vokabeln lernen, unregelmäßige Verben, Zinseszinsrechnung. Ein kurzer Überblick über die Grundlagen der EDV, eine knappe Wiederholung der wichtigsten BASIC-Elemente und eine Einführung in die Grundzüge der Problemanalyse vervollständigen das Ganze. Mit diesem Buch machen die Hausaufgaben wieder Spaß!

DAS SCHULBUCH ZUM COMMODORE 64, 1984, über 300 Seiten, DM 49,–

Tempo!

MASCHINENSPRACHE FÜR FORTGESCHRITTENE ist bereits das zweite Buch von Lothar Englisch zum Thema Maschinenprogrammierung mit dem COMMODORE 64. Hier wird von der Problemanalyse bis zum Maschinensprachealgorithmus in die Grundlagen der professionellen Maschinenspracheprogrammierung eingeführt. In diesem Buch finden Sie unter anderem folgende Themen behandelt: Problemlösungen in Maschinensprache, Programmierung von Interruptroutinen, Interruptquellen beim COMMODORE 64, Interrupts durch CIA's und Videocontroller, Programmierung der Ein-Ausgabe-Bausteine, die CIA's des COMMODORE 64, Timer, Echtzeituhr, parallele und serielle Ein/Ausgabe, BASIC-Erweiterungen, Programmierung eigener BASIC-Befehle und -Funktionen, Möglichkeiten zur Einbindung ins Betriebssystem sowie viele weitere Tips & Tricks zur Maschinenprogrammierung. Dieses Buch sollte jeder haben, der wirklich intensiv mit der Maschinensprache des COMMODORE 64 arbeiten will.

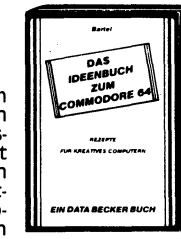
MASCHINENSPRACHE FÜR FORTGESCHRITTENE, 1984, ca. 200 Seiten, DM 39,–

Macht Druck.

DAS GROSSE DRUCKERBUCH für Drucker-Anwender mit COMMODORE-Computern ist endlich da! Es enthält eine riesige Sammlung von Tips & Tricks, Programmlistings und Hardwareinformationen. Rolf Brückmann und Klaus Gerits beschäftigen sich mit Sekundäradressen, Anschluß einer Schreibmaschine am Userport, Druckerschnittstellen (Centronics, V 24, IEC-Bus), hochauflösender Grafik, Text- und Grafikhardcopy, Grafik mit Standardzeichensatz, formatierter Datenausgabe, Plakatschrift, Textverarbeitung und vieles mehr. Zusätzlich wird das Betriebssystem des MPS801 zerlegt, mit Prozessorbeschreibung (8035), Blockschaltbild und einem ausführlich kommentierten ROM-Listing. Thomas Wiens schrieb den Teil über die Programmierung des Plotters VC-1520: Handhabung des Plotters, Programmierung von Sonderzeichen, Funktionendarstellung, Kuchen und Säulendiagramme, Entwurf dreidimensionaler Gegenstände. Natürlich wieder viele interessante Listings. Unentbehrlich für jeden, der einen COMMODORE 64 oder VC-20 und einen Drucker besitzt.

DAS GROSSE DRUCKERBUCH, 1984, über 300 Seiten, DM 49,–

Tausendsassa.



Fast alles, was man mit dem COMMODORE 64 machen kann, ist in diesem Buch ausführlich beschrieben. Es ist nicht nur spannend zu lesen wie ein Roman, sondern enthält neben nützlichen Programmlistings vor allem viele, viele Anwendungsmöglichkeiten des C64. Dabei wurde besonderer Wert darauf gelegt, daß das Buch auch für Laien leicht verständlich ist. Eine Auswahl aus der Themenvielfalt: Gedichte vom Computer, Einladung zur Party, Diplomarbeit – professionell gestaltet, individuelle Werbebriefe, Autokosten im Griff, Baukostenberechnung, Taschenrechner, Rezeptkartei, Lagerliste, persönliches Gesundheitsarchiv, Diätplan elektronisch, intelligentes Wörterbuch, kleine Notenschule, CAD für Handarbeit, Routenoptimierung, Schaufensterwerbung, Strategiespiele. Teilweise sind Programmlistings fertig zum Eintippen enthalten, soweit sich die „Rezepte" auf 1–2 Seiten realisieren ließen. Wenn Sie bisher nicht immer wußten, was Sie mit Ihrem 64er alles anfangen sollten, nach dem Lesen des IDEENBUCHES wissen Sie's bestimmt!

DAS IDEENBUCH ZUM COMMODORE 64, 1984, über 200 Seiten, DM 29,–

Prof. 64.

Ein faszinierendes Buch, um in die Welt der Wissenschaft einzusteigen, hat Rainer Severin geschrieben. Zunächst werden Variablentypen, Rechengenauigkeit und nützliche POKE-Adressen des COMMODORE 64 bezüglich den Anforderungen wissenschaftlicher Probleme analysiert. Verschiedene Sortieralgorithmen wie Bubble, Quick und Shell-Sort werden miteinander verglichen. Die Programmbeispiele aus der Mathematik nehmen dabei eine zentrale Stelle im Buch ein: Nullstellen nach Newton, numerische Ableitung mit dem Differenzenquotienten, lineare und nichtlineare Regression, Chi-Quadrat-Verteilung und Anpassungstest, Fourieranalyse und -synthese, Skalar-, Vektor- und Spatprodukt, ein Programmpaket zur Matrizenrechnung für Inversion, Eigenwerte und vieles weitere mehr. Programme aus der Chemie (Periodensystem), Physik, Biologie (Schadstoffe in Gewässern – Erfassung der Meßwerte), Astronomie (Planetenpositionen) und Technik (Berechnung komplexer Netzwerke, Platinenlayout am Bildschirm) und viele weitere Softwarelistings zeigen die riesigen Möglichkeiten auf, die der Computer in Wissenschaft und Technik hat.

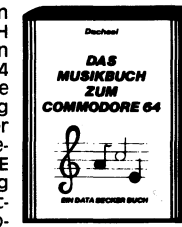
COMMODORE 64 FÜR TECHNIK UND WISSENSCHAFT, 1984, über 200 Seiten, DM 49,–

Grundkurs.

Das neue BASIC-Trainingsbuch zum C-64 ist eine ausführliche, didaktisch gut geschriebene Einführung in das CBM BASIC V2. Alle Befehle werden ausführlich erläutert. Dieses Buch geht aber über eine reine Befehlsbeschreibung hinaus, es wird eine fundierte Einführung in die Programmierung gegeben. Von der Problemanalyse bis zum fertigen Algorithmus lernt man das Entwerfen eines Programmes und den Entwurf von Datenflußplänen. ASCII-Code und verschiedene Zahlensysteme wie hexadezimal, binär und dezimal sind nach der Lektüre des Buches keine Fremdworte mehr. Die Programmierung von Schleifen, Sprüngen, bedingten Sprüngen lernt man leicht durch „learning by doing". So enthält das Trainingsbuch viele Aufgaben, Übungen und unzählige Beispiele. Den Schluß des Buches bildet eine Einführung ins professionelle Programmieren, in der es um mehrdimensionale Felder, Menuesteuerung und Unterprogrammtechnik geht. Endlich ein Buch, das Ihnen wirklich hilft, solide und sicher BASIC zu lernen.

BASIC TRAININGSBUCH ZUM COMMODORE 64, 1984, ca. 250 Seiten, DM 39,–

Sang und Klang!



Der COMMODORE 64 ist ein Musikgenie. DAS MUSIKBUCH hilft Ihnen, die riesigen Klangmöglichkeiten des C64 zu nutzen. Die Themenbreite reicht von einer Einführung in die Computermusik über die Erklärung der Hardwaregrundlagen des COMMODORE 64 und die Programmierung in BASIC bis hin zur fortgeschrittenen Musikprogrammierung in Maschinensprache. Einiges aus dem Inhalt: Soundregister des COMMODORE 64, Gate-Signal, Programmierung der "ADSR"-Werte, Synchronisation und Ring-Modulation, Counterprinzip, lineare und nichtlineare Musikprogrammierung, Frequenzmodulation, Interrupts in der Musikprogrammierung und vieles mehr. Zahlreiche Beispielprogramme, komplette Songs und nützliche Routinen ergänzen den Text. Geschrieben wurde das Buch von Thomas Dachsel, dem Autor der weltbekannten Musikprogramme Synthimat und Synthesound. Erschließen Sie sich die Welt des Sounds und der Computermusik mit dem Musikbuch zum C-64!

DAS MUSIKBUCH ZUM COMMODORE 64, über 200 Seiten, DM 39,–

Nützlich.

Das Trainingsbuch zu MULTIPLAN bietet eine gute Einführung in die Grundlagen der Tabellenkalkulation. Dabei wird großer Wert auf ein möglichst schnelles Einarbeiten in die wichtigsten Befehle gelegt, so daß man bald sicher mit MULTIPLAN arbeiten kann, ob nun auf dem COMMODORE 64 oder einem anderen Rechner. Am Ende wird man in der Lage sein, den umfangreichen Befehlssatz von MULTIPLAN auch kommerziell zu nutzen. Übungen am Ende jedes Kapitels sorgen dafür, daß man das Gelernte lange behält. Grundlage des Buches sind viele Seminare, die der Autor zu MULTIPLAN konzipiert und erfolgreich durchgeführt hat.

DAS TRAININGSBUCH ZU MULTIPLAN, 1984, ca. 250 Seiten, DM 49,–

Für Tüftler.

Ein hochinteressantes Buch für Hobbyelektroniker hat Rolf Brückmann vorgelegt. Er ist ein engagierter Techniker, für den der Computer Hobby und Beruf zur gleichen Zeit ist. Vor allem aber kennt er den C-64 in- und auswendig. So werden einführend die Schnittstellen des COMMODORE 64 detailliert beschrieben und kurz die Funktionsweise der CIAs 6526 erläutert. Hauptteil des Buches sind die Beschreibungen der vielfältigen Einsatzmöglichkeiten des COMMODORE 64. Die vielen Schaltungen, von Rolf Brückmann alle selbst

entwickelt, sind jeweils umfangreich dokumentiert und leichtverständlich erklärt. Die Reihe der hier ausführlich behandelten Anwendungen mit dem COMMODORE 64 ist äußerst umfangreich: Motorsteuerung, Stoppuhr mit Lichtschranke, Lichtorgel, A/D-Wandler, Spannungsmessung, Temperaturmessung und vieles mehr. Dazu kommen noch eine Reihe kompletter Schaltungen zum Selberbauen, wie ein EPROM Programmiergerät für den C-64, eine EPROM-Karte, ein Frequenzzähler und Sprachein/ausgabe (!). Zusätzlich sind jeweils Schaltplan, Softwarelisting und zu einigen Schaltungen sogar zusätzlich Platinenlayouts vorhanden.

DER COMMODORE 64 UND DER REST DER WELT, 1984, ca. 220 Seiten, DM 49,–

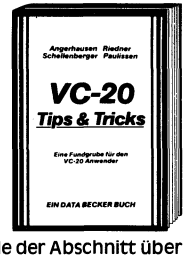
Computerkünstler.

Das Grafikbuch zum COMMODORE 64 Buch aus der Bestseller-Serie von DATA BECKER stammt aus der Feder von Axel Plenge. Es geht weit über die reine Hardware-Beschreibung der Grafikeigenschaften des C-64 hinaus. Der Inhalt reicht von den Grundlagen der Grafikprogrammierung bis zum Computer Aided Design. Es ist ein Buch für alle, die mit ihrem C-64 kreativ tätig sein wollen. Themen sind z. B.: Zeichensatzprogrammierung, bewegte Sprites, High-Resolution, Multicolor-Grafik, Lightpenanwendungen, Betriebsarten des VIC, Verschieben der Bildschirmspeicher, IRQ-Handhabung, 3-Dimensionale Grafik, Projektionen, Kurven-, Balken- und Kuchendiagramme, Laufschriften, Animation, bewegte Bilder. Viele Programmlistings und Beispiele sind selbstverständlich. Das COMMODORE-BASIC V2 unterstützt die herausragenden Grafikeigenschaften des C-64 bekanntlich kaum. Hier helfen die vielen Beispielprogramme in diesem Buch weiter, die die faszinierende Welt der Computergrafik jedermann zugänglich machen. Kompetent ist der Autor dazu wie kaum ein anderer, schließlich hat er das äußerst leistungsfähige Programm SUPERGRAFIK geschrieben.

DAS GRAFIKBUCH ZUM COMMODORE 64, 1984, 295 Seiten, DM 39,–

Vielfalt.

Auf dem neuesten Stand ist VC-20 TIPS & TRICKS von Dirk Paulissen gebracht worden, der über hundert Seiten hinzufügte. Bisher schon enthalten waren Informationen über Speicheraufbau des VC-20 und die Erweiterungsmöglichkeiten, ein Grafikkapitel über programmierbare Zeichen, Laufschrift und die Supererweiterung. Stark erweitert wurde der Abschnitt über POKEs und andere nützliche Routinen. Ob es um die Programmierung der Funktionstasten, Programme die sich selber starten, „Maus"-Simulation mit dem Joystick oder die Änderung von Speicherbereichen geht, man ist immer wieder über die Fülle der Möglichkeiten erstaunt. Der Clou dieses Buches sind aber die vielen Programmlistings. Die BASIC-Erweiterungen allein stellen schon ein erstklassiges Toolkit dar: APPEND (Anhängen von Programmen, AUTO (automatische Zeilennummerierung), BASIC-Befehle auf Tastendruck, PRINT POSITION, UNNEW, Strings größer als 88 Zeichen einlesen und vieles mehr. Die Bandbreite reicht von Spielen wie Goldgräber oder Starshooter bis zu nützlichen Programmen wie Cassetteninhaltsverzeichnis und -katalog mit automatischem Suchen nach Dateien und einem Terminkalender. Für den VC-20 Anwender ist dieser 324 Seiten-Wälzer eine wahre Fundgrube, in der es immer etwas neues zu entdecken gibt.



VC-20 TIPS & TRICKS, 3. erweiterte und überarbeitete Auflage, 1984, 324 Seiten, DM 49,–

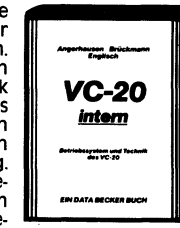
Interessant.

Einen guten Einstieg in PASCAL bietet dieses Trainingsbuch. Es gibt eine leichtverständliche Einführung, sowohl in UCSD-PASCAL wie auch in PASCAL64, wobei allerdings EDV-und BASIC-Grundkenntnisse vorausgesetzt werden. Der Autor, Ottmar Korbmacher, ist Student der Mathematik. Ihm gelingt es, in einem sprachlich aufgelockerten Stil mit vielen interessanten Beispielprogrammen, dem Leser Programmstrukturen, Ein/Ausgabe, Arithmetik und Funktionen, Prozeduren und Rekursionen, Sets, Files und Records näherzubringen. Die Übungsaufgaben am Ende jeden Kapitels helfen dabei, das Gelernte zu vertiefen. Ein Anhang mit allen PASCAL-Schlüsselworten, der ansich schon ein umfangreiches Lexikon darstellt, macht das Buch für jeden PASCAL-Anwender interessant.

DAS TRAININGSBUCH ZU PASCAL, 1984, ca. 250 Seiten, DM 39,–

Bewährt.

Die bereits dritte Auflage von VC-20 INTERN ist wieder erheblich erweitert worden. Das Buch beschäftigt sich ausführlich mit der Technik und dem Betriebssystem des VC-20. Dazu gehört natürlich zuerst einmal ein ausführlich dokumentiertes ROM-Listing. Dazu gehört auch die Belegung der Zeropage, dem wichtigsten Speicherbereich für den 6502-Prozessor, eine übersichtliche Auflistung der Adressen aller Betriebssystemroutinen, ihrer Bedeutung und ihrer Übergabeparameter. Dies ermöglicht dem Programmierer endlich, den VC-20 von Maschinensprache aus sinnvoll einzusetzen. Denn warum Routinen, die bereits vorhanden sind, noch einmal schreiben? Weiterer Inhalt: Einführung in die Maschinensprache – Maschinensprachemonitor, Assembler, Disassembler – Verbindung von Maschinensprache- und BASIC-Programmen – Beschreibung der wichtigen IC's des VC-20 – Blockschaltbild – drei Original COMMODORE-Schaltpläne. Das Buch braucht jeder der sich intensiv mit der Maschinenspracheprogrammierung des VC-20 auseinandersetzen möchte.



VC-20 INTERN, 3. Auflage, 1984, ca. 230 Seiten, DM 49,–

Starthilfe!

Das sollte Ihr erstes Buch zum COMMODORE 64 sein: 64 FÜR EINSTEIGER ist eine sehr leicht verständliche Einführung in Handhabung, Einsatz, Ausbaumöglichkeiten und Programmierung des COMMODORE 64, die keinerlei Vorkenntnisse voraussetzt. Sie reicht vom Anschluß des Geräts über die Erklärung der einzelnen Tasten und Funktionen sowie die Peripheriegeräte und ihre Bedienung bis zum ersten Befehl. Schritt für Schritt führt das Buch Sie in die Programmiersprache BASIC ein, wobei Sie nach und nach eine komplette Adressenverwaltung erstellen, die Sie anschließend nutzen können. Zahlreiche Abbildungen und Bildschirmfotos ergänzen den Text. Viele Anwendungsbeispiele geben nützliche Anregungen zum sinnvollen Einsatz des COMMODORE 64. Das Buch ist sowohl als Einführung als auch als Orientierung vor dem 64er Kauf gut geeignet.

64 FÜR EINSTEIGER, 1984, ca. 200 Seiten, DM 29,–

Von A bis Z.

So etwas haben Sie gesucht: Umfassendes Nachschlagewerk zum COMMODORE 64 und seiner Programmierung. Allgemeines Computerlexikon mit Fachwissen von A–Z und Fachwörterbuch mit Übersetzungen wichtiger englischer Fachbegriffe – das DATA BECKER LEXIKON ZUM COMMODORE 64 stellt praktisch drei Bücher in einem dar. Es enthält eine unglaubliche Vielfalt an Informationen und dient so zugleich als kompetentes Nachschlagewerk und als unentbehrliches Arbeitsmittel. Viele Abbildungen und Beispiele ergänzen den Text. Ein Muß für jeden COMMODORE 64 Anwender!

DAS DATA BECKER LEXIKON ZUM COMMODORE 64, 1984, 354 Seiten, DM 49,–

Fundgrube.

64 Tips & Tricks ist eine hochinteressante Sammlung von Anregungen zur fortgeschrittenen Programmierung des COMMODORE 64, POKE's und andere nützliche Routinen, interessanten Programmen sowie interessanten Programmiertips & -tricks. Aus dem Inhalt: 3D-Graphik in BASIC – Farbige Balkengraphik – Definition eines eigenen Zeichensatzes – Tastaturbelegung und ihre Änderung – Dateneingabe mit Komfort – Simulation der Maus mit einem Joystick – BASIC für Fortgeschrittene – C-64 spricht deutsch – CP/M auf dem COMMODORE 64 – Druckeranschluß über den USER-Port – Datenübertragung von und zu anderen Rechnern – Expansion-Port – Synthesizer in Stereo – Retten einer nicht ordnungsgemäß geschlossenen Datei – Erzeugen einer BASIC-Zeile in BASIC – Kassettenpuffer als Datenspeicher – Sortieren von Stringfelder – Multitasking auf dem COMMODORE 64 – POKE's und die Zeropage – GOTO, GOSUB und RESTORE mit berechneten Zeilennummern, INSTR und STRING-Funktion – Repeat-Funktion für alle Tasten – und vieles andere mehr. Alle Maschinenprogramme mit BASIC-Ladeprogrammen. 64 Tips & Tricks ist eine echte Fundgrube für jeden COMMODORE 64 Anwender. Schon über 65000mal verkauft!

64 TIPS & TRICKS, 1984, über 300 Seiten, DM 49,–

Know-how!

350 Seiten dick ist die 4. erweiterte und überarbeitete Auflage von 64 INTERN geworden. Das bereits über 65000mal verkaufte Standardwerk bietet jetzt noch mehr Informationen. Hinzugekommen ist ein Kapitel über den IEC-Bus und viele, viele Ergänzungen, die sich im Laufe der Zeit angesammelt haben. Ebenfalls überarbeitet und noch ausführlicher ist jetzt die Dokumentation des ROM-Listings. Weitere Themen: genaue Beschreibung des Sound- und Video-Controllers mit vielen Hinweisen zur Programmierung von Sound und Grafik, der Ein/Ausgabesteuerung (CIAs), BASIC-Erweiterungen (RENEW, HARDCOPY, PRINTUSING), Hinweise zur Maschinenprogrammierung wie Nutzung der E/A-Routinen des Betriebssystems, Programmierung der Schnittstelle RS 232, ein Vergleich VC20 – C-64 – CBM zur Umsetzung von Programmen. Dies und viele weitere Informationen machen das umfangreiche Werk zu einem unentbehrlichen Arbeitsmittel für jeden, der sich ernsthaft mit Betriebssystem und Technik des C-64 auseinandersetzen will. Zum professionellen Gehalt des Buches tragen auch zwei Original-COMMODORE-Schaltpläne zum Ausklappen und zahlreiche ausführlich beschriebene und dokumentierte Fotos, Schaltbilder und Blockdiagramme bei.

64 INTERN, 4. überarbeitete und erweiterte Auflage, 1984, ca. 350 Seiten, DM 69,–

Erfolgreich.

64 für Profis zeigt, wie man erfolgreich Anwendungsprobleme in BASIC löst und verrät die Erfolgsgeheimnisse der Programmierprofis. Vom Programmentwurf über Menüsteuerung, Maskenaufbau, Parametrisierung, Datenzugriff und Druckausgabe bis hin zur guten Dokumentation wird anschaulich mit vielen Beispielen dargestellt wie Profi-Programmierung vor sich geht. Besonders stolz sind wir auf die völlig neuartige Datenzugriffsmethode QUISAM, die in diesem Buch zum ersten Mal vorgestellt wird. QUISAM erlaubt eine beliebige Datensatzlänge, die dynamisch mit der Eingabe der Daten wächst. Eine lauffertige Literaturstellenverwaltung veranschaulicht die Arbeitsweise von QUISAM. Neben diesem Programm finden Sie noch weitere Programme zur Lager- und Adressenverwaltung, Textverarbeitung und einen Reportgenerator. Alle diese Programme sind mit Variablenliste versehen und ausführlich beschrieben. Damit sind diese für Ihre Erweiterungen offen und können von Ihnen an Ihre persönlichen Bedürfnisse angepaßt werden. Steigen Sie in die Welt der Programmierprofis ein.

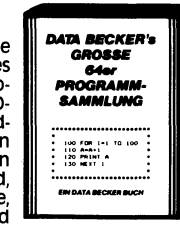
64 FÜR PROFIS, 2. Auflage, 1984, ca. 300 Seiten, DM 49,–

Rundum gut!

Endlich ein Buch, das Ihnen ausführlich und verständlich die Arbeit mit der Floppy VC-1541 erklärt. Das große Floppybuch ist für Anfänger, Fortgeschrittene und Profis gleichermaßen. interessant. Sein Inhalt reicht von der Programmspeicherung bis zum DOS-Zugriff, von der sequentiellen Datenspeicherung bis zum Direktzugriff, von der technischen Beschreibung bis zum ausführlich dokumentierten DOS-Listing, von den Systembefehlen bis zur detaillierten Beschreibung der Programme auf der Test-Demo-Diskette. Exakt beschriebene Beispiel- und Hilfsprogramme ergänzen dieses neue Superbuch. Aus dem Inhalt: Speichern von Programmen – Floppy-Systembefehle – Sequentielle Datenspeicherung – relative Datenspeicherung – Fehlermeldungen und ihre Ursachen – Direktzugriff – DOS-Listing der VC-1541 – BASIC-Erweiterungen und Programme – Overlaytechnik – Diskmonitor – IEC-Bus und serieller Bus – Vergleich mit den großen CBM-Floppies. Ein Muß für jeden Floppy-Anwender! Bereits über 45.000mal verkauft.

DAS GROSSE FLOPPY-BUCH, 2. überarbeitete Auflage, 1984, ca. 320 Seiten, DM 49,–

Füttern erwünscht!



Diese beliebte umfangreiche Programmsammlung hat es in sich. Über 50 Spitzenprogramme für den COMMODORE 64 aus den unterschiedlichsten Bereichen, von attraktiven Superspielen (Senso, Pengo, Master Mind, Seeschlacht, Poisson Square, Memory) über Grafik- und Soundprogramme (Fourier 64, Akustograph, Funktionsplotter) und mathematische Programme (Kurvendiskussion, Dreieck) sowie Utilities (SORT, RENUMBER, DISK INIT, MENUE) bis hin zu kompletten Anwendungsprogrammen wie „Videothek", „File Manager" und einer komfortablen Haushaltsbuchführung, in der fast professionell gebucht wird. Der Hit zu jedem Programm sind aktuelle Programmiertips und Tricks der einzelnen Autoren zum Selbermachen. Also nicht nur abtippen, sondern auch dabei lernen und wichtige Anregungen für die eigene Programmierung sammeln.

DATA BECKER's GROSSE 64er PROGRAMMSAMMLUNG, 1984, 250 Seiten, DM 49,–

Bestseller aus bester Hand

BASIC-PLUS.

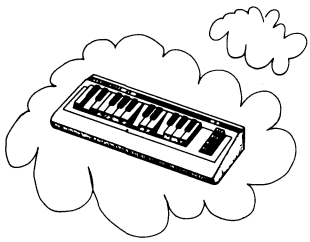
SIMON's BASIC ist ein Hit – wenn man es richtig nutzen kann. Auf über 300 Seiten erklärt Ihnen das DATA BECKER Trainingsbuch detailliert den Umgang mit den über 100 Befehlen des SIMON's BASIC. Alle Befehle werden ausführlich dargestellt, auch die, die nicht im Handbuch stehen! Natürlich zeigen wir auch die Macken des SIMON's BASIC und geben wichtige Hinweise wie man diese umgeht. Natürlich enthält das Buch viele Beispielprogramme und viele interessante Programmiertricks. Weiterer Inhalt: Einführung in das CBM-BASIC 2.0 – Programmierhilfen – Fehlerbehandlung – Programmschutz – Programmstruktur – Variablen – Zahlenbehandlung – Eingabekontrolle – Ein/Ausgabe Peripheriebefehle – Graphik – Zeichensatzerstellung – Sprites – Musik – SIMON's BASIC und die Verträglichkeit mit anderen Erweiterungen und Programmen. Dazu ein umfangreicher Anhang. Nach jedem Kapitel finden Sie Testaufgaben zum optimalen Selbststudium und zur Lernerfolgskontrolle.

DAS TRAININGSBUCH ZUM SIMON's BASIC, 2. überarbeitete Auflage, 1984, ca. 380 Seiten, DM 49,–

Schrittmacher.

Eine leicht verständliche Einführung in die Maschinenspracheprogrammierung für alle, denen das C-64 BASIC nicht mehr ausreicht. Sie lernen Aufbau und Arbeitsweise des 6510-Mikroprozessors kennen und anwenden. Dabei werden die Analogien zu BASIC Ihnen beim Verständnis helfen. Ein weiteres Kapitel beschäftigt sich mit der Eingabe von Maschinenprogrammen. Dort erfahren Sie auch alles über Monitor-Programme sowie über Assembler. Zum einfachen und komfortablen Erstellen Ihrer eigenen Maschinensprache enthält das Buch einen kompletten ASSEMBLER, damit Sie gleich von Anfang an komfortabel und effektiv programmieren können. Weiterhin finden Sie dort einen DISASSEMBLER, mit dem Sie sich Ihre Maschinenprogramme oder die Routinen des BASIC-Interpreters und des BASIC-Betriebssystems ansehen können. Ein besonderer Clou ist ein in BASIC geschriebener Einzelschrittsimulator, mit dem Sie Ihre Programme schrittweise ausführen können. Dabei werden Sie nach jedem Schritt über Registerinhalte und Flags informiert und können den logischen Ablauf Ihres Programmes verfolgen. Eine unschätzbare Hilfe, besonders für den Anfänger. Als Beispielprogramm finden Sie ausführlich beschriebene Routinen zur Grafikprogrammierung und für BASIC-Erweiterungen. Natürlich sind alle Beispiele und Programme auf den C-64 zugeschnitten.

DAS MASCHINENSPRACHEBUCH ZUM COMMODORE 64, ca. 200 Seiten, DM 39,–



SYNTHIMAT

SYNTHIMAT verwandelt Ihren COMMODORE 64 in einen professionellen, polyphonen, dreistimmigen Synthesizer, der in seinen unglaublich vielen Möglichkeiten großen Systemen kaum nachsteht.

SYNTHIMAT in Stichworten:
drei Oszillatoren (VCOs) mit 7 Fußlagen und 8 Wellenformen – drei Hüllkurvengeneratoren (ADSRs) – ein Filter (VCF) mit 8 Betriebsarten und Resonanzregulierung – VCF mit Eingang für externe Signalquelle – ein Verstärker (VCA) – Ringmodulation mit allen drei VCOs – 8 softwaremäßig realisierte Oszillatoren (LFOs) – kräftiger Klang durch polyphones Spielen – zwei Manuale (Solo und Begleitung) – speichern von bis zu 256 Klangregistern – schneller Registerwechsel – speichern von 9 Registerdateien auf Diskette – „Bandaufnahme" auf Diskette durch direktes Spielen – keine lästige Noteneingabe – speichern von bis zu 9 „Bandaufnahmen" je Diskette – integrierte 24 Stunden-Echtzeituhr – einstellbares PITCH-BENDING – farblich gekennzeichnete, übersichtlich angeordnete Module – umfangreiches Handbuch – läuft mit einem Diskettenlaufwerk – Diskettenprogramm.

DM 99,–

STRUKTO 64

STRUKTO 64 ist eine fantastische neue Programmiersprache für strukturiertes Programmieren mit dem C-64 und für alle Programmierer geeignet, die den C-64 als Allround-Computer einsetzen und auf einfache Weise anspruchsvolle Programme erstellen wollen.

STRUKTO 64 in Stichworten:
Interpretersprache, die die Vorzüge von BASIC und PASCAL vereint – strukturiertes Programmieren – übersichtliche Programme – leichte Erlernbarkeit – einfache Bedienung – eingebautes Toolkit erleichtert das Eingeben und Verbessern von Programmen – leichteres Arbeiten mit der Floppy – Sprite-Editor ermöglicht das Einlesen der Sprite-Formen direkt vom Bildschirm – Graphikbedienung wird mit gut durchdachten Befehlen unterstützt – Abspielen von Musik ist unabhängig vom Programmablauf möglich – ca. 80 neue Befehle – lieferbar als Diskettenprogramm – ausführliches deutsches Handbuch.

DM 99,–

Für viele ein Traum, für die meisten bisher zu teuer: die Rede ist von einer echten Datenbank für den 64er. SUPERBASE 64 füllt eine Lücke. Nicht allein die Kapazität, die verwaltet werden kann, bewegt sich in professionellen Regionen, die ausgeprägten Fähigkeiten des SUPERBASE 64 im Rechnen und Kalkulieren lassen dieses Paket beinahe als Rund-Um-Software erscheinen.

SUPERBASE 64 in Stichworten:
maximale Datensatzlänge 1108 Zeichen, verteilt auf bis zu 4 Bildschirmseiten – bis zu 127 Felder pro Datensatz, wobei Textfelder bis zu 255 Zeichen lang sein können – insgesamt 15 Einzeldateien können zu einer SUPERBASE-Datenbank verknüpft werden – Speicherkapazität nur durch Diskette begrenzt – umfangreiche Auswertungsmöglichkeiten und komfortabler Report-Generator – Kalkulationsmöglichkeiten und Rechnen – Import- (Einlesen von externen Daten) und Export- (Ausgabe von SUPERBASE Dataien als sequentielle Datei) Funktionen ermöglichen Datenaustausch mit anderen Programmen – durch leistungsfähige, eigene Datenbanksprache auch als kompletter Anwendungsgenerator verwendbar.

DM 398,–

MASTER 64

MASTER 64 ist ein professionelles Programmentwicklungssystem für den C-64, das es Ihnen ermöglicht, die Programmentwicklungszeit auf einen Bruchteil der sonst üblichen Zeit zu reduzieren. MASTER 64 bietet einen Programmkomfort, den Sie nutzen sollten.

MASTER 64 in Stichworten:
70 zusätzliche Befehle – Bildschirmmaskengenerator – definieren von Bildschirmzonen – Eingabe aus Zonen – formatierte Ausgabe – Abspeicherung von Bildschirminhalten – Arbeiten mit mehreren Bildschirmmasken – ISAM Dateiverwaltung, in der Datensätze über einen Zugriffschlüssel angesprochen werden können – Datensätze bis zu 254 Zeichen – Schlüssellänge bis zu 30 Zeichen – Dateigröße nur von Diskettenkapazität abhängig – Zugriff über Schlüssel und Auswahlmasken – Bildschirm- und Druckmaskengenerator – Erstellung beliebiger Formulare und Ausgabemasken – BASIC-Erweiterungen – Toolkitfunktionen – Mehrfachgenaue Arithmetik (Rechnen mit 22 Stellen Genauigkeit).

DM 198,–

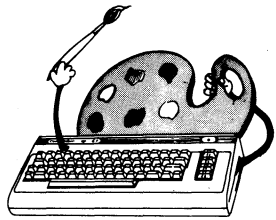
TEXTOMAT

Das Bearbeiten von Texten gehört zum wichtigsten Betätigungsfeld von Homecomputer-Anwendern. So ist es auch nicht verwunderlich, daß eine Unzahl verschiedener Textprogramme für den 64er angeboten wird. TEXTOMAT zeichnet sich dadurch aus, daß er auch vom Einsteiger sofort benutzt werden kann. Über eine Menuezeile können alle Funktionen angewählt werden. Selbstverständlich beherrscht TEXTOMAT deutsche Umlaute und Sonderzeichen.

TEXTOMAT in Stichworten:

Diskettenprogramm – durchgehend menuegesteuert – deutscher Zeichensatz auch auf COMMODORE-Druckern Rechenfunktionen für alle Grundrechenarten – 24.000 Zeichen pro Text im Speicher – beliebig lange Texte durch Verknüpfung – horizontales Scrolling für 80 Zeichen pro Zeile – läuft mit 1 oder 2 Floppies – frei programmierbare Steuerzeichen – Formularsteuerung für Randeinstellung u.s.w. – komplette Bausteinverarbeitung – Blockoperationen, Suchen und Ersetzen – Serienbriefschreibung mit DATAMAT – formatierte Ausgabe auf Bildschirm – an fast jeden Drucker anpaßbar – ausführliches deutsches Handbuch mit Übungslektionen.

DM 99,–



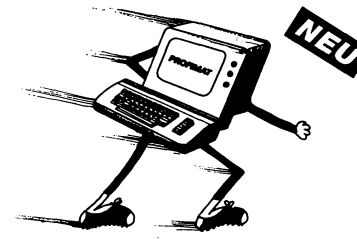
PAINT PIC

Malen (!) mit dem Computer, welch eine faszinierende Idee. Mit dem Malprogramm PAINT PIC für den COMMODORE 64 wird diese Idee Realität. Mit PAINT PIC ist es auch für den Einsteiger leicht, fantastische Computerbilder zu erstellen. Man kann die Bilder auf Diskette abspeichern und wieder laden und selbstverständlich steht auch weiterhin der COMMODORE-Zeichensatz zur Verfügung. Wichtig: PAINT PIC benötigt keine zusätzliche Hardware.

PAINT PIC in Stichworten:

Programmsteuerung: Tastatur – Steuerung des Stifts: Cursortasten und eckige Klammer (diag.) (Joystick kann benutzt werden) – Routinen: Linien, Rechtecke, Dreiecke, Parallelogramme, Kreise, Kreisbögen, Ellipsen, Bestimmung von Mittelpunkt, und perspektivischer Linie, Kopieren und Drehen von Teilbildern, Verdoppeln, halbieren und spiegeln von Teilbildern – Modi: Malstiftmodus (schmale Linie) Pinselmodus (8 verschiedene Breiten) (Art der Linie selbst definierbar) — Textmodus (kompl. Zeichensatz COMMODORE) (Hoch-Tiefschrift) – Speichern: Teilbilder (Blöcke) oder ganze Bilder — Menue: 1 Hauptmenue mit 8 Untermenues – mit ausführlichem deutschen Handbuch – Diskettenprogramm – Bilder kann man auf Diskette abspeichern.

DM 99,–



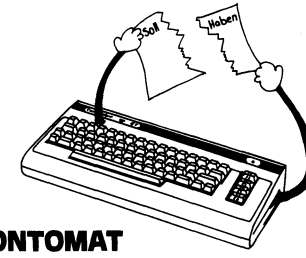
PROFIMAT

Wer sich tiefer in die Innereien des Computers begeben will, kommt ohne besonderes Werkzeug nicht aus. Einerseits muß der volle Einblick in alle Speicherbereiche möglich sein, andererseits soll der Umgang mit Maschinenprogrammen so komfortabel wie möglich gestaltet sein. PROFIMAT hat Lösungen für beide Probleme: Der Maschinensprache-Monitor PROFI-MON bietet alle Hilfsmittel zum Umgang mit Maschinenprogrammen; PROFI-ASS ist ein Macro-Assembler, der das Schreiben von Maschinenprogrammen fast so einfach macht wie das Programmieren in BASIC.

PROFIMAT in Stichworten:

Registerinhalte und Flags anzeigen – Speicherinhalte anzeigen – Maschinenprogramme laden, ausführen und speichern – Speicherbereiche durchsuchen, vergleichen, füllen und verschieben – echter Einzelschrittmodus – Setzen von Unterbrechungspunkten – schneller Trace-Modus – Rückkehr zu BASIC – formatfreie Eingabe – Verkettung beliebig vieler Quellprogramme – erzeugter Objektcode kann in Speicher oder auf Diskette gehen – formatiertes Assemblerlisting – ladbare Symboltabellen – redefinierbare Symbole – Operatoren – Unterstützung der Fließkommaarithmetik – bedingte Assemblierung – Assemblerschleifen – MACROS mit beliebigen Parametern.

DM 99,–

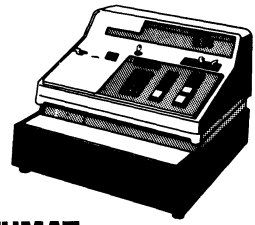


KONTOMAT

KONTOMAT ist ein menuegesteuertes Einnahme-Überschußprogramm nach §4(3) EStG mit Kassenbuch, Bankkontenüberwachung, automatischer Steuerbuchung, AFA Tabellenerstellung, Kontenblättern, Ermittlung der UST.-Voranmeldungswerte und Monats- und Jahresabrechnung. Der neue KONTOMAT ist voll parameterisiert und läßt sich damit an Ihre Bedürfnisse anpassen. Für alle Gewerbetreibenden, die nicht laut HGB zur Buchführung verpflichtet sind. KONTOMAT ist für den gewerblichen Einsatz, aber auch als Lernprogramm oder zur Haushaltsbuchführung geeignet.

KONTOMAT in Stichworten:
Diskettenprogramm – maximal 120 Konten – Beträge mit bis zu 6 Vor- und 2 Nachkommastellen – 4 Mehrwert- und Vorsteuersätze – intervallmäßige Belegeingabe – 4 Buchungsarten (SOLL, HABEN, SOLL/HABEN und HABEN/SOLL) – Anzeige der Soll- und Habensumme bei mehrfachen Buchungssätzen – komfortable Belegeingabe mit Datum, Buchungstext, Stuerkennzeichen und Betrag – Druck des Journals während der Belegeingabe – Druck von umfangreichen Kontenblättern – Druck einer Summen- und Saldenliste mit Monats- und Jahresumsatzsummen – betriebswirtschaftliche Auswertung mit Druckausgabe – Ermittlung und Druckausgabe der Umsatzsteuerzahllast – Speicherung der Anlagegüter und automatische Abschreibung am Jahresende – übersichtliche AfA-Liste – arbeitet mit 1 oder 2 Laufwerken – umfangreiches deutsches Handbuch.

DM 148,–

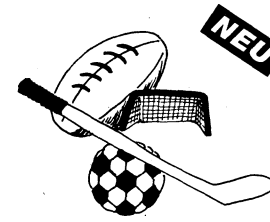


FAKTUMAT

Mit FAKTUMAT ist das Schreiben von Rechnungen kein Alptraum mehr. Eine Sofortfakturierung mit integrierter Lagerbuchführung. Individuelle Anpassung von Steuersätzen, Maßeinheiten und Firmendaten. Kunden- und Artikelstamm voll pflegbar. Schneller Zugriff auf Kunden- und Artikeldaten, über freidefinierbaren, 6-stelligen Schlüssel. Automatische Fortschreibung von Artikel- und Kundendaten, individuell nutzbar. Alles in allem die Arbeits- und Zeitersparnis, die Sie sich schon immer gewünscht haben.

FAKTUMAT in Stichworten:
voll menuegesteuert – läuft mit einer oder zwei Floppies – Diskettenwechsel (eine Floppy) nur beim Wechsel vom Hauptmenue ins Unterprogramm und umgekehrt – mit Ausnahme des Ausschaltens der Floppy während der Verarbeitung werden alle Fehler abgefangen (z. B. Drukker nicht eingeschaltet – arbeitet mit 1525, 1526 (?), MPS 801, EPSON Drucker und DATA BECKER Interface – voll parameterisiert: Firmenkopf, MWSt. und Rabattsätze, Größe der Dateien beliebig wählbar – 5 Zeilen für Firmenkopf je 30 Zeichen (erste Zeile erscheint auf der Rechnung in Breitschrift – 4 Mehrwertsteuer-Sätze; während der Rechnungsschreibung können also Artikel mit unterschiedlichem Mehrwertsteuer-Satz verrechnet werden – 10 Rabattsätze (Rabattsatz 1 vorbelegt mit 0%), bei der Rechnungsschreibung kann jedem Artikel ein Rabattsatz zugewiesen werden – maximal 1900 Artikel bei 50 Kunden oder 950 Kunden bei 100 Artikel (max. Artikel = [1000-Kunden]*2; max. Kunden = [2000-Artikel]/2) – manuelle Eingabe von Artikeln und/oder Kunde während der Rechnungsschreibung – d. h. es können mehr Artikel verrechnet weden als überhaupt in die Datei passen (bei Verzicht auf Lagerbuchführung) bzw. es können Rechnungen an Kunden geschrieben werden, die nicht erfaßt wurden – integrierte Lagerbuchführung mit Ausgabe einer Inventurliste – Rechnungsbeträge und Datum werden in der Kundendatei festgehalten – Druck von: Rechnung (mit Abbuchen aus Lager), Rechnung (ohne Abbuchen aus Lager), Lieferschein – deutsches detailliertes Handbuch mit Übungs- und Anwendungsteil – deutsche Bedienerführung innerhalb des Programms (z. B. „Artikel nicht vorhanden" anstelle „RECORD NOT PRESENT").

DM 148,–

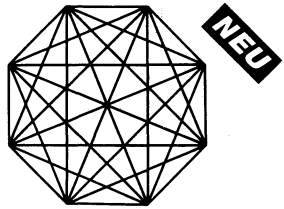


UNI-TAB

Heute schon die Bundesliga-Tabelle von morgen kennen, das geht mit UNI-TAB. Alle Rechnereien, die man ohne dieses Programm nie machen würde, lassen sich in Sekundenschnelle durchführen. Wer will, kann mit simulierten Spielergebnissen den Weltmeister '86 vorausberechnen. Aber nicht nur Fußball-Ligen können tabellarisch erfaßt werden, fast alle Sportarten sind UNI-TAB-fähig. Gag am Rande: für viele Sportarten stehen die bekannten Piktogramme zur Verfügung.

UNI-TAB in Stichworten:
Menuesteuerung über die Funktionstasten mit leicht verständlichen Auswahlmöglichkeiten – Bedienerfreundlich (Mannschaften werden über Kennzahlen gesteuert) – Ligen mit 4 bis 20 Mannschaften können verwaltet werden (6 bis 38 Spieltage möglich) – unsinnige Ligen (z. B. 13 Mannschaften sollen 5 Spieltage absolvieren) sind ausgeschlossen – favorisierte Mannschaft kann während des Programmablaufs durch reverse Darstellung gekennzeichnet werden – Tabelle kann geändert werden (wichtig bei Spielanullierungen) – drei verschiedene Tabellenarten können abgespeichert und später eingelesen werden (die aktuelle Tabelle [unabhängig von der Vollständigkeit eines Spieltages], der komplette Spieltag [Vollständigkeit und Nummer des Spieltages werden automatisch errechnet], die simulierte Tabelle [der Anwender kann so selbst Schicksal spielen und seinen Tip später mit dem tatsächlichen Geschehen vergleichen]) – zwei verschiedene Arten der Saisonübersicht (die statistische Übersicht zeigt an, welchen Tabellenplatz das jeweilige Team bei welchem Punkte- und Torverhältnis an den einzelnen Spieltagen einnahm; die graphische Übersicht zeigt die Leistungskurve jeder Mannschaft) – alle Tabellen und Graphiken sind als Hardcopy auf einem Drukker darstellbar – bei Fehlbedienung (z. B. gewünschte Druckausgabe bei nicht eingeschaltetem Drucker) erscheinen leicht verständliche deutsche Fehlermeldungen.

DM 69,–



SUPERGRAFIK 64

Entdecken Sie die faszinierende Welt der Computergraphik mit SUPERGRAFIK 64, der starken Befehlserweiterung mit den vielseitigen Möglichkeiten. Durch die neue verbesserte Version jetzt noch leistungsstärker.

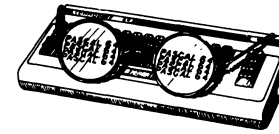
SUPERGRAFIK 64 in Stichworten:

2 unabhängige Graphikseiten (320 x 200 Punkte) – logische Verknüpfung der beiden Graphikseiten (AND, OR, EXOR) – 1 Standard Low-Graphik Seite (80 x 50 Punkte) – Normalfarben Graphik (300 x 200 Punkte) – Multicolor-Graphik (160 x 200 Punkte) – verdecktes Zeichnen (z. B. Text sichtbar, Graphikseite 2 wird erstellt) – Textfenster in der Graphik – 183 Befehle und Befehlskombinationen (1. Für jeden Befehl wählbare Zwischenmodi: Zeichnen, Löschen, Punktieren, Graphik-Cursor bewegen, Zeichnen mit/ohne Farbsetzung, Punkte zählen; 2. Durch einfache Befehle zu steuernde Graphikfiguren: Punkt, Linie, Linienschar, Linie vom Graphik-Cursor, Kreise, Kreisbögen, Ellipse, Ellipsenbögen, selbstdefinierbare Figuren, rotieren und vergrößern dieser Figuren, Rahmen, Feld, Text in Graphik; 3. Weitere Graphikbefehle: Graphikseiten- und Moduswechsel, Graphik löschen, Graphik invertieren, Scrolling von Text und Graphik, Wählen der Rahmen-, Hintergrund-, Zeichen- oder Punktfarbe) – Speichern, Laden von Graphik (auch verdeckt) – Kopieren des Textbildschirms in die Graphikseite – Hardcopies für EPSON, Seikosha GP100VC, Farb(!)drucker Seikosha GP700 und andere mit DATA BECKER Interface – 16! Sprites gleichzeitig auf dem Bildschirm – alle Sprite-Eigenschaften veränderbar – Positionieren und Bewegen (!) von 16 Sprites gleichzeitig und unabhängig voneinander, während das übrige Programm weiterläuft (IRQ) – Sprite-Kollisionsüberprüfung, Joystickunterstützung – automatische Unterbrechung des BASIC-Programms bei Kollisionen (Interrupt), Sprung in Unterbrechungsroutine, dann Weiterführung des Hauptprogramms – komfortable Soundprogrammierung mit Verstellung aller möglichen Sound-Parameter (Lautstärke, Klang, Filter, Tonhöhe, Tonlänge), ebenfalls unabhängig vom übrigen Programmlauf – zahlreichen Programmiertools (MERGE, RENUMBER usw.) – umfangreiche Anleitung – Diskettenprogramm.

DM 99,–

PASCAL 64

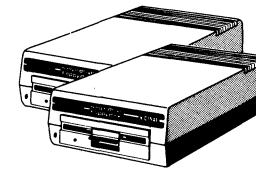
Beim Wort „Compiler" fällt dem Eingeweihten sicher der Begriff „Geschwindigkeit" ein. Ein PASCAL-Compiler sollte jedoch weitere Assoziationen wecken. Strukturiertes Programmieren heißt das Zauberwort. PASCAL wurde eigens zu didaktischen Zwecken entwickelt und erfüllt diese Aufgabe auch heute noch. Der PASCAL 64 Compiler bringt diese phantastische Programmiersprache auf den 64er.
Gerade die neue, verbesserte Version unterstützt die Möglichkeiten des C-64 in jeder Hinsicht und macht leistungsfähige Programme möglich.



PASCAL 64 in Stichworten:

besitzt einen sehr umfangreichen Befehlsvorrat – erlaubt Interruptprogrammierung und bietet Schnittstellen zu Monitor und Assembler – erzeugt sehr schnelle Programme in reinem Maschinencode – unterstützt relative Dateiverwaltung, Graphik und Sound – bietet die Datentypen REAL, INTEGER, CHAR und BOOLEAN sowie Aufzähltypen und POINTER, die zu Datenstrukturen RECORD, SET, ARRAY und PACKED ARRAY kombiniert werden können – erlaubt vorzeitigen Abschluß von Prozeduren mit EXIT, uneingeschränkte Rekursionen und komfortable Verarbeitung von Teilfeldern (Strings) – ist ein ausgereiftes, deutsches Produkt und wird mit ausführlichem Handbuch geliefert.

DM 99,–



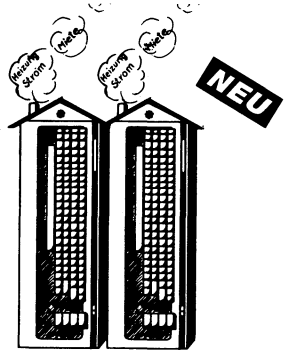
DISKOMAT

Der Umgang mit Diskettenlaufwerken ist für viele noch immer mit Geheimnissen belastet. Andere stören sich an den wenig komfortablen Diskettenbefehlen des BASIC V2. DISKOMAT bringt Abhilfe; alle Diskettenbefehle des BASIC 4.0 stehen zur Verfügung. Außerdem können mit dem Programm SUPERTWIN zwei 1541-Laufwerke wie ein Doppellaufwerk verwaltet werden. Für Benutzer, die sich die Fähigkeiten der Floppy 1541 ganz erschließen wollen, steht der DISK-MONITOR bereit; er macht es endlich möglich, den direkten Zugriff auf einzelne Blocks einfach und bequem vorzunehmen.

DISKOMAT in Stichworten:

Diskettenprogramm – DISK BASIC unterstützt Diskettenbefehle des BASIC 4.0 (CONCAT, HEADER, APPEND, RENAME, OPEN, COLLECT, DSAVE, SCRATCH, DCLOSE, BACKUP, DLOAD, DIRECTORY, RECORD, COPY, CATALOG, DS & DS$) – SUPER TWIN behandelt 2 Laufwerke 1541 wie ein Doppellaufwerk – DISK-MONITOR ermöglicht direkte Analyse und Manipulation von Disketten (direktes Lesen und Schreiben einzelner Blöcke, ändern von Blöcken mittels Bildschirm-Editor, Anzeige des Diskettenstatus, direktes Absenden von Disketten-Befehlen) – ausführliches deutsches Handbuch beschreibt jeden einzelnen der 3 Programmteile.

DM 99,–

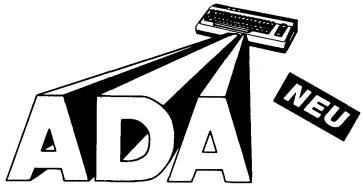


HAUSVERWALTUNG

Jetzt können alle Hausbesitzer aufatmen: das Programm HAUSVERWALTUNG bietet ihnen eine sehr komfortable Verwaltung der Mietwohnungen mit dem COMMODORE 64.
Alles, was Sie dazu brauchen, ist ein COMMODORE 64, ein Diskettenlaufwerk 1541, ein anschlußfähiger Drucker und das obengenannte Programm HAUSVERWALTUNG. Die nachfolgenden und viele weitere leistungsfähige Features ermöglichen eine äußerst rationelle Verwaltung Ihrer Mietwohnungen.

HAUSVERWALTUNG in Stichworten:
Diskettenprogramm – Verwaltung von 50 Einheiten pro Objekt möglich – Stammdatenverwaltung für Häuser und Mieter – Verbuchen der Miete, Nebenkosten und Garagenmieten – Mietkontoanzeige – Haus- und Mieteraufstellung – Mahnungen – Verbuchen der anfallenden Kosten – Kostengegenüberstellung – Jahresendabrechnung mit automatischem Jahresübertrag – umfangreiches deutsches Handbuch.

DM 198,–



TRAININGSKURS zu ADA

Diese Programmiersprache der Zukunft, die das Pentagon in Auftrag gegeben hat, wird jetzt durch DATA BECKER auch dem C-64 Anwender zugänglich gemacht durch den TRAININGSKURS zu ADA, der eine sehr gute Einführung in diese Supersprache bietet. Der dazu gelieferte Compiler liefert ein umfangreiches Subset der Sprache.

ADA in Stichworten:
blockstrukturierte Programme – modularer Aufbau der Programme – ermöglicht die Behandlung von Ausnahmezuständen – Fehlerüberprüfung beim Übersetzen und zur Laufzeit – ermöglicht das einfache Einbinden von Maschinenprogrammen – sehr leichtes Arbeiten mit Programmbibliotheken – Programmdiskette enthält Editor, Übersetzer, Assembler und Disassembler – umfangreiches deutsches Handbuch.

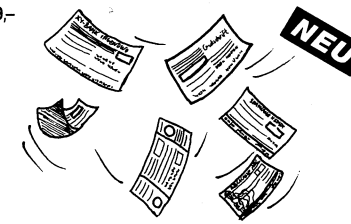
DM 198,–

DATAMAT

Daten verwalten kann ein schier endloses Hantieren mit Karteikästen und Aktenordnern bedeuten; kann aber auch C-64 plus DATAMAT heißen. Dann wird Suchen und Sortieren zum Spaß. Der DATAMAT bietet in seiner neuen Version einiges, was in dieser Preisklasse bisher unvorstellbar schien. Nicht nur Geschwindigkeit und Bedienungsfreundlichkeit wurden weiter verbessert, auch die Anpassung an die meisten Drucker ist inzwischen machbar.

DATAMAT in Stichworten:
menuegesteuertes Diskettenprogramm, dadurch extrem einfach zu bedienen – für jede Art von Daten – völlig frei gestaltbare Eingabemaske – 50 Felder pro Datensatz – 253 Zeichen pro Datensatz – bis zu 2000 Datensätze pro Datei je nach Umfang – Schnittstelle zu TEXTOMAT – läuft mit 1 oder 2 Floppies – völlig in Maschinensprache – extrem schnell – deutscher Zeichensatz auch auf COMMODORE-Druckern – fast jeder Drucker anschließbar – ausdrucken über RS 232 – duplizieren der Datendiskette – verbesserte Benutzerführung – Hauptprogramm komplett im Speicher (kein Diskettenwechsel mehr) – integrierte Minitextverarbeitung – deutsches Handbuch mit Übungslektionen
Sie können:
jeden Datensatz in 2 – 3 Sekunden suchen – nach beliebigen Feldern selektieren – nach allen Feldern gleichzeitig sortieren – Listen in völlig freiem Format drucken – Etiketten drucken.

DM 99,–



ZAHLUNGSVERKEHR

Umfangreicher Zahlungsverkehr kann zur Plage werden. Das Software-Paket ZAHLUNGSVERKEHR übernimmt den größten Teil dieser Arbeit. Außer den notwendigen Fähigkeiten für das Ausfüllen und Auflisten von Überweisungen und Schecks ist der ZAHLUNGSVERKEHR in der Lage, Sammellisten, Einzugslisten etc. selbständig zusammenzustellen.

ZAHLUNGSVERKEHR in Stichworten:
Diskettenprogramm – max. 100 Zahlungsempfänger pro Diskette – drei definierbare Absenderbanken – 25 Zahlungsdateien – 14 frei definierbare Formulare – Kontrolldruck bei Belegeingabe möglich – Eingabe von Rechnungsdaten oder eines Verwendungszwecks – Ausdruck einer Sammel-Überweisungsliste – Korrekturmöglichkeit der einzelnen Zahlungsdateien – arbeitet mit einer oder zwei Floppies – umfangreiches deutsches Handbuch.

DM 148,–

DAS STEHT DRIN:

64 Tips & Tricks Bd. 2 enthält eine Fülle hochkarätiger Programme, Anregungen und viele nützliche Routinen. Ein Buch, das für jeden, der auf COMMODORE 64 eigene Programme schreiben will, eine unentbehrliche Hilfe ist.

Aus dem Inhalt:

- Softwareschutz
- Befehlserweiterung – selbst gemacht!
- Grafik – Zeichendefinition
- Spieleprogrammierung
- Betriebssystem: ROM in RAM
- Betriebssystem-Routinen
- Wie speichert der Computer eine BASIC-Zeile
- Hardware-Tips
- Laufschrift
- Arbeiten mit zwei Bildschirmen
- Modifiziertes INPUT

  und vieles mehr . . .

UND GESCHRIEBEN HAT DIESES BUCH:

Das Autorenteam mit Tobias Weltner, Ralf Hornig und Jens Trapp arbeitet mit dem 64er, seit es diesen Rechner gibt. Alle sind begeisterte Programmierer, die ihre gesamte Erfahrung in dieses Buch gesteckt haben.

ISBN 3-89011-065-7