Commodore
Sachbuch

Wilhelm Besenthal
Jens Muus

Alles über den Plus 4

Lernen und Nachschlagen

Basic-Kurs ★ Dateiverwaltung ★ Anwendersoftware
★ Beispielprogramme ★ Maschinensprache

Alles über den Plus/4

Alles über den Plus/4

Lernen und Nachschlagen

- BASIC-Kurs
- Dateiverwaltung
- Anwendersoftware
- Beispielprogramme
- Maschinensprache

Wilhelm Besenthal
Jens Muus

Markt & Technik Verlag AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Besenthal, Wilhelm:
Alles über den Plus 4 : Lernen u. Nachschlagen ; BASIC-Kurs Dateiverwaltung, Anwendersoftware, Beispielprogramme, Maschinensprache / Wilhelm Besenthal ; Jens Muus. – Haar bei München : Markt-und-Technik-Verlag, 1986.
(Commodore Sachbuch)
ISBN 3-89090-410-6
NE: Muus, Jens:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86

ISBN 3-89090-410-6

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Kösel, Kempten
Printed in Germany

Inhaltsverzeichnis

Vorwort

Als der Plus/4 vor einigen Jahren auf dem Markt erschien, wurde ihm keine sehr große Zukunft vorhergesagt. Bei dem damaligen Verkaufspreis sicher eine richtige Prognose.

Jetzt hat sich das Blatt gründlich gewendet. Was dieser Rechner zum heutigen Preis bietet, ist schon fast sensationell zu nennen. Da gibt es einen Speicher von 64 KByte RAM, diverse Schnittstellen und dazu kommt dann noch die eingebaute Software. Bedienen läßt sich dieser Computer sehr leicht, besitzt er doch eine sehr gute Tastatur. Wer dann seine ersten Programmiererfahrungen sammeln möchte, wird dabei von dem umfangreichen BASIC 3.5 unterstützt.

Mit diesem Buch wenden wir uns an all diejenigen, die mehr über ihren Plus/4 erfahren möchten. Wir haben uns bemüht Fragen zu beantworten, die das mitgelieferte Handbuch offen läßt.

Zur Gliederung des Buches:

Begonnen wird mit wichtigen Grundlagen des Computerns. In den ersten Kapiteln erfahren Sie etwas über den Aufbau und die Arbeitsweise Ihres PLUS/4. Im Anschluß daran folgt eine sehr ausführliche Erklärung der wichtigsten BASIC-Befehle. In diesem Teil des Buches wird auch das Arbeiten mit Kassettenrecorder und Diskettenlaufwerk beschrieben.

Der zweite Teil des Buches befaßt sich mit der Maschinensprache. Nach der Beschreibung des Maschinensprachmonitors folgt eine kurze Erläuterung der Maschinensprachbefehle. Im weiteren Verlauf erfahren Sie zum Beispiel etwas über das Bankswitching des PLUS/4, über die Interruptprogrammierung, wie man einen handelsüblichen Drucker mit Centronicsschnittstelle an den Rechner anschließt, über den

DATA-Erzeuger, den Bau einer Alarmanlage usw. Alles möchten wir Ihnen an dieser Stelle nicht verraten, Sie werden aber sicher überrascht sein, was Ihnen Ihr Rechner alles bietet.

Abschließend folgt dann ein sehr ausführlicher Anhang, der es ermöglicht, wichtige Dinge schnell nachzuschlagen und der auch weiteres Wissen vermittelt. In einem kleinen Computerlexikon werden die wichtigsten Begriffe der Computerei erklärt.

Alle Kapitel sind in sich abgeschlossen. Sie können also jederzeit das Sie am meisten interessierende Kapitel aufschlagen und durcharbeiten. Für knifflige Sachen ist es aber erforderlich, daß Sie gewisse Vorkenntnisse besitzen, die Ihnen in den ersten Kapiteln vermittelt werden. Es bietet sich daher an, das Buch von vorn beginnend durchzulesen.

Zu den Programmen in diesem Buch:

Viele Programme dieses Buches wurden sehr ausführlich erklärt. Diese Erklärungen sollen Ihnen das Verstehen und ein eventuell gewünschtes Ändern erleichtern. Den Autoren ist bewußt, daß nicht immer die Ideallösung eines Problems gewählt wurde, die Programme sollten aber kurz und vor allem übersichtlich bleiben. Sie werden nach Durchlesen des Buches sicher in der Lage sein, die Programme Ihren Bedürfnissen anzupassen. Alle Programme wurden von uns ausführlich getestet und sollten fehlerfrei laufen. Sollte es dennoch Schwierigkeiten geben, überprüfen Sie Ihr eingegebenes Programm oder versuchen Sie den Fehler mit Hilfe der Programmbeschreibung zu finden.

Alle in diesem Buch verwendeten Elektronikbauteile bezogen wir von der Firma Metz Elektronic, Schuhstr. 11, 3110 Uelzen 1.

Besonders danken möchten wir an dieser Stelle unseren Frauen, die auf uns manche Stunde verzichten mußten, und beim Abtippen des Manuskripts nicht unwesentlich geholfen haben. Dank gilt auch unserer Lektorin beim Markt&Technik Verlag Frau Baumann und unserem Freund Hans Jürgen Hadenfeldt, durch deren Hilfe manches Problem schneller gemeistert wurde.

So, und nun nichts wie hinein ins Vergnügen und in die faszinierende Welt Ihres Plus/4. Uns bleibt nun nur noch Ihnen viel Spaß und Erfolg beim Durchlesen und Arbeiten mit diesem Buch zu wünschen.

Jens Muus/Wilhelm Besenthal

1 Der Commodore Plus/4

Mit dem Commodore Plus/4 erschien vor nun fast zwei Jahren ein Rechner auf dem Markt, der im Bereich der Homecomputer neue Maßstäbe setzte. Er verfügt von Haus aus über nicht weniger als 64 KByte RAM, ein ausgesprochen gutes BASIC und nicht zuletzt wurden in ihm noch vier Programme fest eingebaut. Zum gleichen Zeitpunkt erschienen auch seine kleineren Brüder, der C16 und C116.

Seit kurzem wurden die Preise aller drei Rechner drastisch gesenkt. Es folgte ein Verkaufsboom, von dem man wohl vorher bei Commodore nicht zu träumen wagte. Wenn man sieht, für wie wenig Geld man heute den Plus/4 mit Floppy erstehen kann, muß die Kaufentscheidung eigentlich zu seinen Gunsten ausfallen. Durch die immer größer werdende Verbreitung erscheinen auch immer mehr gute Programme auf dem Markt. Durch die Kompatibilität der Rechner Plus/4, C16 und C116 wird die Erstellung vernünftiger Programme noch beschleunigt.

Mit dem Plus/4 kam endlich ein Homecomputer auf den Markt, der in den meisten Anwendungsbereichen begeistert. Er verfügt über eine sehr gute und übersichtliche Tastatur, die einem das oft lange Arbeiten am Gerät erleichtert. Wer eigene Programme erstellen möchte, findet ein sehr umfangreiches BASIC vor. Von dem früheren Minimal-BASIC des VC20 und C64 kann beim Plus/4 keinerlei Rede mehr sein. Das Arbeiten und Programmieren mit dem Befehlssatz macht wirklich Spaß, nicht zuletzt weil auch endlich die Fehlersuche in Programmen durch spezielle Befehle erleichtert wird.

Das Erstellen kürzerer Maschinenprogramme ist ebenfalls sehr einfach. Zum Programmieren besitzt der Plus/4 einen, im Betriebssystem implementierten, Maschinensprachmonitor. Das Arbeiten mit ihm ist recht einfach gehalten.

Wer seinen Plus/4 mit der Außenwelt verbinden möchte, hat die Qual der Wahl. Er besitzt nämlich hierfür eine Menge Schnittstellen. Mit ihnen können fast alle Probleme gemeistert werden. Als wichtigste Verbindung findet man bei ihm den User-Port, eine für Commodore typische Schnittstelle. Die Anschlußbelegung und Programmierung weicht aber von der des VC20 und C64 ab. Des weiteren gibt es den seriellen Bus für die Commodore-Zusatzgeräte, den Kassettenport, den Expansionsport und auch noch zwei Buchsen zum Anschließen der Joysticks. Im User-Port wurde noch eine RS 232 Schnittstelle integriert.

Wie schon erwähnt, ist der Plus/4 zu den Rechnern C16 und C116 kompatibel. Diese Kompatibilität liegt am fast gleichen Hardwareaufbau und am selbem Betriebssystem und BASIC-Interpreter. BASIC-Programme, die auf dem VC20 oder C64 geschrieben worden sind, werden nur dann laufen, wenn Sie keine POKE-, PEEK-, SYS-, USR- oder WAIT-Befehle benutzen.

Ein paar Worte noch zu den wichtigsten Zusatzgeräten.

Die meisten Commodore-Zusatzgeräte können übernommen und angeschlossen werden. Als Drucker sind zum Beispiel 1525, 1526, MPS 801, MPS 802 und MPS 803 anschließbar. Leider verfügen sie nicht über einen deutschen Zeichensatz, es fehlen demnach u.a. die Umlaute. Wir beschreiben daher in diesem Buch den Anschluß von Druckern, die über eine Centronics-Schnittstelle verfügen. Diese haben den Vorteil, daß sie auch an anderen Computern betrieben werden können und zudem meist grafiktauglich sind. Wer seinen Plus/4 ohne Diskettenstation erstanden hat, sollte beim nachträglichen Kauf auf die 1551 zurückgreifen, da diese wesentlich schneller arbeitet als die 1541.

In den folgenden Kapiteln werden wir Ihnen zunächst einige Grundlagen vermitteln.

2 Grundsätzlicher Aufbau eines Computers

Der Aufbau eines Computers ist für die meisten Menschen ein Buch mit sieben Siegeln. Dabei ist sein Innenleben gar nicht so kompliziert. Sie, lieber Leser, sollen nun erfahren, was in dem Kasten so vor sich geht.

Grundsätzlich sollte jeder, der mit Computern umgeht, wissen, was sich hinter Begriffen wie »RAM«, »ROM« oder »Betriebssystem« verbirgt. Keine Angst, jetzt kommt keine hochtechnische Beschreibung des Computerinnenlebens. Der Grundaufbau eines Computers ist folgender: Der Computer besteht lediglich aus einem Schreib-Lese-Speicher, aus einem Nur-Lese-Speicher, einer Ein- und Ausgabe-Einheit und dem Herz des Computers, dem Mikroprozessor. Das ist auch schon alles. Wie man schaltungstechnisch einen funktionstüchtigen Computer aufbaut, brauchen Sie nun tatsächlich nicht zu wissen. Wir müssen uns das vorher Gesagte aber noch etwas genauer ansehen, denn mit unseren Bezeichnungen (Nur-Lese-Speicher usw.) können Sie sicher noch nicht viel anfangen.

In unserem Buch und auch in unserer Zeichnung verwenden wir Bezeichnungen, die wir an dieser Stelle näher erläutern möchten.

RAM Random Access Memory, Schreib-Lese-Speicher
Das RAM ist der Arbeitsspeicher für Ihre Programme und Daten.

ROM Read Only Memory, Nur-Lese-Speicher
Im ROM ist das Betriebssytem, das BASIC und die im Plus/4 eingebaute Software gespeichert.

CPU Central Processing Unit, der Mikroprozessor
Im Plus/4 befindet sich eine CPU vom Typ 7501. Die 7501 ist kompatibel zum Typ 6502.

Wir haben dazu eine Zeichnung angefertigt (Bild 2.1), die Ihnen das Innenleben Ihres Commodore Plus/4 grob verdeutlichen soll.

Wir werden an anderer Stelle noch auf weitere Spezialbegriffe eingehen. Mit RAM, ROM und CPU können wir Ihnen die Arbeitsweise des Computers schon erklären. Sehen Sie sich dazu das Bild 2.1 an.

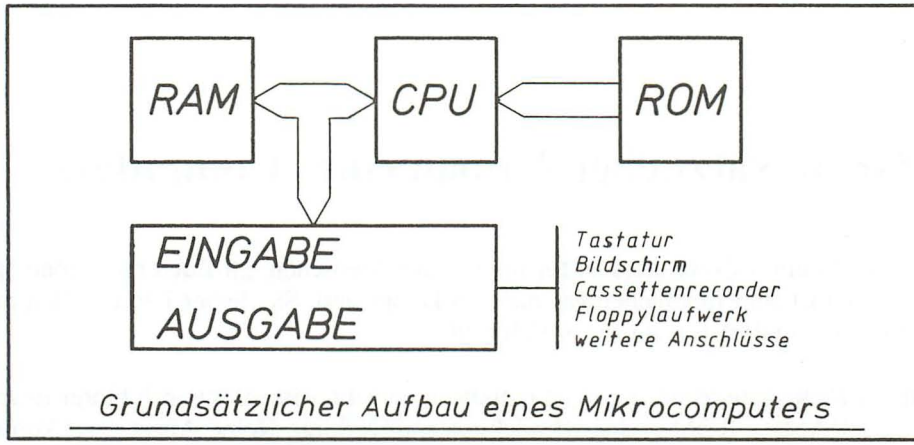


Bild 2.1: *Die Hardwarebauteile des Plus/4*

Das wichtigste Bauteil in Ihrem Plus/4 ist zweifellos die CPU. Die CPU ist das Teil im Computer, das ein wenig Intelligenz besitzt. Dieser Chip kann Zahlen addieren, subtrahieren, Zahlen vergleichen und logische Verknüpfungen vornehmen. Er ist außerdem in der Lage, Speicherinhalte aus dem ROM oder dem RAM auszulesen und die Werte in ihrem eigenen Rechenregister zu speichern. Natürlich kann sie auch Werte an das RAM oder an Ein-/Ausgabe-Bausteine übergeben.

Sie sehen, unsere CPU kann noch nicht einmal 2 mit 2 multiplizieren. Was aber den Mikroprozessor auszeichnet, ist die enorme Geschwindigkeit, mit der die Befehle abgearbeitet werden. Die CPU arbeitet ständig Programme ab. Wenn Sie Ihren Plus/4 einschalten, liest die CPU an einer ganz bestimmten Stelle im ROM (Adresse $FFFC und $FFFD) den entsprechenden Speicherinhalt aus und beginnt mit der Abarbeitung der gespeicherten Programme. Diese Programme sind nicht zu verwechseln mit BASIC-Programmen, es handelt sich vielmehr um die sogenannten Maschinensprachprogramme. Auch die Maschinensprache können Sie erlernen. Wenn Sie Interesse daran haben, lesen Sie das Kapitel über die Maschinensprache mit dem Plus/4.

Hier soll es nun aber um die Programmierung in BASIC gehen. BASIC ist ebenso wie die Maschinensprache eine Programmiersprache. Nur kann unsere CPU BASIC-Befehle leider nicht verstehen, sie kennt nur ihre Maschinensprachbefehle. Damit wir mit unserem Plus/4 nun auch in BASIC programmieren können, ist im ROM ein Übersetzungsprogramm vorhanden. Dieses Programm stellt sich übrigens selbst vor, wenn Sie den Plus/4 einschalten. Auf dem Bildschirm erscheint die Meldung

COMMODORE BASIC V3.5 60671 BYTES FREE
3-PLUS-1 ON KEY F1

Der BASIC-Interpreter (Interpretation = Deutung) hat also die Bezeichnung V3.5. Mit dem Interpreter werden BASIC-Befehle in Maschinensprache übersetzt.

Fassen wir zusammen: Das ROM ist ein Speicherbaustein, das nur ausgelesen werden kann. Der Inhalt kann nicht verändert werden. Sie sehen, auf dem Bild zeigt der Pfeil nach links zur CPU. Daten abspeichern kann man in das RAM oder in Ein-/Ausgabebausteine. Beim RAM spricht man auch von einem »flüchtigen Speicher«. Das heißt, der Speicherinhalt geht bei einer Unterbrechung der Versorgungsspannung (Ausschalten des Computers) verloren.

Die Ein-/Ausgabe, auch I/O (In/Out) genannt, ermöglicht überhaupt erst das Arbeiten mit dem Computer.

Die Verbindungen zwischen den einzelnen Bausteinen haben besondere Bezeichnungen. Zunächst gibt es den Adreßbus. Über den Adreßbus kann jede Speicherzelle einzeln »angesprochen« werden. Die Speicherzellen sind durchnumeriert. Durch den Schaltzustand der Adreßleitungen wird eine ganz bestimmte Speicherzelle gewählt.

Dadurch weiß die CPU aber noch nicht, welcher Wert nun in dieser Speicherzelle enthalten ist. Dafür gibt es den Datenbus. Dieser besteht aus acht Leitungen. Man spricht daher auch vom 8-Bit-Mikroprozessor. Die angesprochene Speicherzelle gibt ihren Inhalt nun auf die acht Datenleitungen, und die CPU kann die Daten somit lesen.

Es gibt im Computer noch eine Reihe anderer Leitungen, z.B. den Steuerbus. Wir möchten aber hier nicht weiter auf diese Dinge eingehen und verweisen auf weiterführende Literatur, die sich mit der Hardware des 6502 befaßt.

Sie haben gelesen, daß der Mikroprozessor Zahlen addieren und subtrahieren kann. Wie ist es nun möglich, eine Multiplikation oder eine Division auszuführen? Tatsächlich kann unsere CPU solche »höhere« Mathematik nicht bewältigen. Aber durch eine geschickte Programmierung im Betriebssystem und im BASIC-Interpreter wird dieses Problem beseitigt. Mehr dazu im Kapitel 20 und 21.

2.1 Beispielprogramm BREAK OUT

Das Haupteinsatzgebiet eines Homecomputers wie des Plus/4 sind zweifellos Spiele. Also spielen wir jetzt einmal. Wir haben dazu das Programm »BREAK OUT« geschrieben. Zugegeben, es handelt sich dabei nicht um Spielhallenqualität, aber Spaß macht es trotzdem. Bei diesem Spiel fliegt ein Ball auf dem Bildschirm hoch und runter. Sie müssen nun versuchen, diesen Ball mit dem Schläger zu treffen. Den Schläger bewegen Sie mit der Taste »Z« nach links und mit der »/«-Taste nach rechts. Starten können Sie das Spiel mit der Leertaste.

Sie haben drei Bälle zur Verfügung. Geben Sie nun das Programm ein, und speichern Sie es bitte unbedingt ab, bevor Sie es zum ersten Mal starten. Wenn Sie das Programm abgespeichert haben, geben Sie den Befehl

RUN

ein. In unserem Programm sehen Sie eine Vielzahl von BASIC-Befehlen, die Sie jetzt natürlich noch nicht kennen. Sie haben das Buch ja auch noch vor sich.

Und nun viel Spaß bei »BREAK OUT«.

```
10 REM BREAK OUT
20 COLOR0,1:COLOR1,2
30 FORI=1630TO1724:READI$:POKEI,DEC(I$):NEXT:SYS1630
35 PRINTCHR$(147)
40 CHAR,5,0,"VERSUCHE: 1":V=1
50 CHAR,20,0,"PUNKTE: 0":P=0
60 FORI=0TO39:POKE3112+I,100:POKE3152+I,45:NEXT
65 IFV=4THEN170:ELSECHAR,14,0,STR$(V)
70 GETKEYA$:IFA$<>" "THEN70
80 XS=INT(RND(0)*40):YD=1:Y1=3:X1=XS:X=X1:Y=Y1
90 XD=INT(RND(0)*3-1)
100 POKE3072+X+Y*40,32:POKE3072+X1+Y1*40,81
105 X=X1:Y=Y1
110 X1=X1+XD:IFX1<0ORX1>39THENXD=-XD:GOTO110
120 Y1=Y1+YD:IFY1<2THENYD=-YD:GOTO120
130 IFPEEK(3072+X1+Y1*40)=119THENYD=-YD:Y1=Y1+YD:GOTO90
140 IFPEEK(3072+X1+Y1*40)=45THENP=P+1:CHAR,28,0,STR$(P):YD=-
YD:GOTO90
150 IFY1>24THENV=V+1:POKE3072+X+Y*40,32:GOTO65
160 GOTO100
170 CHAR,0,9,"SIE HABEN ":PRINTP;" PUNKTE.":PRINT
180 POKE239,0:INPUT"NOCH EIN SPIEL
(J/N)";JN$:IFJN$="J"THEN35:ELSESYS65526
```

```
200 DATA78,A9,77,8D,14,03,A9,06,8D,15,03
210 DATAA9,C0,85,D0,A9,0F,85,D1,A9,01,85
220 DATAD2,58,60,C6,D2,A5,D2,D0,0F,A9,04
230 DATA85,D2,20,11,DB,C9,2F,F0,07,C9,5A
240 DATAF0,1A,4C,0E,CE,A5,D0,C9,E5,F0,F7
250 DATAA0,00,A9,20,91,D0,A0,03,A9,77,91
260 DATAD0,E6,D0,4C,8C,06,A5,D0,C9,C0,F0
270 DATAE0,A0,02,A9,20,91,D0,C6,D0,A0,00
280 DATAA9,77,91,D0,4C,0E,CE
```

3 Zahlensysteme

Bei der Entwicklung von Rechenmaschinen stand man vor dem Problem, wie diese Maschinen intern Zahlen darstellen sollten. Schon Konrad Zuse, der Entwickler des ersten programmierbaren Rechners der Welt, erkannte, daß nur das binäre Zahlensystem in Frage kommen konnte. 1941 war dieser Computer, eine Unzahl von Relais, fertig und funktionierte sehr gut.

Mit diesem binären Zahlensystem arbeiten heute alle Mikrocomputer auf der Welt. Das binäre Zahlensystem kennt nur zwei Ziffern, 0 und die 1. Bei unserem Zehnersystem dagegen gibt es zehn verschiedene Ziffern. Es ist sehr schwierig, auf einer Leitung zehn verschiedene Zustände darzustellen. Bei unserem Computer ist alles viel einfacher, es gibt nur die »1«, d.h. eine hohe Spannung, und die »0«, das bedeutet eine kleine Spannung oder gar keine Spannung.

In der Tabelle haben wir die drei wichtigen Zahlensysteme Binär, Dezimal und Hexadezimal gegenübergestellt. Hier ist wieder ein neuer Begriff aufgetaucht: Hexadezimalsystem. Zur übersichtlicheren Darstellung von Binärzahlen wird in der Computertechnik das sogenannte Sedezimalsystem gebraucht. Allgemein üblich ist aber der Begriff »Hexadezimal«. Es handelt sich hierbei um die Zahlen zur Basis 16.

Binär	Dezimal	Hexadezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
11110001	241	F1
1111111111111111	65535	FFFF

Sie können sich mit Ihrem Plus/4 ja einmal die Schreibweise dieser Hex-Zahlen ansehen. Dazu geben Sie bitte ein:

MONITOR

Sie befinden sich nun im sogenannten Monitorprogramm, das ist ein Teil des Betriebssytems (das Monitorprogramm werden wir im Kapitel über die Maschinensprache erläutern). Geben Sie nun folgendes ein:

M FF00

Auf dem Bildschirm erscheint eine Reihe von Ziffern und Buchstaben. Hierbei handelt es sich, von den zehn Spalten am rechten Bildschirmrand einmal abgesehen, um die hexadezimale Schreibweise. Bei den Hexadezimalzahlen werden die Ziffern mit Werten von 10 bis 15 als Buchstaben dargestellt. Während bei dem Binärsystem schon bei der Zahl 2 ein Übertrag erfolgt, beim Dezimalsystem bei der Zahl 10, gibt es beim Hexadezimalsystem erst bei der Zahl 16 einen Übertrag. Zur besseren Übersichtlichkeit sind in unserem Buch Binär- und Hex-Zahlen gekennzeichnet.

Beispiel einer Hex-Zahl:

$FF00 (erkennbar am Dollar-Zeichen)

Und die Binärzahl:

%01001110 (erkennbar am Prozentzeichen)

Dezimalzahlen sind nicht gekennzeichnet.

Diese Kennzeichnungen werden allgemein in der Computerliteratur verwendet.

Wir haben eingangs erwähnt, daß im Computer der Adreßbus 16 Leitungen »breit« ist und der Datenbus acht Leitungen hat. Durch das binäre System ist es möglich, auf dem Adreßbus Zahlen bis zur Größe von 2^{16} = 65536 und auf dem Datenbus Werte bis 2^8 = 256 darzustellen. Das heißt, in unserem Computer können maximal 65536 verschiedene Speicherzellen angesprochen werden. Beim Plus/4, das sei am Rande er wähnt, sieht das etwas anders aus; hier können durch raffinierte elektronische Schaltungen mit den 16 Adreßleitungen noch mehr Speicherzellen erreicht werden.

Jede Speicherzelle kann Werte von 0 bis 255 annehmen. Vergleiche: acht Datenleitungen binär = 2^8 = 256 (die Null muß natürlich mitgezählt werden).

Dieser sehr lange und komplizierte Abschnitt über den Aufbau des Computers und die Zahlensysteme ist nun bald geschafft. Sie wissen jetzt schon eine ganze Menge über die Computer. Zwei überaus wichtige Begriffe müssen wir uns aber noch ansehen.

3.1 BIT und BYTE

Bit ist die Abkürzung für binary digit (Digitalziffer). Jede der acht Datenleitungen im Computer stellt ein Bit dar. Ein Bit ist die kleinste Einheit im Binärsystem und kann den Wert 1 oder 0 haben (Strom - kein Strom). Jeweils acht Bit ergeben ein Byte. Die Bytes werden hexadezimal dargestellt. Dazu zwei Beispiele:

8 Bit	1 Byte	Dezimal
%0010 1101	= $2D	= 45
%1111 0001	= $F1	= 241

Im Plus/4 haben wir also einen 16-Bit-Adressbus und einen 8-Bit-Datenbus.

Die Bezeichnung Byte steht für acht Bit. Wenn Sie in den folgenden Kapiteln die Bezeichnung Kbyte lesen, so könnten Sie vermuten, daß es sich dabei um 1000 Byte (ein Kilo-Byte) handelt. Das »K« steht aber nicht für »Kilo«, sondern für 2^{10} Bytes, das sind 1024 Byte. Außerdem ist die Bezeichnung für »Kilo« ein kleines »k«. Zur Unterscheidung schreibt man also Kbyte mit großem »K«. Jetzt können Sie auch genau ausrechnen, wie viele RAM-Speicherzellen Ihr Plus/4 hat, nämlich 64 Kbyte, also 64x1024 Byte. Das sind genau 65536 Byte. Das eingebaute ROM hat ebenfalls 64 Kbyte, also auch 65536 Byte.

Soviel zu Bit und Byte. Drucken wir jetzt ein Programm ab, das beliebige Dezimalzahlen in Binärzahlen umwandelt. Schließlich ist der Computer ja zum Programmieren da. Geben Sie den Programmtext genau wie abgedruckt ein und beenden Sie jede Programmzeile mit der RETURN-Taste.

3.2 Programm Umrechnung von Dezimal- in Binärzahlen

```
10 PRINT CHR$ (147)
20 PRINT "GEBEN SIE EINE DEZIMALZAHL EIN"
30 INPUT D : A = D
40 D = D / 2
50 IF D = INT (D) THEN H$ = "0" + H$ : GOTO 70
60 H$ = "1" + H$
70 D = INT (D) : IF D > 0 THEN GOTO 40
80 PRINT A; "= "; H$
90 H$ = ""
100 PRINT
110 GOTO 20
```

Wenn Sie alle Zeilen eingegeben haben, starten Sie das Programm, indem Sie den Befehl

RUN

eingeben und anschließend die RETURN-Taste drücken. Jetzt läuft das Programm, sofern Sie sich nicht vertippt haben. Sollte auf dem Bildschirm ein Text, wie »ERROR IN« erscheinen, müssen Sie das Programm noch einmal mit dem Befehl

LIST

auflisten lassen und mit unserem Listing in diesem Buch vergleichen, eventuell verbessern bzw. die fehlerhafte Zeile neu eingeben. Der Computer gibt Ihnen übrigens eine kleine Hilfe. Er schreibt nämlich hinter »ERROR IN« eine Zahl. Diese Zahl gibt die Programmzeile an, in der der Fehler aufgetreten ist, bzw. in der sich der Fehler be-

merkbar gemacht hat. Doch dazu im Kapitel 19 mehr. Gehen wir jetzt einmal davon aus, daß das Programm fehlerfrei eingetippt wurde. Es erscheint die Aufforderung:

```
GEBEN SIE EINE DEZIMALZAHL EIN
?
```

In der nächsten Zeile steht ein Fragezeichen. Der Computer wartet jetzt auf eine Eingabe von Ihnen. Kommen Sie doch einfach der Aufforderung nach und geben die Zahl

241

ein (RETURN-Taste nicht vergessen). Der Computer rechnet nun die entsprechende Binärzahl aus und druckt sie auf dem Bildschirm aus.

11110001

Anschließend können Sie eine neue Zahl eingeben. Probieren Sie das ein paarmal aus, und lassen Sie auch größere Zahlen berechnen. Allerdings werden Zahlen mit mehr als neun Ziffern nicht mehr genau berechnet.

Eine kurze Erklärung zum Programm. Sie haben das ganze Buch ja noch vor sich, daher wird Ihnen jetzt einiges fremd vorkommen. Lesen Sie trotzdem die Erklärung, dann sehen Sie schon einmal, wie so ein Programm aufgebaut ist.

In der Programmzeile 10 steht ein Befehl, der den Bildschirm löscht.

Zeile 20 läßt den Text auf dem Bildschirm ausgeben.

In Zeile 30 erwartet der Computer die Eingabe einer Zahl und übergibt sie an zwei Variablen (A und D).

Zeile 40 dividiert diese Zahl durch 2.

In Zeile 50 wird festgestellt, ob das Ergebnis eine gerade oder eine ungerade Zahl ergibt. Wenn eine gerade Zahl herauskommt, wird im Ergebnis für die Binärzahl, das ist in diesem Beispiel H$, beginnend von rechts eine »0« eingetragen. Außerdem springt das Programm in die Programmzeile 70.

Wenn das Ergebnis aus Zeile 50 eine ungerade Zahl ergibt, wird hier in Zeile 60 bei der Binärzahl eine »1« eingetragen.

In Zeile 70 wird unsere Dezimalzahl abgerundet. Dann wird festgestellt, ob die Zahl den Wert 0 hat oder größer als 0 ist. Ist die Zahl größer als 0, dann springt das Programm wieder in die Zeile 40. Andernfalls geht es in der Zeile 80 weiter.

In der Zeile 80 wird die eingegebene Dezimalzahl und das binäre Ergebnis ausgedruckt.

Zeile 90 löscht die Variable mit dem Ergebnis.

In Zeile 100 wird eine Leerzeile gedruckt. Das macht den Ausdruck etwas übersichtlicher.

Zum Schluß bekommt der Computer in Zeile 110 den Befehl, wieder zur Zeile 20 zu springen.

Das war doch ganz einfach, nicht wahr? Achten Sie einmal darauf, wie schnell der Computer die Zahlen berechnet. Mit dem Taschenrechner oder mit Bleistift und Papier hätten Sie sicher länger gerechnet.

Noch ein kleiner Tip: Sie möchten sicher dieses Programm nicht jedesmal neu eintippen, wenn Sie eine Zahl umwandeln müssen. Es empfiehlt sich daher, das Programm auf eine Kassette oder eine Diskette abzuspeichern. Nur müssen Sie das Programm dazu erst einmal stoppen. Das erreichen Sie, indem Sie die RUN/STOP-Taste drücken. Probieren Sie das aus, und - nichts geschieht...

Das liegt an der Programmzeile 30. Dort steht der BASIC-Befehl INPUT. Dieser Befehl macht die RUN/STOP-Taste unwirksam. Wir müssen das Programm also abbrechen, wenn gerade kein INPUT-Befehl abgearbeitet wird, z. B. wenn der Computer gerade die Binärzahl berechnet. Geben Sie also eine Zahl ein oder drücken Sie nur die RETURN-Taste und drücken dann unmittelbar darauf die RUN/STOP-Taste. Das Programm stoppt eventuell nicht beim ersten Versuch, aber spätestens beim fünften Mal haben Sie den Dreh heraus und der Computer meldet

BREAK IN 60 (die Programmzeile, die zuletzt abgearbeitet wurde).

In diesem Zusammenhang ein kleiner Tip:

Wenn Sie ein Programm haben, das sich nicht abbrechen läßt, drücken Sie die RUN-/STOP-Taste und halten sie fest. Dann drücken Sie die RESET-Taste und lassen sie wieder los. Wenn dann auf dem Bildschirm die Einschaltmeldung des Monitorprogramms erscheint, können Sie auch die RUN/STOP-Taste wieder loslassen. Nun brauchen Sie nur noch ein »X« einzugeben und die RETURN-Taste zu drücken, dann ist Ihr Computer wieder für BASIC bereit.

Jetzt können Sie das Programm abspeichern. Das machen Sie mit dem Kassettenrecorder:

SAVE "DEZIMAL/HEX"

Oder Sie speichern auf eine Diskette mit:

DSAVE "DEZIMAL/HEX"

Das Programm ist abgespeichert, und Sie können es jederzeit wieder in Ihren Computer neu einladen.

3.3 HEX$ und DEC

Der Plus/4 bietet Ihnen noch eine weitere Möglichkeit, verschiedene Zahlensysteme umzurechnen. Und zwar kann der Rechner durch einen einfachen BASIC-Befehl Hexadezimalzahlen in Dezimalzahlen umwandeln. Dieser Befehl lautet:

HEX$ (n)

(n steht für eine Dezimalzahl von 0 bis 65535)

Geben Sie zum Beispiel

PRINT HEX$ (241)

ein. Auf dem Bildschirm erscheint:

00F1
Das ist ein sehr nützlicher und starker Befehl, den viele andere Computer nicht kennen. Das Plus/4-BASIC kann diese Prozedur aber auch umgekehrt ausführen. So können durch den Befehl

DEC ("hex")

(hex steht für eine Hex-Zahl von 0 bis FFFF)

Hexadezimalzahlen unkompliziert und schnell in Dezimalzahlen umgewandelt werden. Das probieren wir aus. Geben Sie

PRINT DEC ("00F1")

ein und Sie sehen, der Computer bekommt das richtige Ergebnis:

241

Soviel zu den Zahlensystemen. Wir kommen nun zu einem weiteren Beispielprogramm.

3.4 Beispielprogramm LOTTO

Das folgende Programm erzeugt per Zufallsgenerator die bekannten Lottozahlen 6 aus 49. Wir wünschen Ihnen von nun an jede Woche 6 Richtige.

```
10 REM LOTTO
20 DIMA(49)
30 PRINTCHR$(147)
40 PRINTSPC(12)"**LOTTOZAHLEN**"
50 FORI=1TO6
60 LI=INT(RND(0)*49+1)
70 IFA(LI)=1THEN60
80 A(LI)=1
90 L(I)=LI
100 NEXT
110 FORI=1TO6:A(L(I))=0:NEXT
120 FORJ=1TO6
130 FORI=1TO5
140 IFL(I)>L(I+1)THENX=L(I+1):L(I+1)=L(I):L(I)=X
150 NEXTI,J
160 CHAR,3,15,""
170 FORI=1TO6:PRINTL(I);"  ";:NEXTI
180 CHAR,5,24,"NOCH EINEN TIP (J/N)?"
190 GETKEYA$:IFA$="J"THEN30
200 END
```

4 Rechnen mit dem Computer

Wozu dieses Kapitel, werden Sie fragen. Daß ich mit dem Computer rechnen kann, ist doch klar. Stimmt, oft genug hört man ja auch die Bezeichnung »Rechner«. Natürlich kann unser Computer rechnen. Dabei gibt es jedoch einiges zu beachten. Zunächst beherrscht unser Plus/4 selbstverständlich die Grundrechenarten

- Addition
- Subtraktion
- Multiplikation
- Division
- Potenzieren.

Sie wissen sicherlich, daß man bei folgender Aufgabe zu unterschiedlichen Ergebnissen kommen kann, je nachdem, in welcher Reihenfolge gerechnet wird.

2 + 10 x 5 = 52

oder

2 + 10 x 5 = 60

Natürlich ist das erste Ergebnis richtig. Sie erinnern sich sicher an die Regel: Punktrechnen geht vor Strichrechnen. Das heißt, Multiplikation oder Division geht vor Addition oder Subtraktion. Nach dieser Regel rechnet auch der Computer.

Es gibt aber noch weitere Prioritäten:

Potenzieren
vor Multiplikation und Division
vor Addition und Subtraktion

Wenn in unserem Beispiel zuerst 2 mit 10 addiert und dann das Ergebnis mit 5 multipliziert werden soll, dann müßten wir schreiben:

(2 + 10) x 5 = 60

Aufgaben in Klammern werden zuerst ausgeführt. Sie können auch mehrere Klammern verwenden:

((2 + 10) x 5) ^ 3 = 216000

Wenn die zweite Klammer fehlen würde, hätte der Computer zunächst 2 + 10 gerechnet, dann 5 ^ 3 und die Ergebnisse dann multipliziert (12 x 125). Das Ergebnis wäre dann 1500. Sie werden in diesem Buch sehen, daß wir in unseren Programmen teilweise sehr umfangreiche Rechenoperationen durchführen. Dabei muß man sehr genau aufpassen, in welcher Reihenfolge der Computer rechnet.

Sie können alle Rechenoperationen direkt mit dem PRINT-Befehl ausführen. Neben den Rechenoperationen gibt es noch die Vergleichsoperationen, die wir im Kapitel 6 erläutern. Auch die Vergleichsoperationen können Sie ausdrucken lassen bzw. die Ergebnisse in Variablen übernehmen.

4.1 Die Befehle SGN, ABS, SQR, EXP, LOG

Um das Vorzeichen einer Zahl zu erfahren, können Sie den Befehl SGN verwenden.

Syntax:

SGN (x)

Für x kann eine Variable oder eine komplette Formel eingesetzt werden. Als Ergebnis erscheint entweder -1 oder 0 oder 1.

Um den absoluten Wert einer Zahl zu bilden, bedient man sich der Funktion ABS. Dabei wird ein eventuell vorhandenes negatives Vorzeichen in ein positives Vorzeichen umgewandelt.

Syntax:

ABS (x)

Auch hier kann für x entweder ein einzelner Wert oder eine Variable oder eine komplette Formel eingesetzt werden. Das Ergebnis von ABS ist immer eine positive Zahl oder Null.

Der Plus/4 kann auch Quadratwurzeln bilden. Dafür gibt es den BASIC-Befehl SQR.

Syntax:

SQR (x)

Für x darf kein negativer Wert eingesetzt werden, denn es gibt keine Zahl, die quadriert ein negatives Ergebnis ergibt!

Exponenten zur Basis der Zahl e (natürliche Zahl = 2,7182818....) werden mit EXP berechnet.

Syntax:

EXP (x)

x gibt an, wie oft die Zahl e mit sich selbst multipliziert werden muß. Dazu ein Beispiel:

EXP(5) = e * e * e * e * e = 148,413159

Die Funktion LOG (x) berechnet den natürlichen Logarithmus einer Zahl. Der Logarithmus ist der Exponent, mit dem man die Zahl e (2,7182818....) potenzieren muß, um das Ergebnis x zu erhalten.

Syntax:

LOG (x)

Auch hier kann für x entweder nur eine Zahl oder auch komplette Formeln eingesetzt werden. Allerdings darf das Argument in der Klammer keine negative Zahl sein. Der Computer würde eine Fehlermeldung ausgeben.

Es gibt außer den Logarithmen zur Basis e auch noch andere Logarithmen, z. B. zur Basis 10 (Zehnerlogarithmen). Die Zehnerlogarithmen kennt Ihr Plus/4 leider nicht, es ist aber dennoch möglich, sie zu berechnen. Dazu gibt es folgende Formeln:

lg x = 0,434294 * ln x (Bildung des Zehnerlogarithmus)

ln x = 2,302585 * lg x (Umrechnung Zehnerlogarithmus in natürlichen Logarithmus)

ln = Bezeichnung für Logarithmus zur Basis e

lg = Bezeichnung der Zehnerlogarithmen

4.2 Winkelfunktionen

Der Plus/4 beherrscht selbstverständlich auch die Winkelfunktionen. Dafür beinhaltet das BASIC folgende Befehle:

SIN (x)

COS (x)

TAN (x)

ATN (y)

x gibt den Winkelwert an. Dazu aber einige Anmerkungen:

Sie wissen sicher, daß Winkel die Werte von 0 bis 360 Grad haben können. Es kann im Programm aber durchaus vorkommen, daß »Winkel« über 360 Grad berechnet werden sollen, z. B. soll ein Kreis eineinhalb Mal gezeichnet werden. Dazu kann man z. B. bei dem Befehl CIRCLE auch Werte über 360 Grad eingeben, ohne eine Fehlermeldung zu riskieren. Der Computer zieht bei Winkeln über 360 Grad automatisch 360 oder ein Vielfaches davon ab.

Die Winkel dürfen nicht im Gradmaß angegeben werden, sondern müssen erst in das Bogenmaß umgerechnet werden. Das geschieht mit der Formel:

Winkel * PI / 180

Das Ergebnis liegt bei Winkeln von 0 bis 360 Grad zwischen 0 und 2 PI. Die Formel kann auch direkt als Argument in Klammern hinter den Befehlen SIN, COS und TAN stehen.

Das Bogenmaß (Zeichen = rad) ist die Länge des Teilumfangs eines Kreises mit dem Radius 1 (=»Einheitskreis«) für Winkel von 0 bis 360 Grad.

Der Arcustangens ATN ist die Umkehrfunktion des Tangens. Als Argument y wird also der Tangenswert eines Winkels angegeben. Das Ergebnis ist das Bogenmaß des Winkels.

Syntax:

ATN (y)

Und damit beenden wir unsere Reise in die Welt der Mathematik. Bedenken Sie, daß der Plus/4 nicht so genau rechnet wie ein Taschenrechner, denn der BASIC-Interpreter benutzt Näherungsformeln, die zwangsläufig zu Ungenauigkeiten führen.

4.3 Definition von Funktionen mit DEF FN

Mit diesem Befehl können Sie kompletten Formeln einen Namen geben. Über diesen Namen lassen sich die Formeln jederzeit wieder aufrufen. Das Prinzip ist ähnlich wie bei den Variablen. Haben Sie ein Programm, in dem oft mit derselben langen Formel gerechnet werden muß, erspart Ihnen dieser Befehl Tipparbeit und letztendlich auch Speicherplatz. Es ist auch möglich, mehrere Funktionen zu definieren.

Syntax:

DEF FN Name (Variable) = Formel

Name = Für den Namen der Funktion gilt das gleiche wie für Variablennamen: Er muß mindestens ein Zeichen lang sein und das erste Zeichen muß ein Buchstabe sein. Im Namen dürfen keine BASIC-Befehle »versteckt« sein! Der Name der Variablen muß in Klammern angegeben werden. Dabei ist es egal, ob die Variable in der Funktion vorkommt oder nicht.

Beispiel:

```
DEF FN WINKEL (W) = ABS(SIN(W*3.1415/180))
```

DEF FN darf nur in einem Programm verwendet werden. Im Direktmodus würde eine Fehlermeldung erscheinen.

Sie können diese Funktion mit dem Namen WINKEL in Ihrem Programm jederzeit aufrufen. Das geschieht durch:

FN WINKEL (W)

Dabei müßte der Variablen W vorher der Winkelwert zugewiesen werden. Man kann auch statt W den Winkelwert direkt angeben:

FN WINKEL (45)

Es wird hier der Wert 45 der Variablen W automatisch übergeben.

In der Formel dürfen auch mehrere Variablen verwendet werden. Allerdings kann nur eine Variable in der beschriebenen Weise einen Wert automatisch zugewiesen bekommen.

4.4 Der RND-Befehl

Durch den Befehl RND lassen sich Zufallszahlen erzeugen. Diese Zahlen haben »zufällige« Werte. Sie sind größer als 0 und kleiner als 1. Die Zufallszahlen gebrauchen wir z. B. für Spiele, denken Sie an das Simulieren eines Würfelspiels. Ebenso können Zufallszahlen für statistische Berechnungen verwendet werden. Eine weitere Anwendung wäre evtl. das Verschlüsseln von Daten.

Syntax:

RND (x)

x = beliebige Zahl oder Variable.

Zu beachten sind für x allerdings drei wichtige Zahlenbereiche: Positive Zahlen, negative Zahlen und 0. Sehen Sie sich einmal die Zufallszahlen an. Geben Sie dazu ein:

```
10 PRINT RND(1)
20 GOTO 10
```

Vor Ihnen laufen lange Zahlenreihen über den Bildschirm. Auf den ersten Blick erscheinen diese Zahlen auch wirklich »zufällig« zu sein. Aber wieso eigentlich »Zufallszahlen«? Unser Computer ist doch ein rein digital arbeitendes Gerät, das nach genau vorgegebenen Programmen arbeitet. Dann dürften doch Zufallszahlen gar nicht möglich sein!

Stimmt, die »Zufallszahlen« werden nach genau vorgegebenen Formeln berechnet. Außerdem wiederholen sich die Zufallszahlen nach einer gewissen Zeit. Zum Beweis schalten Sie bitte Ihren Plus/4 aus, schalten ihn wieder ein und geben ein:

PRINT RND(1)

Auf dem Bildschirm erscheint die Zahl

1.07870447E-03

RND (positive Zahl) beginnt immer mit dem gleichen Wert. Auch die folgenden Werte haben immer die gleiche Folge. Das liegt daran, daß die Zufallszahlen nach den vorgegebenen Formeln berechnet werden. Sehen wir uns den Befehl RND mit negativem Klammerargument an. Geben Sie bitte ein:

PRINT RND(-1)

Auf dem Bildschirm erscheint

2.99196472E-08

Nun kommt etwas Besonderes: Geben Sie denselben Befehl erneut ein, und Sie werden das gleiche Ergebnis noch einmal bekommen.

Für jede negative Zahl wird eine ganz bestimmte Zufallszahl errechnet. Sie können das mit jedem beliebigen negativen Wert ausprobieren. Z. B. ergibt RND(-220.999) den Wert .910092327.

Die dritte Möglichkeit, Zufallszahlen zu erzeugen, ist der Befehl RND mit dem Argument 0. Bei RND(0) wird die Zufallszahl mit Hilfe der internen Uhr des Plus/4 erzeugt. Sie bekommen durch den Befehl

RND(0)

stets unterschiedliche Zufallszahlen, obwohl diese Zufallszahlen natürlich genaugenommen auch immer mit dem gleichen Wert beginnen. Nur können Sie das kaum überprüfen, da Sie den Befehl nicht immer genau zum gleichen Zeitpunkt (seit Einschalten bzw. Drücken der RESET-Taste) geben können.

Auf den ersten Blick erscheint der Befehl RND(0) als die ideale Art, Zufallszahlen zu erzeugen.

Wir versuchen herauszubekommen, ob die Zufallszahlen gleichmäßig verteilt sind oder ob evtl. bestimmte Zahlen bevorzugt werden. Dazu schreiben wir ein kurzes Programm.

```
10 REM GLEICHVERTEILUNG VON ZUFALLSZAHLEN
20 PRINTCHR$(147)
30 X=INT(RND(1)*10)
40 Z(X)=Z(X)+1
50 PRINTCHR$(19)
60 FORT=0TO9
70 PRINTT;Z(T)
80 NEXT
90 GOTO30
```

Wenn Sie dieses Programm eingeben und starten, werden Sie sehen, daß die gleichmäßige Verteilung der Zufallszahlen durchaus gewährleistet ist. Wir haben das mehrmals geprüft.

In diesem Programm wird die Häufigkeit der auftretenden Zufallszahlen von 0 bis 9 angezeigt. Sie können in Zeile 30 auch folgendes eingeben:

```
30 X=INT(RND(0)*10)
```

RND(0) hat den Vorteil, daß stets unterschiedliche Anfangswerte vorkommen werden.

4.5 Der INT-Befehl

Mit dem INT-Befehl wird der Nachkommaanteil einer Zahl abgeschnitten. Das Ergebnis ist also eine ganze Zahl. Dazu ein Beispiel:

```
A=10.34
PRINT INT(A)
```

Das Ergebnis ist 10. Dieser Befehl kann zum Auf- oder Abrunden einer Zahl verwendet werden. Es ist auch möglich, zur nächsten ganzen Zahl zu runden (das übliche Runden: ab 0,5 wird aufgerundet, sonst abgerundet). Dazu brauchen wir aber schon ein kleines Programm:

```
10 INPUT A
20 PRINT INT(A+.5)
30 GOTO 10
```

Bei diesem Beispiel wird die eingegebene Zahl mit 0,5 addiert, dann wird der Nachkommaanteil abgeschnitten. Dadurch bekommen wir eine Auf- bzw. Abrundung.

Geben Sie bitte den Wert 1,4 ein (die Eingabe muß lauten: 1.4). Der Computer addiert 0,5 dazu und bekommt als Ergebis 1,9. Der Nachkommaanteil wird mit INT abgeschnitten und es bleibt eine 1 über. Wenn Sie dagegen 1,5 eingeben, errechnet der Computer den Wert 2,0 (1,5 + 0,5 = 2,0). Der Befehl INT kann hier keinen Nachkommaanteil abschneiden, also bleibt das Ergebnis eine 2.

Sie können mit INT Zahlen, die mehrere Dezimalstellen besitzen, auf eine beliebige Anzahl von Stellen hinter dem Komma begrenzen. Dazu wieder ein Beispiel: Sie möchten die Zufallszahlen auf zwei Stellen hinter dem Komma begrenzen. Die Zufallszahlen sollen Werte größer Null und kleiner 10 enthalten.

```
10 X = RND(0)*10
20 PRINTX,,INT(X*100)/100
30 GOTO 10
```

In Zeile 10 werden die Zufallszahlen erzeugt und mit 10 multipliziert. Das ergibt Zahlen größer 0 und kleiner 10. Sie haben im vorherigen Abschnitt gesehen, daß die Zufallszahlen meist neun Stellen haben.

In der Programmzeile 20 werden diese Zufallszahlen mit 100 multipliziert, dann durch INT abgerundet und anschließend durch 100 dividiert. Die ursprüngliche Zufallszahl wird links und die gerundete Zahl rechts auf dem Bildschirm ausgedruckt.

Die Multiplikation mit 100 und anschließende Division durch 100 ergeben zwei Stellen hinter dem Komma. Möchten Sie dagegen nur eine Stelle hinter dem Komma haben, dann multiplizieren und dividieren Sie mit 10. Bei drei Stellen mit 1000 usw.

Lesen Sie zum Thema Runden bitte auch das Kapitel 11. Dort wird der Befehl PRINT USING beschrieben. Mit PRINT USING ist es ebenfalls möglich, Werte zu runden.

4.6 Beispielprogramm SUPERHIRN

Zum Thema Zufallszahlen und Rechnen mit dem Computer folgt nun ein Beispielprogramm. Sie kennen sicher das Spiel SUPERHIRN. Jemand denkt sich eine Zahl. Die Aufgabe des Spielers besteht nun darin, diese Zahl durch Probieren und Kombinieren zu erraten.

Unser Programm fragt zunächst danach, wie viele Stellen die Zahl haben soll. Sie können die Frage mit 2 bis 6 beantworten. Anschließend geben Sie bitte ein, welche Zif-

fern in der Zahl vorkommen dürfen. Es sind Ziffern von 2 bis 8 erlaubt. Nach diesen Eingaben beginnen Sie mit Ihrer »Ratearbeit«. Sie haben zehn Versuche. Jede Eingabe muß mit der RETURN-Taste abgeschlossen werden. Der Computer überprüft dann Ihre Zahl mit seiner »erdachten« Zufallszahl. Wenn einer Ihrer Ziffern mit der Zufallszahl übereinstimmt, wird ein ausgefüllter Punkt ausgedruckt. Haben Sie in Ihrer Zahl eine Ziffer, die zwar in der Zufallszahl vorkommt, aber nicht an der richtigen Stelle steht, wird ein offener Punkt ausgegeben.

Die Variable VM beinhaltet die Anzahl der Versuche. Wenn Sie erst ein wenig Übung haben, können Sie die Anzahl der Versuche verringern. Dazu müssen Sie die Programmzeile 55 ändern.

Wir wünschen Ihnen viel Spaß bei SUPERHIRN.

```
10 REM SUPERHIRN
15 PRINT CHR$(147);"SUPERHIRN"
20 INPUT"ANZAHL DER STELLEN (2 BIS 6)";ZM
30 IF ZM<2 OR ZM>6 THEN20
40 INPUT"ZIFFERN VON 1 BIS (MAX.8)";ZH
50 IFZH<2ORZH>8THEN40
55 V=0:R=0:VM=10
60 GOSUB200:REM ZUFALLSZAHL
70 GOSUB300:REM BILDSCHIRM
80 DO UNTIL V=VM
90 GOSUB500:REM EINGABE
95 X=0:R=0
100 GOSUB800:REM PRUEFEN
110 IFR=ZMTHENCHAR,0,VM+8,"":PRINT"   GUT KOMBINIERT!":GOTO160
120 V=V+1
130 LOOP
140 CHAR,0,VM+8,"":PRINT"DIE RICHTIGE ZAHL IST:";
150 FORI=1TOZM:PRINTZ(I);:NEXT:PRINT
160 INPUT"NOCH EIN VERSUCH (J/N)";JN$
170 IFJN$="J"THEN55:ELSEEND
195 :
200 REM ZUFALLSZAHL
205 :
210 FORI=1TOZM:Z(I)=INT(RND(0)*ZH)+1:NEXTI
220 RETURN
295 :
300 REM SPIELBILD
305 :
310 PRINTCHR$(147);"***SUPERHIRN***":PRINT
330 PRINT"10 VERSUCHE, ZIFFERN VON 1 BIS ";ZH:PRINT
340 PRINT"VERSUCH        ERGEBNIS"
350 PRINT
360 FORI=6TO15
370 PRINT"NR.";I-5;TAB(7);":      :"
380 NEXTI
```

```
390 CHAR,1,VM+8,"":PRINT"IHRE ZAHL, BITTE"
400 RETURN
495 :
500 REM EINGABE
505 :
510 DOUNTILX>ZM
520 GETKEYZ$
525 IFZ$=CHR$(20)ANDX=0THEN520
530 IFZ$=CHR$(20)THENZ$=ZA$(X-1):ZA$(X)=" ":X=X-1:GOTO565
540 IFZ$=CHR$(13)ANDX=ZMTHEN560
545 IFZ$=CHR$(13)ANDX<ZMTHEN520
550 IFZ$<"1"ORZ$>CHR$(48+ZH)THEN520
555 IFX=ZMTHEN520
560 X=X+1
565 IFX<1THENX=0
570 ZA$(X)=Z$:ZA$(X+1)=" "
580 GOSUB700:REM AUSGABE
590 LOOP
600 RETURN
695 :
700 REM AUSGABE
705 :
710 FORI=1TOZM
720 CHAR,7+I,6+V,"":PRINTZA$(I)
730 NEXTI
740 RETURN
795 :
800 REM PRUEFEN
802 :
805 FORJ1=1TOZM:Z1(J1)=Z(J1):NEXTJ1
810 CHAR,15,6+V,""
820 FORJ1=1TOZM
830 IFZ1(J1)=VAL(ZA$(J1))THENPRINTCHR$(113);:R=R+1:Z1(J1)=0:
ZA$(J1)="9"
835 NEXTJ1
836 FORJ1=1TOZM
840 FORJ2=1TOZM
850 IFZ1(J1)=VAL(ZA$(J2))THENPRINTCHR$(119);:Z1(J1)=0:ZA$(J2)
="9":GOTO870
860 NEXTJ2
870 NEXTJ1
880 FORJ1=1TOZM:ZA$(J1)="":NEXTJ1
890 RETURN
```

5 Logische Verknüpfungen

Wenden wir uns den logischen Verknüpfungen zu. Es folgt ein kleines bißchen Schaltalgebra. Sie müssen aber kein Mathematiker sein, um dieses Kapitel zu verstehen. Wir werden Ihnen erklären, wozu Sie die logischen Verknüpfungen (im Bedienungshandbuch logische Operatoren genannt) brauchen und was sie bewirken.

Hier sind zunächst die BASIC-Befehle zu diesem Thema:

- AND
- OR
- NOT

5.1 AND (Konjunktion)

Beginnen wir mit der UND-Funktion. Als Definition gilt: Das Ergebnis einer UND-Funktion ist 1, wenn beide zu verknüpfenden Aussagen 1 sind.

Einfacher zu verstehen wäre: Der Motor meines Autos springt an, wenn der Tank gefüllt ist und ich den Zündschlüssel drehe.

An einer Wahrheitstabelle kann man das gut erkennen.

A	UND	B	Ergebnis
0		0	0
0		1	0
1		0	0
1		1	1

So eine Verknüpfung kann man ganz ausgezeichnet mit Binärzahlen durchführen. Das sehen wir uns mit den Zahlen 107 und 210, in binärer Schreibweise %0110 1011 und %1101 0010, an.

		UND		
A	%0110	1011	107	$6B
B	%1101	0010	210	$D2
ergibt	%0100	0010	66	$42

Sie sehen, bei den Dezimalzahlen und den Hex-Zahlen könnte man nicht ohne weiteres zum Ergebnis kommen. Wer weiß schon, daß 107 UND 210 66 ergeben? Das sieht auch etwas eigenartig aus, ist aber logisch. Verwechseln Sie nicht UND oder AND mit + (Plus)!

Bei den Binärzahlen allerdings sieht man sofort, was als Ergebnis herauskommen muß. Schreiben Sie einfach die Zahlen rechtsbündig untereinander und vergleichen Sie Stelle für Stelle.

Wir überprüfen jetzt unsere Funktion mit dem Computer. Geben Sie unser Beispiel ein:

```
PRINT 107 AND 210
```

Tatsächlich erscheint das richtige Ergebnis:

```
66
```

Wir können auch die Hexadezimalzahlen eingeben:

```
PRINT HEX$(DEC("6B") AND DEC("D2"))
```

Wieder gibt es das richtige Ergebnis $0042.

5.2 OR (Adjunktion)

Die Definition für OR, die ODER-Funktion:

Das Ergebnis einer ODER-Funktion zweier Werte ist 1, wenn mindestens ein Wert 1 ist. Das Ergebnis ist 1, wenn der erste Wert oder der zweite Wert 1 ist.

Wir probieren das mit

PRINT 107 OR 210

aus. Der Computer gibt als Ergebnis 251 aus. Das sehen wir uns mit der Wahrheitstabelle und Binärzahlen an.

A	OR	B	Ergebnis
0		0	0
0		1	1
1		0	1
1		1	1

Unser Beispiel:

		ODER		
A	%0110	1011	107	$6B
B	%1101	0010	210	$D2
ergibt	%1111	1011	251	$FB

Die Hex-Zahlen überprüfen wir noch einmal mit dem Computer:

PRINT HEX$(DEC("6B") OR DEC("D2"))

Der Computer gibt $00FB aus.

Außer AND und OR gibt es noch den Befehl NOT, den wir jetzt genauer ansehen werden.

5.3 NOT (Negation)

1 ist nicht 0. »Logisch!« werden Sie sagen, 0 ist auch nicht 1. Sehen Sie, so leicht ist Schaltalgebra. Aber auch hier wollen wir Ihnen nicht die Wahrheitstabelle vorenthalten.

A	NOT A
0	1
1	0

Allerdings funktioniert das nicht mit unserem BASIC-Befehl NOT. PRINT NOT 1 ergibt nicht 0, sondern -1. Der Computer führt den Befehl aber trotzdem richtig aus, nur wird aufgrund der internen Darstellung der Zahlen ein scheinbar falscher Wert ausgegeben.

5.4 Die EXCLUSIV-ODER-Verknüpfung

Für diese Funktion gibt es keinen BASIC-Befehl, jedoch gibt es beim Plus/4 einen Grafik-Befehl, der eine Exclusiv-Oder-Verknüpfung durchführt. Es handelt sich dabei um den Befehl GSHAPE, den wir im Kapitel über die Grafik genau erklären werden. Außerdem werden Sie diese Art der Verknüpfung im Kapitel 20 und 21 wiederfinden. Das Ergebnis einer Exclusiv-Oder-Verknüpfung, kurz EXOR genannt, ist dann 1, wenn entweder die erste Aussage 1 ist oder die zweite. Das Ergebnis ist aber nicht 1, wenn beide Aussagen 1 oder beide 0 sind. Man spricht bei EXOR auch von der »Alternative«. Dazu die Wahrheitstabelle

A	EXOR	B	Ergebnis
0		0	0
0		1	1
1		0	1
1		1	0

Hier wieder unser Beispiel:

	EXOR			
A	%0110	1011	107	$6B
B	%1101	0010	210	$D2
ergibt	%1011	1001	185	$B9

5.5 Anwendungen für die logischen Verknüpfungen

Ein Anwendungsbeispiel haben wir schon genannt: Der Befehl GSHAPE führt logische Operationen durch. Dabei wird ein Teil des Grafikbildschirms mit einem anderen Bitmuster verknüpft.

Ein anderes Beispiel: In Ihrem Plus/4 gibt es Steuerregister, in denen jedes Bit eine andere Bedeutung hat. Solche Register gibt es im TED-Chip. Wir sehen uns das Register mit der Adresse 65286 an. Das Bit 3 zeigt an, ob der Bildschirm 24 oder 25 Zeilen beinhalten soll. Wenn Bit 3 den Wert 0 hat, werden 24 Zeilen dargestellt, bei einer 1 werden 25 Zeilen dargestellt.

Im Normalbetrieb hat der Bildschirm 25 Zeilen, also können wir ihn jetzt einmal auf 24 Zeilen umschalten. Dazu muß Bit 3 den Wert 0 bekommen. Das Register 65286 hat aber noch 7 andere Bits, die für verschiedene Funktionen gebraucht werden. Diese dürfen wir aber nicht verändern. Jetzt benötigen wir die logischen Funktionen:

Mit der UND-Funktion können wir einzelne Bits »ausblenden«, also den Wert 0 geben. Mit der ODER-Funktion dagegen kann man einzelne Bits zwangsläufig zu 1 machen.

Wir geben folgendes ein:

```
POKE(65286),PEEK(65286)AND247
```

Mit dem Befehl POKE kann man beliebige Werte (von 0 bis 255) in Speicherzellen setzen.

Der Befehl PEEK kann den Inhalt einer Speicherzelle auslesen.

Zunächst wird die Zelle 65286 ausgelesen, dann wird der Wert mit 247 UND-verknüpft und anschließend wieder in der Zelle 65286 abgelegt. Nun hat unser Bit 3 den Wert 0, und Sie sehen, daß der Bildschirm oben und unten um jeweils eine halbe Zeile »abgeschnitten« ist. Wieso nun gerade der Wert 247?

Wir haben eine Binärzahl mit dem Wert % 0001 1011

Die Bits werden von rechts nach links bezeichnet, beginnend mit Bit 0. Also hat unser Bit 3 den Wert 1.

Jetzt machen wir einfach eine UND-Verknüpfung:

alter Wert	%0001	1011
UND	%1111	0111
Ergebnis	%0001	0011

%1111 0111 entspricht dem Wert 247 (dezimal).

Nun möchten wir natürlich wieder den alten Zustand bekommen. Bit 3 muß also wieder den Wert 1 bekommen. Dazu brauchen wir die ODER-Verknüpfung.

alter Wert	%0001	0011
ODER	%0000	1000
Ergebnis	%0001	1011

%0000 1000 hat den Wert 8 (dezimal). Geben wir ein:

POKE65286,PEEK(65286)OR8

Wir sehen, es hat geklappt; unser Bildschirm ist wieder normal.

Ein weiteres Anwendungsgebiet für die logischen Operationen ist das Verschlüsseln von Programmen. Sie können Ihre Programme oder die Programmnamen so sehr verschlüsseln, daß sogar Experten eine Weile brauchen, bis sie den Code geknackt haben. Wir möchten hier allerdings nicht weiter auf diese Dinge eingehen, weil wir dabei zu sehr ans »Eingemachte« gehen müßten. Für die Programmierung in BASIC reicht es aus, was wir Ihnen in diesem Kapitel über die logischen Verknüpfungen gesagt haben.

5.6 Beispielprogramm STREICHHÖLZER

Hierbei handelt es sich um ein Spiel gegen den Computer. Auf dem Bildschirm werden zehn Streichhölzer dargestellt. Nun dürfen die Spieler, also Sie und der Computer, abwechselnd ein Streichholz wegnehmen oder wieder hinlegen. Derjenige, der das letzte Streichholz wegnimmt, hat verloren.

Geben Sie jetzt bitte das Programm ein und speichern Sie es ab. Dann starten Sie das Programm mit

RUN

Der Computer fordert Sie dann auf ein Streichholz wegzunehmen (Taste »-«) oder eins hinzulegen (Taste »+«). Dieses Spiel ist eigentlich ganz einfach, und Sie werden bald herausbekommen, was für Sieg und Niederlage verantwortlich ist. Versuchen Sie einmal, das Spiel zu erweitern, indem man 1, 2 oder 3 Hölzchen wegnehmen oder hinlegen darf.

```
10 REM STREICHHOELZER
20 COLOR0,1:COLOR1,2:COLOR2,3:COLOR3,8:GRAPHIC3,1
30 S=10
40 CHAR1,0,24,"ICH HABE 0 HOELZCHEN:DU HAST 0 HOELZCHEN"
50 GOSUB500
60 CHAR,0,1,"MOMENT BITTE, ERST MAL SEHEN, WER"
70 CHAR,0,2,"ANFAENGT."
80 GOSUB490:GOSUB490
90 X=INT(RND(0)*2)
100 GOSUB480
110 IFX=0THENCHAR,0,1,"ICH FANGE AN.":GOTO130
120 CHAR,0,1,"DU FAENGST AN!":GOTO250
130 GOSUB490
140 GOSUB480
150 X=INT(RND(0)*2):Y=INT(RND(0)*6):IFY=1THEN260
160 IFS=1ORS=3THEN260
170 IFS=4ORS=2THEN210
180 IFX=1THENIFH0=0THENGOTO210:ELSEGOTO260
190 IFX=1ANDS/2<>INT(S/2)THENGOTO260:ELSE210
200 GOTO260
210 GOSUB480
220 CHAR,0,1,"ICH NEHME EIN HOELZCHEN WEG."
230 H0=H0+1:S=S-1:GOSUB510
240 IFS=0THEN470
250 GOTO310
260 GOSUB490
270 GOSUB480
280 IFH0=0THEN220
290 CHAR,0,1,"ICH LEGE EIN HOELZCHEN DAZU."
300 H0=H0-1:S=S+1:GOSUB510
310 GOSUB490
320 GOSUB480
330 CHAR,0,1,"DU BIST DRAN (-/+)"
340 POKE239,0:GETKEYA$:IFA$="-"ORA$="+"THEN350:ELSE340
350 GOSUB480
360 IFA$="+"THEN400
370 IFS=1THEN430
380 CHAR,0,1,"DU NIMMST ALSO EINS WEG."
390 H1=H1+1:S=S-1:GOSUB510:GOTO130
```

```
400 IFH1=0THENCHAR,0,1,"GEHT NICHT!!":GOTO310
410 CHAR,0,1,"DU LEGST EIN HOELZCHEN DAZU."
420 H1=H1-1:S=S+1:GOSUB510:GOTO130
430 CHAR,0,1,"SO'N PECH, JETZT HAST DU VERLOREN!"
440 CHAR,0,3,"NOCH EIN SPIEL (J/N)?"
450 POKE239,0:GETKEYA$:IFA$="J"THENRUN
460 GRAPHIC0,1:END
470 GOSUB480:CHAR,0,1,"DU HAST GEWONNEN, PRIMA!":GOTO440
480 FORI=0TO39:CHAR,I,1," ":CHAR,I,2," ":NEXT:RETURN
490 FORI=1TO800:NEXT:RETURN
500 FORI=1TOS:GOTO550
510 IFS<2THENSA=1:ELSESA=S-1
520 CHAR,8,24," ":CHAR,28,24," "
530 CHAR,8,24,STR$(H0):CHAR,28,24,STR$(H1)
540 FORI=SATOS
550 CIRCLE2,I*10+20,50,3,6
560 PAINT2,I*10+20,50
570 FORJ=I*10+20-1TOI*10+20+1
580 DRAW3,J,55TOJ,155
590 NEXTJ:NEXTI
600 PAINT0,(S+1)*10+20,50
610 CIRCLE0,(S+1)*10+20,50,3,6
620 FORJ=SATOS+1
630 DRAW0,J,55TOJ,155
640 NEXTJ:RETURN
```

6 Vergleichsbefehle

6.1 Der IF...THEN-Befehl

An der Schreibweise dieses Befehls - IF...THEN (WENN...DANN) - können Sie schon sehen, daß es sich hierbei um einen Vergleichsbefehl handelt: Wenn eine Bedingung erfüllt ist, macht der Computer etwas anderes, als wenn die Bedingung nicht erfüllt wäre. Dazu gleich ein kurzes Beispiel:

```
10 A=INT(RND(1)*10)+1
20 INPUT"NENNEN SIE EINE ZAHL ZWISCHEN 1 UND 10";X
30 IF X = A THEN PRINT"RICHTIG!":GOTO 10
40 IF X > A THEN PRINT"MEINE ZAHL IST KLEINER":GOTO20
50 PRINT"MEINE ZAHL IST GROESSER"
60 GOTO20
```

In der Zeile 10 wird eine Zufallszahl ermittelt. Die Zahl liegt zwischen 1 und 10. Der dazugehörige Befehl RND ist im Kapitel 4.4 näher erläutert.

Zeile 20 erwartet eine Eingabe.

In den Zeilen 30 und 40 wird die eingegebene Zahl mit der Zufallszahl verglichen. Wenn beide Zahlen gleich sind, druckt der Computer »RICHTIG« aus und springt zurück in Zeile 10.

In Zeile 40 wird festgestellt, ob die eingegebene Zahl größer als die Zufallszahl ist.

In der Zeile 50 steht kein IF...THEN-Befehl. Den können wir uns auch sparen, denn nach dem Vergleich in den Zeilen 30 und 40 steht fest, daß die Zufallszahl größer ist.

Folgende Vergleiche können durchgeführt werden:

=	gleich
>	größer
<	kleiner
<=	kleiner, gleich
>=	größer, gleich
<> oder ><	ungleich
AND	UND-Funktion
OR	ODER-Funktion

Beispiele für Vergleiche:

10 IF A = B AND B > C THEN...

Die Bedingung ist erfüllt, wenn A den gleichen Wert hat wie B UND wenn B größer ist als C.

10 IF A OR B THEN...

Die Bedingung ist nur dann erfüllt, wenn beide Werte 0 sind. Vergleichen Sie das bitte mit der Wahrheitstabelle für die Verknüpfung OR.

10 IF A AND B THEN...

Wenn das Ergebnis der logischen AND-Verknüpfung größer 0 ist, dann ist die Bedingung erfüllt.

10 IF A < C OR B = C THEN...

Diese Bedingung ist nur dann erfüllt, wenn A kleiner ist als C ODER wenn B gleich C ist.

10 IF A THEN...

Bei diesem »Vergleich« wird nur festgestellt, ob A den Wert 0 hat oder ob A ungleich 0 ist. Wenn A<>0 ist, dann ist die Bedingung erfüllt.

Das Ergebnis der Vergleichsoperationen können Sie auch direkt mit dem PRINT-Befehl ansehen. Dazu wieder Beispiele:

PRINT A = B

Wenn A und B gleich groß sind, wird »-1« ausgedruckt. Wenn A und B ungleich sind, wird »0« ausgedruckt.

PRINT A > B

Wenn A größer als B ist, wird »-1« ausgedruckt, sonst »0«.

PRINT A < B

Wenn A kleiner als B ist, wird »-1« ausgedruckt, sonst »0«.

PRINT A <= B

Wenn A kleiner oder gleich B ist, wird »-1« ausgedruckt, sonst »0«.

PRINT A >= B

Wenn A größer oder gleich B ist, wird »-1« ausgedruckt, sonst »0«.

PRINT A <> B oder
PRINT A >< B

Wenn A kleiner oder größer als B ist, wird »-1« ausgedruckt, sonst »0«.

Alle bisher gezeigten Vergleiche können Sie auch mit Strings bzw. Stringvariablen machen. Dabei wird der Vergleich anhand der entsprechenden Werte in der ASCII-Tabelle durchgeführt. Die zwei folgenden Beispiele sind nur mit Zahlen bzw. numerischen Variablen oder Integervariablen erlaubt.

PRINT A AND B

Das haben wir schon bei den logischen Verknüpfungen gesehen. Es wird eine UND-Verknüpfung zwischen A und B durchgeführt und das Ergebnis dezimal ausgedruckt.

PRINT A OR B

Es wird eine ODER-Verknüpfung zwischen A und B durchgeführt, und das Ergebnis wird dezimal ausgedruckt.

6.2 IF...THEN...ELSE

Zu deutsch: WENN...DANN...SONST. Wenn die Bedingung erfüllt ist, dann soll der Computer etwas machen. Wenn sie nicht erfüllt ist, dann wird das ausgeführt, was nach der Anweisung ELSE steht. Denken Sie noch einmal an den normalen IF...THEN-Befehl. Wenn die Bedingung nicht erfüllt ist, wird die nächste Programmzeile ausgeführt.

Hier noch einmal das Beispiel »Zahlenraten«:

```
10 A=INT(RND(1)*10)+1
20 INPUT"NENNEN SIE EINE ZAHL ZWISCHEN 1 UND 10";X
30 IFX=ATHENPRINT"RICHTIG!":GOTO10:ELSEIFX>ATHENPRINT"MEINE
ZAHL
IST KLEINER":GOTO10
40 PRINT"MEINE ZAHL IST GROESSER"
50 GOTO 10
```

Anmerkung: Die Zeile 30 ist länger geworden als erlaubt. Sie kann so nicht eingegeben werden. Sie können aber fast alle BASIC-Befehle bei der Eingabe abkürzen. Das machen Sie hier am besten mit dem Befehl PRINT. Die Abkürzung dafür ist das Fragezeichen. Wenn Sie also die Zeile 30 mit ? statt PRINT eingeben, dann läßt sich auch eine so lange Programmzeile eingeben. Auf die Abkürzungen kommen wir später noch zurück.

Sie sehen am Beispielprogramm, daß wir eine Programmzeile gespart haben. Beachten Sie aber, daß IF...THEN...ELSE immer zusammen in einer Programmzeile stehen müssen. Sie können ELSE nicht in die nächste Zeile übernehmen.

Wir haben bisher nur Zahlen bzw. numerische Variablen in unserem Vergleichsbefehl betrachtet. Es ist aber auch möglich Strings zu vergleichen. Allerdings mit der Einschränkung, daß reine logische Verknüpfungen nicht erlaubt sind. Natürlich können Sie aber AND und OR als Erweiterung eines Vergleichs trotzdem benutzen.

Ein Beispiel:

```
10 INPUT"PROGRAMM BEENDEN (J/N)";A$
20 IF A$="J"THENEND:ELSEGOTO10
```

Abschließend möchten wir noch auf das Kapitel »Programmschleifen« verweisen, denn dort werden auch Vergleiche durchgeführt.

6.3 Der WAIT-Befehl

Mit dem WAIT-Befehl lassen sich beliebige Speicherzellen mit einem vorgegebenen Wert logisch verknüpfen. Wenn das Ergebnis 0 ist, wird die Speicherzelle erneut überprüft. Der Computer wartet (daher »WAIT«); er stoppt das Programm, bis das Ergebnis ungleich 0 ist.

Achtung: Seien Sie sehr vorsichtig mit diesem Befehl. Es muß sichergestellt sein, daß die Verknüpfung mit der angesprochenen Speicherzelle irgendwann 0 ergibt. Das Programm würde sonst bei dem WAIT-Befehl stehenbleiben. Sie hätten dann nur noch

die Möglichkeit, den Computer auszuschalten oder die RESET-Taste zu drücken, um das Programm abzubrechen.

Die Schreibweise des WAIT-Befehls:

WAIT a,x(,y)

a = Speicheradresse (0 bis 65535)
x = Wert für die UND-Verknüpfung (0 bis 255)
y = Wert für die Exclusiv-Oder-Verknüpfung (0 bis 255)

Ein Beispiel: (Geben Sie diesen Befehl bitte nicht ein!)

WAIT 192,1,15

Der Inhalt der Speicherzelle 192 wird mit 15 EXCLUSIV-ODER verknüpft. Anschließend wird das Ergebnis mit 1 UND-verknüpft. Die EXOR- und die UND-Verknüpfung wird so lange wiederholt, bis das Ergebnis ungleich 0 ist.

Achtung: Sie müssen sich vorher darüber im klaren sein, welches Ergebnis Sie bekommen!

Wenn der zweite Wert nicht angegeben wird (WAIT 192,1), wird vom Computer der Wert 0 angenommen, der bei der EXOR-Verknüpfung keine Änderung des Anfangswertes bewirkt. In diesem Fall wird der Inhalt der Speicherzelle 192 mit dem Wert 1 UND-verknüpft. Bei dem WAIT-Befehl handelt es sich auch um einen Vergleichsbefehl, nur bietet WAIT keine Alternative wie IF...THEN. Es wird einfach so lange gewartet, bis der vom Programm oder Programmierer erwartete Wert erreicht wird.

WAIT kann sehr gut zur Abfrage der Tastatur oder der Joysticks gebraucht werden. Ferner eignet sich WAIT auch dazu, bestimmte Ein- und Ausgabefunktionen abzuwarten (Kassettenrecorder oder Diskettenlaufwerk). Voraussetzung dabei ist, daß man genau weiß, welche Speicherzellen wann welchen Wert annehmen.

Wir zeigen Ihnen einige Beispiele, die Sie in Ihren Programmen verwenden können.

WAIT 239,8

In der Speicherzelle 239 wird vom Betriebssystem die Anzahl der gedrückten Tasten abgelegt. Wir warten mit unserem WAIT-Befehl so lange, bis acht Tasten gedrückt wurden. Probieren Sie das aus. Wenn die achte Taste gedrückt wurde, ist der WAIT-Befehl erfüllt und es werden die acht Tasten auf dem Bildschirm ausgegeben.

Sie sehen an unserem Beispiel, daß kein zweiter Wert angegeben wurde. Das heißt, daß die EXCLUSIV-ODER-Verknüpfung der Speicherzelle 239 mit 0 erfolgt. Dadurch ist das Ergebnis praktisch genau dem Wert der Speicherstelle 239. Dieses Ergebnis wird mit 8 UND-verknüpft. Angenommen, es wurden erst 7 Tasten gedrückt, dann würde 8 mit 7 UND-verknüpft. Das Ergebnis wäre 0. Der WAIT-Befehl würde weiter warten, bis die achte Taste gedrückt wird.

Sie können mit WAIT auch die interne Uhr des Plus/4 abfragen bzw. warten, bis die Uhr einen bestimmten Wert angenommen hat. Die Speicherzellen 163 bis 165 zeigen den aktuellen Wert der internen Uhr an.

WAIT 163,1

wartet so lange, bis der Inhalt von 163, verknüpft mit 1, ungleich 0 ergibt. Wenn die Speicherzelle 163 einen Wert ungleich 1 hat, wartet das Programm.

6.4 Beispielprogramm REAKTIONSTEST

Mit dem folgenden Programm können Sie Ihre Reaktionszeit messen. Wenn Sie das Programm starten, färbt sich der Bildschirm grün. Nach einer unbestimmten Zeit ändert sich die Bildschirmfarbe in Rot. Dann müssen Sie die RETURN-Taste drücken. Der Computer druckt anschließend Ihre Reaktionszeit aus.

```
10 REM REAKTIONSZEIT
20 SCNCLR:COLOR0,1:COLOR1,2
30 PRINTSPC(13)"REAKTIONSTEST"
40 PRINT:PRINT"ZUM START DRUECKEN SIE DIE 'G'-TASTE."
50 PRINT:PRINT"WENN SICH DER BILDSCHIRM ROT EINFAERBT,"
60 PRINT:PRINT"DRUECKEN SIE DIE RETURN-TASTE."
70 GETA$:IFA$<>"G"THEN70
80 SCNCLR
90 COLOR0,6
100 X=INT(RND(0)*5000+1000)
110 FORI=1TOX:NEXT
120 COLOR0,3:TI$="000000"
130 INPUTA
140 T=TI
150 COLOR0,1
160 PRINT"SIE HABEN";T/60;"SEKUNDEN GEBRAUCHT."
170 FORI=1TO1000:NEXT
180 PRINT:PRINT"NOCH EINMAL (J/N) ?"
190 POKE239,0:GETKEYA$:IFA$="J"THEN20
200 END
```

7 Die Tastatur des Plus/4

Bevor Sie sich dieses Buch kauften, werden Sie sicher schon einige Erfahrungen mit der Tastatur Ihres Rechners gesammelt haben. Wir setzen in diesem Kapitel voraus, daß Sie die Cursortasten und grundlegende Funktionen der Tasten schon kennen. Der Plus/4 besitzt für einen Rechner dieser Preisklasse eine wirklich vorzügliche Tastatur. Die Aufteilung der Tastatur entspricht dem amerikanischen Standard. So sind im Vergleich zu einer deutschen Schreibmaschine die Tasten Y und Z vertauscht, auch die Umlaute und das ß sucht man vergebens. In diesem Kapitel wollen wir Ihnen das Arbeiten mit der Tastatur und ihre Besonderheiten näherbringen. Ermöglicht einem doch die Kenntnis der einzelnen Tasten das komfortable Bedienen dieses Rechners. Fangen wir also an.

7.1 Der Direktmodus

Wenn Sie Ihren Plus/4 einschalten, dann befinden Sie sich im Direktmodus. Direktmodus bedeutet, daß jede Aktivität, die Sie starten, sofort vom Rechner ausgeführt wird.

Ein Beispiel:

PRINT "DIES IST DER DIREKTMODUS"

Drücken Sie jetzt die RETURN-Taste. Auf dem Bildschirm erscheint nun:

DIES IST DER DIREKTMODUS

Sie sehen, der Rechner ist Ihrem Befehl, den Text »DIES IST DER DIREKTMODUS« auszugeben, sofort nachgekommen. Diese Anweisung wurde aber auch nicht

gespeichert. Sie können auch Rechenoperationen im Direktmodus ausführen. Geben Sie nun folgendes ein und drücken dann wieder die RETURN-Taste:

PRINT 5*3

Sie erhalten als Ergebnis 15.

Sie können im Direktmodus die meisten BASIC-Befehle verwenden, doch kommen viele erst in einem Programm voll zur Geltung. Im Direktmodus sind alle Tasten des Rechners aktiviert. Jeder Tastendruck oder seine möglichen Kombinationen werden sofort ausgeführt.

7.2 Die RETURN-Taste

Mit dieser Taste, die sich rechts auf der Tastatur befindet, haben Sie eben schon Ihre ersten Erfahrungen gesammelt. Diese Taste ist eine der wichtigsten Ihres Plus/4. Wird diese Taste gedrückt, so wird dem Rechner das Ende einer Eingabe mitgeteilt. Der Rechner stellt fest, in welcher Bildschirmzeile der Cursor steht, und versucht den Text der Zeile zu interpretieren, um den Anweisungen Folge zu leisten. Sind die Anweisungen für ihn nicht verständlich, meldet er sich mit einer Fehlermeldung zurück.

Jede Befehlszeile darf aus zwei Bildschirmzeilen bestehen. Ist sie länger, erfolgt ebenfalls eine Fehlermeldung. Befinden sich keine Zeichen vor dem Cursor, so erfolgt lediglich ein Zeilenvorschub. Drücken Sie nun einige Male die Leertaste und dann die RETURN-Taste. Der Cursor befindet sich nun in der ersten Spalte der nächsten Zeile. Wiederholen Sie das Ganze, diesmal aber mit SHIFT/Leertaste. Der Rechner meldet sich mit READY zurück, der Cursor steht drei Zeilen unter der Eingabezeile. Sie sehen, der Rechner hat beim zweiten Mal versucht, die Eingabe zu interpretieren.

Sie können den Cursor auch an den Anfang der nächsten Zeile mit SHIFT/RETURN setzen. Die eingegebenen Anweisungen werden nun aber nicht ausgeführt. Sollte es sich um eine Programmzeile handeln, so wird sie nicht in den Programmspeicher übernommen.

Merken wir uns:

Jede Eingabe von Befehlen muß mit der RETURN-Taste abgeschlossen werden!

SHIFT/RETURN hat optisch zwar den gleichen Effekt, führt aber die eingegebenen Anweisungen nicht aus und übernimmt auch keine Programmzeilen in den Speicher!

7.3 Die CTRL-Taste

Mit dieser Taste können Sie Sonderfunktionen, die die Tastatur Ihnen bietet, aktivieren. Sie müssen die CTRL-Taste immer gemeinsam mit einer anderen Taste betätigen. Der ausschließliche Druck auf CTRL bewirkt nichts. Oben auf der Tastatur befinden sich die Tasten für die Zahleneingabe. Diese Tasten sind von vorne ebenfalls beschriftet. Die erste Reihe der Beschriftung kann mit der CTRL-Taste aktiviert werden. Die ersten acht Tasten beziehen sich auf die Zeichenfarbe. So erscheint nach Drücken von CTRL zusammen mit der Zifferntaste 3 ein roter statt eines vorher schwarzen Cursors. Alle Zeichen, die nun eingegeben werden, erscheinen in Rot. Sie können die Farbe beliebig oft ändern, dem Rechner kommt es nur auf den Inhalt und nicht auf die Farbe einer Befehlszeile an. Wie Sie die Hintergrund- und die Rahmenfarbe verändern können, erfahren Sie im Kapitel über die Grafik bei der Beschreibung des COLOR-Befehles.

Nach Druck der CTRL-Taste, zusammen mit der Zifferntaste 9, ist der Reversemodus eingeschaltet. Zeichen, die jetzt eingegeben werden, erscheinen in der Hintergrundfarbe, umgeben mit der Zeichenfarbe. Ausschalten läßt sich dieser Modus mit CTRL/0, ESC/0 oder durch Druck der RETURN-Taste.

Hier eine Übersicht über die CTRL-Funktionen:

CTRL und	**bewirkt Zeichenfarbe**
1	Schwarz
2	Weiß
3	Rot
4	Cyan
5	Purpur
6	Grün
7	Blau
8	Gelb
9	Reversemodus ein
0	Reversemodus aus
S	Hält ein Programm im Ablauf oder ein Listen durch LIST ohne Meldung an. Die Fortführung erfolgt durch Drücken einer zeichenausgebenden Taste.
FLASH ON	Neu eingegebene Zeichen blinken
FLASH OFF	Schaltet Blinkmodus aus

Und nun einige weitere, für Sie vielleicht neue Kombinationen :

CTRL und	bewirkt
H	schaltet die Kombination SHIFT/COMMODORE-Taste aus
I	schaltet SHIFT/COMMODORE-Taste wieder ein
M	wie RETURN-Taste
Q	wie Cursor Up-Taste
;	wie Cursor **RIGHT**-Taste
N	schaltet Groß-/Kleinschrift ein
:	simuliert Druck der ESC-Taste
T	wie DEL-Taste (löscht Zeichen)
E	Zeichenfarbe Weiß
Pfundzeichen	Zeichenfarbe Rot

Bis auf CTRL/H und CTRL/I können Sie die anderen Kombinationen leichter und logischer durch andere Tasten erreichen. Sie sind hier aber der Vollständigkeit halber erwähnt worden. Der Plus/4 besitzt 16 Grundfarben. Wie Sie die anderen acht einschalten können, erfahren Sie jetzt.

7.4 Die COMMODORE-Taste

Diese Taste besitzt nicht umsonst ihren Namen, da sie COMMODORE-spezifisch ist. Sie werden sie auf Tastaturen anderer Hersteller nicht vorfinden. Sie befindet sich links neben der linken SHIFT-Taste. Mit ihr lassen sich alle Grafikzeichen, die links vorne auf den Tasten abgebildet sind, darstellen. Dabei ist es egal, ob der Großschrift-Grafik-Modus oder der Groß-/Kleinschrift-Modus gewählt wurde. Das Umschalten zwischen diesen beiden Modi geschieht durch gleichzeitiges Drücken der SHIFT- und der COMMODORE-Taste. Hier nun die Kombinationen:

C-Taste und	bewirkt Zeichenfarbe
1	Orange
2	Braun
3	Gelb-Grün
4	Pink
5	Blau-Grün
6	Hellblau
7	Dunkelblau
8	Hellgrün
1. Mal **SHIFT**	Groß-/Kleinschrift-Modus ein
2. Mal **SHIFT**	Großschrift-Grafik-Modus ein

Der Taste wurden aber auch noch andere Funktionen übertragen. Haben Sie ein Kassettenlaufwerk am Plus/4 angeschlossen, und möchten Sie mit LOAD ein Programm in den Rechner laden, so erscheint beim Auffinden des ersten oder des im Namen genannten Programmes die Meldung: FOUND Programmname. Drücken der COMMODORE-Taste veranlaßt den Rechner zum Laden des Programmes. Wird diese Taste beim Ausgeben eines längeren Programmes auf dem Bildschirm, durch LIST, gedrückt, so wird die Listgeschwindigkeit stark reduziert. Auch ein Programmablauf kann mit dieser Taste verlangsamt werden.

Bei allen nicht beschriebenen Kombinationen verhält sich die COMMODORE-Taste wie die SHIFT-Tasten.

7.5 Die SHIFT-Tasten

Mit diesen Tasten erreichen Sie alle Symbole, die vorne rechts auf den Tasten abgebildet sind. Ist der Groß-/Kleinschrift-Modus eingeschaltet, so bewirken Sie, wie bei einer Schreibmaschine, das Umschalten auf Großbuchstaben.

Sie finden auf der Tastatur auch noch die Taste SHIFT LOCK. Sie arbeitet wie eine dauernd gedrückte SHIFT-Taste. Durch einen zweiten Druck wird »SHIFT« wieder abgeschaltet.

Drücken Sie SHIFT-RUN/STOP gleichzeitig, so wird, sollten Sie glücklicher Besitzer eines Diskettenlaufwerkes sein, das erste Programm der eingelegten Diskette in den Rechner geladen und automatisch gestartet, allerdings nur, wenn nach dem Einschalten noch kein Programm geladen wurde. Sollte dies der Fall sein, so wird das zuletzt geladene Programm noch einmal geladen. Näheres hierüber finden Sie im Kapitel über das Arbeiten mit der Diskettenstation.

7.6 Die anderen Tasten des Plus/4

Wir möchten uns an dieser Stelle mit einer kurzen Übersicht begnügen.

Das S in der Übersicht steht für die SHIFT-Tasten.

Drücken von	bewirkt
INST/DEL	Zeichen links vom Cursor wird gelöscht
S INST/DEL	An der Cursorposition wird Platz für ein weiteres Zeichen geschaffen
CLEAR/HOME	Der Cursor wird in die erste Spalte der ersten Zeile gesetzt. Der Inhalt des Bildschirmes bleibt unverändert
S CLEAR/HOME	Der Bildschirm wird gelöscht, der Cursor steht danach in der ersten Spalte der ersten Zeile
RUN/STOP	Unterbricht Programme oder Listen
S RUN/STOP	Lädt und startet das Programm, auf welches die Diskstation gerade positioniert ist. Siehe auch DLOAD.

Sicher wird Ihnen schon aufgefallen sein, daß, hält man eine Taste länger gedrückt, die Zeichen wiederholt ausgegeben werden. Wem das nicht so sehr gefällt, der kann die Tastenwiederholung ausschalten.

POKE 1344,0	Nur die Cursor-, DEL- und Leer-Taste(n) werden wiederholt
POKE 1344,64	Es wird keine Taste wiederholt
POKE 1344,128	Es werden alle Tasten wiederholt

7.7 Die Funktionstasten

Eine Besonderheit des Plus/4 besteht darin, daß die Tasten F1 bis F8 frei programmierbar sind. Man sollte sich angewöhnen, diese Möglichkeit zu nutzen, da sie einem sehr viel Tipparbeit ersparen kann. Beim Einschalten des Rechners oder bei einem Reset werden die F-Tasten vom Betriebssystem mit BASIC-Befehlen belegt. Diese Belegung kann aber jederzeit, d.h. sowohl im Direktmodus als auch in einem Programm, geändert werden.

Die Syntax des Belegungsbefehls lautet:

KEY n,"xyz"

n entspricht hier der Nummer der Taste, die belegt werden soll.

xyz stehen für Buchstaben, Zahlen oder Befehle, die nach Drücken der Taste auf dem Bildschirm erscheinen sollen. Geben Sie jetzt in Ihren Rechner KEY ein, und drücken Sie dann die RETURN-Taste. Es erscheint nun die momentane Belegung der Funktionstasten auf dem Bildschirm.

7.7.1 Funktionstasten belegen

Wir wollen nun einmal die Belegung der F1-Taste ändern, indem wir ihr den Befehl PRINT zuweisen. Geben Sie also ein:

KEY1,"PRINT"

Wenn Sie jetzt, natürlich nach Drücken der RETURN-Taste, die F1-Taste drücken, erscheint auf dem Bildschirm »PRINT«.

Nach einem Programmabruch mit Fehlermeldung benötigt man oft die wichtigsten Variablen mit ihren Inhalten. Nennen wir diese einmal A,B,C und belegen die Taste F2 wie folgt:

KEY2,"PRINTA,B,C"+CHR$(13)

Nach Druck der Taste F2 wird dieser Befehl sofort ausgeführt. Bewirkt wird dieses durch das Anhängen von CHR$(13), wobei 13 den ASCII-Code von »Carriage Return« darstellt. Dadurch wird das Drücken der Taste »RETURN« nachgebildet. Eine Zusammenstellung aller ASCII-Codes finden Sie im Anhang. So bringt z. B. PRINT CHR$(68) ein »D« auf den Bildschirm. Etwas schwieriger wird es, wenn Sie z. B. Ihren Namen durch F3 auf dem Bildschirm ausgeben lassen möchten. Geben Sie bitte einmal folgendes ein:

KEY3,"PRINT"+CHR$(34)+"HUBERT"+CHR$(13)

Nach Druck auf F3 erscheint nun »HUBERT« auf dem Bildschirm. Anführungszeichen, die Befehlen nachgestellt sind, müssen durch den CHR$-Code ausgedrückt werden. Dieser Code ist 34.

Sie können aber auch kürzere, in sich abgeschlossene Routinen auf die F-Tasten legen. Ein Beispiel:

```
KEY4,"FORI=0TO100:PRINTI;:NEXT"+CHR$(13)
```

Aber es geht noch besser. Wie Sie vielleicht wissen, lassen sich die meisten Befehle und Funktionen des Plus/4-BASICs abkürzen. So ist z. B. das Fragezeichen die Abkürzung des PRINT-Befehls. Benutzt man diese Abkürzungen bei der Belegung der F-Tasten, so spart man Speicherplatz, und dies kann manchmal recht wichtig sein, da alle 8 Tasten zusammen nicht mehr als 128 Zeichen beinhalten dürfen.

Im Speicher können Sie die Belegung der F-Tasten mit dem Monitor ab Speicherstelle $055F bis $05E6 erkennen.

So, und nun sind die Tasten durch das Experimentieren sicher nicht mehr sinnvoll belegt. Um den Einschaltzustand zu erreichen, nennt das Anleitungsbuch nur zwei Alternativen. Die erste wäre, den Rechner auszuschalten, die zweite, einen Reset durchzuführen. Beide Arten sind nicht zu empfehlen, da ein eventuell vorhandenes Programm verloren wäre. Hier nun unsere Lösung des Problems:

```
SYS 62359
```

Nach Eingabe dieses Befehls und Drücken der RETURN-Taste sind die Tasten wieder wie im Einschaltzustand belegt, sonst hat sich nichts geändert.

Dabei gibt es allerdings eine Ausnahme. Und zwar ist die F1-Taste jetzt nicht mehr mit dem bekannten SYS-Befehl zum Einschalten der Textverarbeitung belegt, sondern mit dem Befehl GRAPHIC. Das hat folgenden Grund: Beim Einschalten des Rechners, bzw. beim Drücken der RESET-Taste wird unter anderem die hier angegebene Routine aufgerufen. Dabei wird die F1-Taste mit dem Befehl GRAPHIC belegt. Anschließend wird aber noch ein sogenannter Modulreset durchgeführt. Beim Modulreset wird im eingebauten Software-ROM eine Routine aufgerufen, die die F1-Taste umprogrammiert, und zwar mit der Befehlszeile »SYS1525: 3-PLUS-1«.

Hierzu sei noch angemerkt, daß uns die Belegung der F1-Taste mit GRAPHIC sinnvoller erscheint. Wie leicht kann es sonst passieren, daß ein versehentliches Starten der eingebauten Software ein im Speicher befindliches Programm unrettbar löscht!

Zum Abschluß dieses Kapitels folgt nun ein Tastenbelegungsprogramm, welches Ihnen das schnelle Belegen der F-Tasten ermöglichen soll. Dieses Programm wird vorab beschrieben, damit Sie es leicht Ihren Bedürfnissen anpassen können.

Zeile 10: Diese Zeile kann bei der Eingabe entfallen

Zeile 20: Der Bildschirm wird gelöscht, der Cursor in die linke obere Ecke gesetzt, und die Variablen werden ebenfalls gelöscht

Zeile 30: Die aktuelle Belegung der F-Tasten wird angezeigt

Zeile 40: Frage nach der Nummer der Taste, die geändert werden soll

Zeile 50: Warten auf Tastendruck

Zeile 60: Sollte die gedrückte Taste ein Buchstabe oder eine Zahl größer 8 oder kleiner 1 sein, springt das Programm wieder in Zeile 40

Zeile 70: Da die Nummer der Taste bis jetzt nur als String vorhanden ist, wird diese nun in eine Zahl umgewandelt

Zeile 80: Hier wird nach der gewünschten Belegung gefragt

Zeile 90: Durch den POKE-Befehl befindet sich der Rechner im Anführungszeichenmodus. In der Variablen Z wird die aktuelle Länge von A$ festgehalten, danach wartet der Rechner auf die Eingabe von Zeichen.

Zeile 100: Wurde die DEL-Taste gedrückt und befand sich noch kein Zeichen in A$, dann wird DEL nicht ausgeführt und zur Zeile 90 zurückgesprungen.

Zeile 110: Alle Zeichen bis auf ESCAPE, werden auf dem Bildschirm angezeigt. Das Betätigen der ESC-Taste wird aber trotzdem in F$ festgehalten.

Zeile 120: War die gedrückte Taste »RETURN«, dann wird die Eingabe als beendet betrachtet und zur Zeile 160 verzweigt.

Zeile 130: Hier wird A$ die »gedrückte Taste« angehängt

Zeile 140: Wurde DEL gedrückt, dann wird das letzte Zeichen von A$ gelöscht.

Zeile 150: Rücksprung zur Zeile 90, um weitere Zeichen aufzunehmen.

Zeile 160: Falls sich keine Zeichen in A$ befinden, wird die Taste nicht geändert, und es erfolgt ein Sprung zur Zeile 190.

Zeile 170: Frage nach »RETURN«

Zeile 180: Warten auf Tastendruck

Zeile 190: Wurde die Frage mit »J« beantwortet, dann wird nun ein CHR$(13), d.h. »RETURN« angehängt.

Zeile 200: Die Taste wird nun entsprechend der Eingaben belegt (die Belegung befindet sich in A$).

Zeile 210: Der Bildschirm wird gelöscht, und die neue Tastaturbelegung angezeigt.

Zeile 220: Frage nach weiteren Belegungen.

Zeile 230: Warten auf Tastendruck, wenn »J«, dann Verzweigung zur Zeile 20, und das Spiel beginnt von vorn.

Zeile 240: Der Bildschirm wird gelöscht und das Programm beendet.

```
10 REM BELEGUNG DER F-TASTEN
20 PRINT CHR$(147);:CLR
30 KEY
40 PRINT:PRINT"WELCHE TASTE AENDERN ? (1-8)"
50 GETKEY T$
60 IF ASC(T$) 49 OR ASC(T$) 56 THEN 40
70 T=VAL(T$)
80 PRINT:PRINT"WOMIT SOLL TASTE "T"BELEGT WERDEN ?":PRINT
90 POKE 203,15:Z=LEN(A$):GETKEY F$
100 IF F$=CHR$(20) AND Z=0 THEN 90
110 IF F$  CHR$(27) THEN PRINT F$;
120 IF F$=CHR$(13) THEN 160
130 A$=A$+F$
140 IF F$=CHR$(20) THEN A$=MID$(A$,1,Z-1)
150 GOTO 90
160 IF A$="" THEN PRINT:PRINT"TASTE NICHT GEAENDERT":GOTO220
170 PRINT:PRINT"SOLL RETURN FOLGEN ? (J)"
180 GETKEY T$
190 IF T$="J" THEN A$=A$+CHR$(13)
200 KEY T,A$
210 PRINT CHR$(147);:KEY
220 PRINT:PRINT"WEITERE TASTEN AENDERN ? (J)"
230 GETKEY T$:IF T$="J" THEN 20
240 PRINT CHR$(147);:END
```

Zur Bedienung:

Es können alle Zeichen und Zahlen auf die Tasten gelegt werden, also Zeichen wie: Komma, Semikolon, Doppelpunkt und Anführungszeichen. Dies wurde durch die vielleicht anfangs etwas verwirrenden Zeilen 90-140 erreicht. Es werden also keine CHR$-Codes benötigt. Der Rechner befindet sich während der Abfrageschleife ständig im Anführungszeichenmodus. Somit ist es möglich, Cursorsteuerzeichen direkt, d.h. ohne PRINT, auf die Tasten zu legen.

Anführungszeichen brauchen Sie nur bei Belegungen wie PRINT"PLUS/4" einzugeben. Falsch wäre "PRINT"PLUS/4".

Es können alle Escape-Funktionen auf die Tasten gelegt werden. Dies geschieht wie im Direktmodus. Ein Beispiel: ESC/A. Erst also die ESC-Taste drücken und dann A eingeben. Genauso können Sie die CTRL-Taste handhaben.

Vor der Eingabe langer Programme laden Sie dieses Programm einfach ein und siedeln es über Ihr einzugebendes Programm an. Das erreichen Sie z. B. durch »RENUMBER60000«. Jetzt kann man das Belegungsprogramm jederzeit durch »RUN60000« aufrufen. Beim Unterbrechen der Programmierarbeit wird es einfach mit abgespeichert. Ist das neue Programm lauffähig, wird das Belegungsprogramm einfach durch »DELETE 60000-« gelöscht. Und nun viel Spaß beim Programmieren und Tastenbelegen.

7.8 Die Escape-Taste

Oben links auf der Tastatur befindet sich die ESC-Taste. Sie ermöglicht dem Programmierer den Zugriff auf besondere Modi oder Funktionen des Rechners, die sonst nicht erreichbar sind. Da auf sie im Handbuch nur sehr kurz und außerdem teilweise falsch eingegangen wird, werden wir uns in diesem Kapitel etwas näher mit ihr beschäftigen.

Wird diese Taste im Direktmodus gedrückt, dann passiert erst einmal nichts. Der Rechner wartet aber auf einen weiteren Tastendruck. Da er weder den Druck der ESC-Taste noch den darauffolgenden Tastendruck anzeigt, wird das Ganze leicht unübersichtlich. Geben Sie aus diesem Grunde, vor dem Weiterlesen, das am Ende dieses Kapitels abgedruckte Maschinenprogramm für die ESC-Taste ein. Es wird Ihnen durch dieses Programm sicher leichter fallen, mit der ESC-Taste zu arbeiten.

Durch Druck der ESC-Taste und einer implementierten Buchstabentaste schaltet der Rechner entweder einen Modus ein, oder es wird sofort eine Funktion ausgeführt. Sollte es sich um eine nicht implementierte Taste handeln, so kehrt der Rechner zu der »normalen« Tastaturfrage zurück, ohne etwas zu ändern.

Funktion hat hier nichts mit der Mathematik zu tun, sondern heißt in diesem Zusammenhang, daß etwas nur einmal ausgeführt wird. Wird dagegen ein Modus eingeschaltet, so reagiert der Rechner auf Tastendrücke anders, bis der Modus wieder ausgeschaltet wird.

Zuerst möchten wir die ESC-Funktionen bzw. Modi im Direktmodus erklären.

Es gibt drei Modi:

Modus 1 ESC A Automatisches Einfügen von Buchstaben:

Nehmen wir einmal an, daß Sie beim Programmieren einen String verkehrt eingegeben haben. Dieser String sollte heißen: »MEINE FREUNDIN LANGWEILT SICH«. Sie haben aber das Wort » FREUNDIN« vergessen und möchten es nun einfügen. Es gibt nun zwei Möglichkeiten.

Die erste wäre, den Cursor auf das Zeichen hinter »MEINE« zu setzen und dann durch Drücken von »SHIFT/DEL« acht Leerzeichen einzufügen, um diese dann mit dem fehlenden Wort zu füllen.

Die zweite wäre folgende: Drücken Sie die ESC-Taste und dann die A-Taste. Gehen Sie nun mit dem Cursor auf das Zeichen hinter »MEINE« und schreiben Sie nun das Wort »FREUNDIN«. Nach Druck der jeweiligen Taste wird das entsprechende Zeichen eingefügt. Nach Einfügen des Wortes drücken Sie erneut die ESC-Taste und dann »C«. Der Einfügemodus wird hierdurch wieder aufgehoben.

Sie werden sicher mit uns darin übereinstimmen, daß die zweite Lösung die komfortablere ist, da man sich mit ihr das Abzählen der einzufügenden Buchstaben ersparte. Die erste Lösung ist sicher dann die bessere, wenn es um das Einfügen nur eines Buchstabens geht, da sie einem, in diesem Falle, Tastendrücke erspart.

Modus 2 ESC M Schaltet das Scrollen des Bildschirmes aus.

Sollten Sie diese Tastenkombination gedrückt haben, dann geschieht folgendes: Steht der Cursor z. B. in der Mitte des Bildschirmes, und Sie geben ein »PRINT A«, dann erfolgt nach »RETURN« der Ausdruck der Variablen unterhalb der Befehlszeile. Steht der Cursor aber in der letzten Zeile, dann erfolgt der Ausdruck in der ersten Zeile des Bildschirmes. Ausschalten, d.h. in diesem Falle das Wiedereinschalten des Bildschirmscrollens, läßt sich dieser Modus durch ESC L. Am sinnvollsten läßt sich dieser Modus sicher in einem Programm nutzen. Das gleiche gilt für den dritten und letzten Modus.

Modus 3 ESC R Verkleinert und löscht den Bildschirm.

Nach Druck von ESC und dann »R« wird der Bildschirm gelöscht und dann, am linken und rechten Rand um je eine Spalte, und am oberen und unteren Bildschirmrand um je eine Reihe, verkleinert. Durch ESC N oder zweimaliges Betätigen von »HOME« wird dieser Modus wieder ausgeschaltet und der Bildschirm gelöscht. Auch dieser Modus läßt sich eigentlich nur sinnvoll in einem Programm nutzen.

Wir kommen nun zu den Funktionen. Erinnern Sie sich bitte: Der Unterschied zwischen Modus und Funktion besteht darin, daß der Modus wieder ausgeschaltet werden muß. Eine Funktion dagegen wird nur einmal und sofort ausgeführt.

Funktion 1 ESC D

Nach dieser Tastenkombination wird die Bildschirmzeile, in der sich der Cursor befindet, gelöscht. Der Bildschirm wird nach oben gescrollt, und der Cursor an den Anfang der Zeile gesetzt. Sollte die Zeile mehr als vierzig Zeichen lang sein, so werden auch diese gelöscht. Diese Funktion läßt sich also auch gut zum Editieren von Programmzeilen verwenden, die noch nicht mit »RETURN« abgeschlossen wurden. Wurde schon ein »RETURN« zum Abschluß der Programmzeile eingegeben, so verschwindet sie nur vom Bildschirm, nicht aber aus dem Programmspeicher.

Funktion 2 ESC I

Die Zeile, in der der Cursor steht, und alle nachfolgenden Zeilen werden nach unten gescrollt. So entsteht eine Leerzeile, an deren Anfang sich der Cursor befindet. Das kann im Direktmodus und für das Editieren gebraucht werden.

Funktion 3 ESC J

Der Cursor wird an den Anfang der Zeile gesetzt, in der er steht. Auch hier kann die Zeile länger als vierzig Zeichen sein. Diese Funktion eignet sich ebenfalls zum Berichtigen von Programmen, insbesondere dann, wenn der Cursor am Ende der Zeile steht und am Anfang etwas zu berichtigen ist. Es geht schneller, als wenn man den Cursor mit den Cursortasten zum Anfang der Zeile bewegt.

Funktion 4 ESC K

Funktioniert wie ESC J, nur daß der Cursor jetzt an das Ende der Zeile gesetzt wird, in der er steht.

Funktion 5 ESC P

Steht der Cursor z. B. in der Spalte 26 einer Textzeile und es wird diese Funktion ausgeführt, dann werden die Spalten 0 bis 26 gelöscht und 26 Leerzeichen erzeugt. Der Cursor bleibt an der Löschposition stehen. Auch diese Funktion eignet sich zum Berichtigen.

Funktion 6 ESC Q

Steht der Cursor wieder in Spalte 26, erfolgt das Löschen aller Zeichen rechts vom Cursor. Anwendung z. B. wie oben.

Funktion 7 ESC V

Der Bildschirm wird um eine Zeile nach oben gescrollt und eine Leerzeile am unteren Bildschirmrand erzeugt. Dies läßt sich sonst nur erreichen, wenn der Cursor in der letzten Zeile steht und dann ein »CURSOR DOWN« ausgeführt wird. Bei dieser Funktion ist es also unerheblich, an welcher Stelle des Bildschirmes der Cursor gerade steht.

Funktion 8 ESC W

Der Bildschirm wird um eine Zeile nach unten gescrollt und eine Leerzeile am oberen Bildschirmrand erzeugt. Dies läßt sich sonst nicht erreichen.

Funktion 9 ESC X

Wird nach Druck der ESC-Taste »X« gedrückt, so kehrt der Rechner in die »normale« Tastaturabfrage zurück, ohne einen ESC-Modus oder eine ESC-Funktion einzuschalten bzw. auszuführen. Es kann aber statt X jede Ziffern- oder jede Buchstabentaste, die nicht implementiert ist, gedrückt werden. Das Ergebnis ist das gleiche.

Funktion 10 ESC O

Angenommen, Sie befinden sich im Direktmodus und haben durch »RVS ON« und durch »FLASH ON« den REVERSE-Modus bzw. den FLASH-Modus eingeschaltet. Um diese Modi wieder auszuschalten, genügt nur: ESC O. Für das Programmieren bietet diese Funktion noch etwas Sinnvolles: Sollten Sie sich im Anführungszeichen-Modus befinden - dieser Modus wird durch das erste Anführungszeichen eingeschaltet und durch das zweite wieder ausgeschaltet - und möchten Sie das zuletzt eingegebene Zeichen berichtigen, dann können Sie diesen Modus durch ESC O ausschalten. Gehen Sie nun mit »CRSR LEFT« zurück und berichtigen das falsche Zeichen. Sie müssen in solchen Fällen immer den Anführungszeichen-Modus verlassen, da auf dem Bildschirm sonst das Steuerzeichen für »CRSR LEFT« erscheint.

Nun noch eine Berichtigung zum Handbuch:

Mit ESC O wird der Einfügemodus N I C H T aufgehoben! Dies wird nur durch ESC C erreicht.

Die meisten Modi und Funktionen eignen sich vorzüglich zur Textverarbeitung. Während man Dinge wie zeilenweises Löschen oder automatisches Einfügen bei anderen Rechnern erst umständlich programmieren muß, reicht beim C16 der Druck einiger Tasten aus. Wie wird nun aber dem Rechner mitgeteilt, daß er z. B. den Modus für das automatische Einfügen einschalten soll? Geben Sie einmal im Direktmodus folgendes ein:

```
PRINT CHR$ (27) "A"
```

Gehen Sie nun einmal mit dem Cursor auf eine beschriebene Bildschirmzeile, und drücken Sie dann eine Buchstabentaste. Sie sehen, der Rechner befindet sich im Einfügemodus. In Programmen sieht die Befehlszeile genauso aus, nur daß dann eine Zeilennummer vorangestellt ist. Nach diesem Verfahren lassen sich alle ESC-Modi/Funktionen in einem Programm nutzen. Wir möchten Ihnen an dieser Stelle empfehlen, die ESC-Taste so oft wie möglich zu nutzen. Optimal eingesetzt, kann Sie Ihnen viel Arbeit ersparen.

7.8.1 Die Fensterprogrammierung

Wie Ihnen sicher schon aufgefallen ist, fehlen bei der Beschreibung der ESC-Funktionen noch deren zwei. Es sind die Befehle ESC T und ESC B. Sie dienen der Fenstererstellung auf dem Plus/4. Ein Fenster ist ein Bildschirmausschnitt. Man kann so zum Beispiel das Ergebnis einer Befehlsfolge in einem begrenzten Teil des Bildschirms sichtbar machen, ohne daß die Eingabe im anderen Teil des Bildschirms zerstört wird. Kommen wir nun zur Bedeutung der ESC-Sequenzen:

ESC T

Jedes Fenster wird beim Plus/4 durch zwei Parameter festgelegt. Dies ist zum einen die linke obere Ecke und zum anderen die rechte untere Ecke des Fensters. Mit der oben stehenden Tastenkombination wird an der momentanen Cursorposition die linke obere Ecke definiert. Es fehlt jetzt nur noch die Definition der rechten unteren Ecke. Dies geschieht mit:

ESC B

Wird der Cursor nach ESC T auf eine andere Stelle des Bildschirms bewegt, so wird mit ESC B die zweite notwendige Eingabe getätigt. Zur Demonstration ein kleines Beispiel:

Gehen Sie durch Drücken der Taste HOME mit dem Cursor in die linke obere Bildschirmecke. Jetzt folgt ESC T. Wir haben die linke obere Ecke unseres Fensters festgelegt. Nun gehen wir mit den CURSOR-Tasten in die zehnte Spalte der dritten Zeile, und es folgt dann: ESC B. Unser Fenster ist definiert. Alle Eingaben von der Tastatur, oder auch alle Ausgaben eines Programms erfolgen ab jetzt in diesem, von uns definierten Teil des Bildschirms. Mit der Taste HOME wird der Cursor nun in die linke obere Ecke unseres Bildschirms gesetzt. Wird zweimal hintereinander diese Taste gedrückt, so wird das Fenster, nicht sein Inhalt, gelöscht.

Diese Art der Bildschirmgestaltung kann auch in Programmen verwendet werden. Hier nun die notwendigen Programmbefehle:

100 PRINT CHR$(27)"T"

Diese Zeile definiert die linke obere Ecke des Fensters.

200 PRINT CHR$(27)"B"

Mit dieser Zeile wird die rechte untere Ecke definiert.

300 PRINT CHR$(19) CHR$(19)

Durch diese Befehle wird das definierte Fenster gelöscht und der Cursor in die linke obere Ecke des Bildschirms gesetzt. Sollen in einem Programm Fenster umdefiniert werden, der Plus/4 läßt leider nur die Definition eines Fensters zu, so muß hierzu ein sehr großer programmtechnischer Aufwand betrieben werden. Wir haben für Sie natürlich eine elegantere Lösung parat.

7.8.2 Fensterprogrammierung mit dem POKE-Befehl

Für die Bildschirmgröße besitzt der Plus/4 vier Speicherstellen. Diese Speicher befinden sich im RAM und können daher von uns mit POKE leicht geändert werden. Hier eine Übersicht:

Speicherstelle	Inhalt	max. Wert	min. Wert
2021	unterer Rand	24	0
2022	oberer Rand	24	0
2023	linker Rand	39	0
2024	rechter Rand	39	0

Mit diesen POKE-Befehlen läßt sich jedes beliebige Fenster definieren, man muß sich nur innerhalb der oben angeführten Grenzen bewegen. Die Grenzen ergeben sich durch den Ursprung des Bildschirms. Er liegt in der nullten Zeile und nullten Spalte. Zum POKEN eines Fensters folgt ein Beispiel:

```
10 POKE 2021,19:POKE 2022,15
20 POKE 2023,19:POKE 2024,39:PRINT CHR$(19)
```

Mit diesen Zeilen wird ein Fenster definiert, dessen obere linke Ecke in der 16. Bildschirmzeile liegt. Die linke untere Ecke befindet sich in der 20. Zeile. Die Ränder wurden wie folgt festgelegt: Der linke Rand befindet sich in der 20. Spalte, und der rechte Rand in der 39. Spalte. Der Befehl PRINT CHR$(19) setzt nach der Definition den Cursor in die linke obere Ecke unseres Fensters. Wer's nicht glaubt, der möge es doch bitte einmal ausprobieren.

Um Ihnen den Umgang mit der ESC-Taste etwas zu erleichtern, haben wir ein kleines Maschinenprogramm für Sie geschrieben.

Bei dem Programm handelt es sich um einen BASIC-Lader. Solche Programme »POKEen« Maschinenprogramme in den Speicher. Speichern Sie das Programm vor dem Starten ab, da es sich nach der Ausführung selbst löscht.

Das BASIC-Programm wird mit »RUN« gestartet. Sollte nun die Meldung »FEHLER IN DATAS« auf dem Bildschirm erscheinen, dann stimmt die Summe der Datas nicht mit der Prüfsumme in Zeile 70 überein. Überprüfen Sie deshalb alle Daten und berichtigen Sie den Fehler. Danach ist das Programm erneut abzuspeichern.

Wenn Sie alles richtig eingegeben haben, erscheint nach dem Starten in der ersten Bildschirmzeile ein schwarzer Balken. Drücken Sie jetzt die ESC-Taste. Die Farbe des Bildschirmrandes muß sich ändern. Wird jetzt »M« gedrückt, dann sehen Sie sofort diesen Buchstaben reverse in der ersten Bildschirmzeile, und der Bildschirmrand erhält seine alte Farbe zurück. Sie haben nun das Scrollen des Bildschirms ausgeschaltet. Um es wieder einzuschalten, brauchen Sie nur wieder die ESC-Taste und dann »L« zu drücken. Jetzt ist auch das reverse M wieder verschwunden.

Auch der Einfügemodus wird angezeigt. Es erscheint dann ein reverses A.

Wir haben der ESC-Taste noch eine weitere Funktion zugeordnet. Drücken Sie einmal nach Verändern der Rand-, Hintergrund- und Zeichenfarbe erst die ESC-Taste und dann die INST/DEL-Taste. Der Bildschirm erscheint, bis auf die erste Zeile, in den gleichen Farben wie nach dem Einschalten oder nach einem Reset. Ein eventuell im Speicher vorhandenes Programm wurde aber nicht zerstört. Ausschalten läßt sich das Ganze auch, und zwar durch ESC und dann CLEAR/HOME.

Wird dieses Programm wieder benötigt, so kann es mit SYS 1630 wieder gestartet werden. So, und nun viel Spaß und gutes Gelingen beim Schreiben von Programmen, die die ESC-Modi und Funktionen nutzen.

Es folgt nun das BASIC-Ladeprogramm

```
10 REM BASICLADER
20 FORI=1630TO1751
30 READI$
40 POKEI,DEC(I$)
50 T=T+DEC(I$)
60 NEXT
70 IFT<>13999THENPRINTCHR$(130);"FEHLER IN DATAS";CHR$(132):END
80 SYS1630:NEW
100 DATA 78,A9,6F,8D,14,03,A9,06,8D,15,03
110 DATA A9,00,85,D0,58,60,A4,C6,C0,40,F0
120 DATA 38,A5,D0,D0,06,C0,34,D0,30,F0,04
130 DATA C0,34,F0,2A,49,FF,85,D0,AD,19,FF
140 DATA 49,FF,8D,19,FF,C0,00,D0,06,20,4E
150 DATA D8,20,0B,F3,C0,39,D0,10,78,A9,0E
160 DATA 8D,14,03,A9,CE,8D,15,03,58,A9,20
170 DATA D0,0C,A6,CD,D0,06,EA,A9,11,20,D2
180 DATA FF,A9,A0,A2,27,9D,00,0C,CA,10,FA
190 DATA AD,E9,07,10,05,A9,8D,8D,01,0C,AD
200 DATA EA,07,10,05,A9,81,8D,03,0C,4C,0E,CE
```

Mit diesem Programm ist das Kapitel über die Tastatur beendet. Gewöhnen Sie es sich an, die Funktionstasten und die ESC-Funktionen häufig zu gebrauchen, stellen sie doch eine erhebliche Arbeitserleichterung dar.

7.9 Beispielprogramm AUTORENNEN

Das folgende Programm nutzt einige ESC-Funktionen. Auf dem Bildschirm erscheint eine »Straße« mit reichlich Kurven. Sie müssen nun Ihr »Auto« so steuern, daß es nicht an den Straßenrand gerät. Nach links lenken Sie mit der 'Z'-Taste, nach rechts mit der '/'-Taste. Mit der Leertaste beginnt das »Rennen«.

Auch hier gilt wieder: Speichern Sie das Programm unbedingt ab, bevor Sie es mit RUN starten! Im Programm wird eine Maschinenspracheroutine gestartet, die den Rechner problemlos lahmlegen kann, wenn sie fehlerhaft eingetippt wird.

Und nun viel Spaß beim Autofahren.

```
10 REM AUTORENNEN
20 PRINTCHR$(147):COLOR0,1:COLOR1,2
30 FORI=1630TO1731:READH$:POKEI,DEC(H$):NEXT
40 FORI=1TO25:PRINTCHR$(27)"W";:PRINTTAB(13)"I"SPC(16)"I"CHR$
(19):NEXT
50 L=13:S=16:VOL8
60 POKE239,0:GETKEYA$:IFA$=" "THENTI$="000000":GOTO70:ELSE60
70 SYS1630
80 X=X+1:IFX>10THENX=0:S=S-1
90 LA=INT(RND(0)*22+1):IFLA<LTHENLB=-1:ELSELB=1
100 FORI=LTOLASTEPLB:PRINTTAB(I)"I"SPC(S)"I"CHR$(27)"W"CHR$(19)
110 IFPEEK(211)=1THEN130:ELSE NEXT
120 L=LA:GOTO80
130 SYS65418:REM VEKTOREN WIEDERHERSTELLEN
140 FORI=1TO10:POKE65286,PEEK(65286)OR7:POKE65287,15:SOUND1,
200,1
150 FORJ=1TO10:NEXTJ:POKE65286,27:POKE65287,8:NEXTI
160 CHAR,0,15,"SIE HABEN          ":CHAR,9,15,STR$(TI)+" PUNKTE."
170 PRINT:PRINTCHR$(27)"Q";
180 POKE239,0:INPUT"NOCH EINE FAHRT (J/N)";JN$
190 IFJN$="J"THENRUN
200 END
210 DATA 78,A9,7F,8D,14,03,A9,06,8D,15,03
220 DATA A9,D4,85,D0,A9,0F,85,D1,A9,01,85
230 DATA D2,A9,F5,A0,00,91,D0,84,D3,58,60
240 DATA C6,D2,A5,D2,D0,0F,A9,06,85,D2,20
250 DATA 11,DB,C9,5A,F0,1E,C9,2F,F0,25,A0
260 DATA 00,B1,D0,C9,09,F0,07,A9,F5,91,D0
270 DATA 4C,0E,CE,A9,5A,91,D0,A9,01,85,D3
280 DATA 4C,0E,CE,A5,D0,C9,C0,F0,E0,C6,D0
290 DATA 4C,94,06,A5,D0,C9,E7,F0,D5,E6,D0
300 DATA 4C,94,06
```

8 Variablen

Geben Sie bitte folgendes in Ihren Rechner ein:

A=5

PRINT A

Nun druckt der Computer eine 5 aus. Der Wert wurde also abgespeichert. Und zwar wurde der Wert als Variable gespeichert. Eine Variable ist ein Wert, der im Computer unter einem Namen jederzeit erreichbar und veränderbar ist. In unserem Beispiel heißt die Variable »A«, und sie hat nun den Wert 5. Wir können jetzt auch den Wert weiter verändern, indem wir z.B schreiben: A = 10.

8.1 Variablennamen

Der Name einer Variablen kann ein oder mehrere Zeichen lang sein, wobei das erste Zeichen ein Buchstabe sein muß. Die weiteren Zeichen können aus Buchstaben oder Ziffern bestehen (alphanumerisch). Der Name kann beliebig lang sein, allerdings unterscheidet der Computer die Variablen nur anhand der ersten zwei Zeichen des Variablennamens. Aber keine Angst, Sie werden niemals so viele Variablen in einem Programm benötigen, wie Kombinationen im Variablennamen möglich sind. Bedenken Sie also, daß der Computer die Variablen FELD, FELS und FE als ein und dieselbe Variable ansieht, denn die ersten zwei Zeichen sind ja gleich. Wir empfehlen daher, daß Sie Ihre Variablen nur mit maximal zwei Zeichen bezeichnen.

In den Variablennamen dürfen keine BASIC-Befehle »enthalten« sein. Achten Sie auch auf die reservierten Variablennamen. Probieren Sie einmal folgendes aus:

```
APFELSINEN = 55
```

Der Computer antwortet mit:

```
?SYNTAX ERROR
```

Haben Sie bemerkt, wo der Fehler liegt? Im Namen ist der BASIC-Befehl für den Sinus versteckt. Dadurch kann der Computer mit diesem Namen nichts anfangen.

Einige Beispiele für verbotene Variablennamen:

GOLD, INTEGER, REKTOR, TONNE, ORT, STRASSE usw.

Ebenso sind natürlich Namen wie ON, TO oder IF verboten.

8.2 Variablentypen

Wir haben bisher nur die sogenannten numerischen Variablen kennengelernt. Es gibt jedoch insgesamt drei verschiedene Variablentypen:

- Numerische Variablen (auch Fließkomma-Variablen genannt)
- Integer-Variablen (Ganzzahlvariablen)
- String-Variablen

Der Computer unterscheidet die Variablen an ihrem Namen. Die numerischen Variablen haben wir schon kennengelernt. Diese Variablen bekommen einen Namen ohne besondere Kennzeichnung, wie z. B. A oder C6 oder APFEL (nicht APFELSINE!). Die numerischen Variablen können Werte von 10^{-39} bis 10^{+38} beinhalten.

Integervariablen müssen im Anschluß an ihren Namen mit einem Prozentzeichen (%) gekennzeichnet werden, z. B. A% oder C6%. Diese Variablen können nur ganze Zahlen beinhalten, also Zahlen ohne Nachkommaanteil. Die Größe dieser Zahl ist begrenzt von -32768 bis +32767. Dafür hat die Integervariable den Vorteil, daß sie weniger Speicherplatz verbraucht und die Abarbeitung in einem Programm einen Geschwindigkeitsvorteil bringt. Trotzdem werden die Integervariablen nur selten verwendet. Schließlich gibt es noch die Stringvariablen. Diese Variablen sind mit einem Dollarzeichen ($) hinter dem Namen gekennzeichnet, z. B. A$ oder C6$. Die String-

variablen können Buchstaben, Ziffern und andere Zeichen (Grafik- und Steuerzeichen) beinhalten. Sie können in einem Programm gleichzeitig die Variablen A, A% und A$ verwenden. Das heißt, die verschiedenen Variablentypen können gleiche Namen haben.

Beispiel:

```
A = 12.34
A% = 12
A$ = "COMMODORE"
```

8.3 Reservierte Variablen

Wir haben festgestellt, daß die Variablen jeden Namen erhalten können, sofern in diesem Namen kein BASIC-Statement »versteckt« ist. Es gibt aber noch eine weitere Einschränkung: Die reservierten Variablen des Plus/4. Es handelt sich hierbei um:

ST TI TI$ ER ERR$ EL DS und DS$

Sehen wir uns eine dieser Variablen an. Geben Sie bitte ein:

```
PRINT TI$
```

Sie sehen eine sechsstellige Zahl. Es handelt sich hierbei um die im Plus/4 gerade aktuelle Uhrzeit. Ja, in Ihrem Computer ist eine Uhr eingebaut. Sie können diese einstellen:

```
TI$ = "ssmmss"
```

(Stunden, Minuten, Sekunden der momentanen Uhrzeit)

Die Uhr ist jetzt eingestellt, und Sie können jederzeit die Variable TI$ wieder abfragen. Das können Sie natürlich in Programmen machen.

TI ist ebenfalls eine Variable für die Zeitmessung. TI wird jede 60stel Sekunde um 1 erhöht. Die Zahl in TI dividiert durch 60 ergibt eine Zeit in Sekunden. TI$ und TI werden beim Einschalten des Computers oder bei einem Reset auf 0 zurückgesetzt. TI wird beim Neueinstellen von TI$ der aktuellen Uhrzeit angepaßt.

ST ist die sogenannte Status-Variable. Diese Variable wird bei der Ein- und Ausgabe von Daten auf Kassette, Diskettenlaufwerk oder Drucker vom BASIC-Interpreter benutzt. Zum Beispiel wird in dieser Variablen vermerkt, ob bei der Datenübertragung Fehler aufgetreten sind.

ER, ERR$ und EL beschreiben wir im Kapitel 19, »Fehlerbehandlung«, und im Anhang A. DS und DS$ werden im Kapitel über die Diskettenbefehle beschrieben.

8.4 Die Dimensionierung

Nehmen wir einmal an, Sie brauchen in einem Programm mehrere Namen, z. B. »Meier«, »Müller«, »Schulz«, »Schmidt«. Diese Namen könnten in Stringvariablen gefaßt werden:

A$ = "MEIER"

B$ = "MUELLER"

C$ = "SCHULZ"

D$ = "SCHMIDT"

Es geht aber auch anders:

A$(1) = "MEIER"

A$(2) = "MUELLER"

A$(3) = "SCHULZ"

A$(4) = "SCHMIDT"

Die Variablen haben nun den gleichen Namen, sie sind jedoch durchnumeriert. Es handelt sich um eine Variable mit »Index«-Nummer. Das hat entscheidende Vorteile bei der Programmierung. Sehen wir uns einmal das folgende Programm an:

```
10 REM EINGABE VON ADRESSEN
20 PRINT CHR$(147)
30 X=X+1
40 IF X = 11 THEN 170
50 PRINT "GEBEN SIE DIE ";X;". ADRESSE EIN"
60 PRINT "(ENDE MIT ";CHR$(34);"XXX";CHR$(34)
70 PRINT
80 INPUT "NACHNAME";N$(X)
```

```
90 IF N$(X) = "XXX" THEN 190
100 INPUT "VORNAME";V$(X)
110 IF V$(X) = "XXX" THEN 190
120 INPUT "PLZ WOHNORT";W$(X)
130 IF W$(X) = "XXX" THEN 190
140 INPUT "STRASSE";S$(X)
150 IF S$(X) = "XXX" THEN 190
160 GOTO 30
170 REM AUSGABE DER ADRESSEN
180 PRINT "MAXIMAL 10 ADRESSEN!"
190 FOR A = 1 TO X - 1
200 PRINT CHR$(147);A;". ADRESSE: ";V$(A);" ";N$(A)
210 PRINT
220 PRINT SPC(14) S$(A)
230 PRINT SPC(14) W$(A)
240 PRINT "TASTE DRUECKEN!"
250 GETKEY T$
260 NEXT
270 INPUT "DEN AUSDRUCK WIEDERHOLEN (J/N)";JN$
280 IF JN$ = "J" THEN 190
```

Eine kurze Programmbeschreibung:

Die Programmzeilen 10 und 170 enthalten den REMark-Befehl, der nichts bewirkt, das Programmlisting aber durch die Bemerkung nach diesem Befehl übersichtlicher macht.

Zeile 20 löscht den Bildschirm.

In Zeile 30 wird die laufende Nummer (der sogenannte Index) unserer Stringvariablen um 1 erhöht.

In Zeile 40 wird geprüft, ob schon 10 Adressen eingegeben wurden. Dann springt das Programm zur Zeile 170.

Zeile 50 gibt die Nummer der laufenden Adresse aus.

Durch Zeile 60 erscheint der Hinweis, daß die Adresseneingabe durch die Eingabe von "XXX" beendet werden kann.

Nun erfolgt bis zur Zeile 150 die Eingabe der Daten. Nach jeder erfolgten Eingabe wird überprüft, ob es sich dabei um "XXX" handelt und das Programm zur Ausgabe springen soll.

190 und 260 beinhalten den FOR-TO-STEP-NEXT-Befehl (wird im Kapitel 13 »Programmschleifen« noch näher erläutert). Hierbei wird der Wert der Laufvariablen A so lange um 1 erhöht, bis der Wert von X minus 1 erreicht wird. Die Variable X stellt ja

bekanntlich die Anzahl der Adressen dar. Da sie aber vor der Eingabe der ersten Adresse schon um 1 erhöht wurde, muß jetzt, bei der Ausgabe, 1 abgezogen werden, denn die Adresse »XXX« soll ja nicht ausgegeben werden.

Nun wird der gesamte Programmteil zwischen FOR und NEXT, also die Zeilen 200 bis 250, so oft ausgeführt, wie Adressen eingegeben wurden.

In den Zeilen 220 und 230 taucht »SPC(14)« auf, das bedeutet, daß 14 Space (Leerzeichen) ausgegeben werden.

Der Befehl GETKEY in Programmzeile 250 stoppt das Programm ab und wartet auf einen Tastendruck.

In den Zeilen 270 und 280 wird abgefragt, ob der Ausdruck wiederholt werden soll oder nicht. Dabei wird nur auf die Eingabe von »J« geprüft.

Es handelt sich bei dem Programm um ein Beispiel, das zur ernsthaften Adressenverwaltung nicht geeignet ist. Es soll Ihnen nur einmal den Umgang mit Variablen zeigen, die eine Indexnummer bekommen. Wie Sie unserem Beispiel entnehmen können, sind die Variablen also einfach durchnumeriert. Anhand dieser Nummer kann man verschiedene Variablen in einen Zusammenhang bringen, z. B. Nachname 1, Vorname 1, Wohnort 1 und Straße 1. Bei den Variablen mit Indexnummer spricht man auch von eindimensionalen Variablen. Es gibt aber auch noch die mehrdimensionalen Variablen. Sehen Sie sich dazu das Bild 8.1 an.

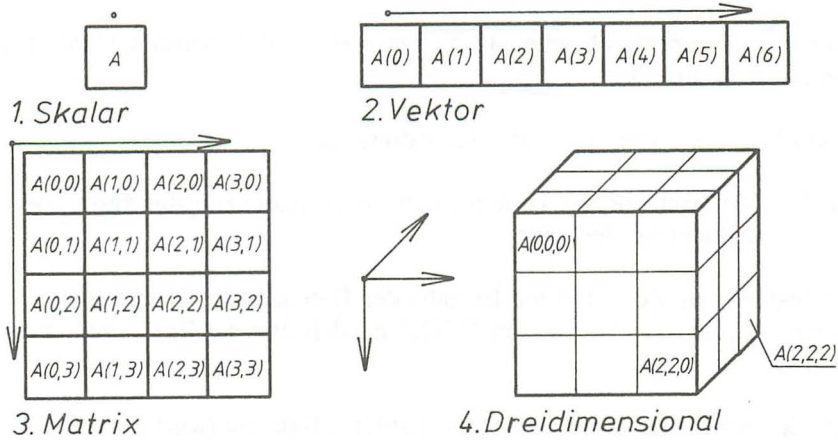


Bild 8.1: *Mehrdimensionale Variablen*

Stellen Sie sich einmal vor, die Variable im ersten Fall sei eine Karteikarte. Im zweiten Fall haben wir mehrere Karteikarten in einem Karteikasten. Im dritten Fall schließlich haben wir es schon mit mehreren Karteikästen zu tun. Bedenken Sie, daß hierbei immer mehr Speicherplatz für die Variablen gebraucht wird. Im vierten Fall wird unsere Variable »dreidimensional«. Der Würfel soll das verdeutlichen. Wir können unsere Gedanken aber noch weiterführen. So gibt es auch vierdimensionale Felder. Leider ist uns keine einfache zeichnerische Darstellung eingefallen. Die Grenzen sind noch nicht erreicht, es können auch Variablen mit noch mehr Indexnummern verwendet werden (Einstein hätte wohl auch nicht gedacht, daß es einmal so viele »Dimensionen« geben würde). Begrenzt wird diese ganze Geschichte durch unseren zur Verfügung stehenden Speicherplatz.

Für die Dimensionierung gibt es den BASIC-Befehl DIM.

Der Befehl DIM

Der Computer muß für die Variablenfelder Speicherplatz reservieren. Bei Indexnummern von 1 bis 10 macht er das automatisch. In unserem Beispielprogramm klappt das auch prima. Wenn wir aber mehr als 10 Elemente brauchen, müssen wir dem Computer sagen, wieviel Speicherplatz er reservieren soll. Wir müssen »dimensionieren«, man spricht auch vom »Deklarieren«.

Wenn unsere Variable A 100 Felder haben soll, schreiben wir:

10 DIM A (100)

DIM muß und kann für jede Variable nur einmal in einem Programm verwendet werden. Mit einem DIM-Befehl können auch mehrere Variablen dimensioniert werden:

10 DIM A$ (25,4) , B (50) , C% (150)

Diese Dimensionierung kann nur durch den Befehl CLR wieder rückgängig gemacht werden. Dabei werden aber alle Variablen gelöscht. Anschließend könnte dann tatsächlich neu dimensioniert werden.

Der Index hinter einem DIM-Befehl kann auch durch eine Variable dargestellt werden. Es könnte z. B. in einem Programm abgefragt werden, wie viele Adressen man bearbeiten möchte. Dabei muß man allerdings aufpassen, der Computer gibt sofort eine Fehlermeldung aus, wenn der Speicherplatz nicht ausreichen sollte. Lesen Sie dazu bitte auch das Kapitel über die Fehlerbehandlung.

```
10 INPUT "GEBEN SIE DIE ANZAHL EIN";A
20 DIM A$ (A)
```

Wenn für eine Variable der DIM-Befehl zum zweiten Mal ohne zwischenzeitliches CLR erfolgt, dann wird ein

REDIM'D ARRAY ERROR

ausgedruckt. Wenn die gewünschte Arraygröße nicht in den zur Verfügung stehenden Speicherplatz paßt, gibt der Computer die Meldung:

OUT OF MEMORY ERROR

8.5 Beispielprogramm GALGENSPIEL

Das folgende Programm ist ein Spiel für zwei Personen. Es handelt sich dabei um ein Ratespiel, bei dem der erste Spieler ein Wort in den Computer eintippt und der zweite Spieler dieses Wort erraten muß. Das, vom ersten Spieler natürlich verdeckt eingegebene, Wort wird in einer Variablen gespeichert. Auf dem Bildschirm erscheinen dann Bindestriche in der Anzahl, wie das Wort Buchstaben hat. Der zweite Spieler hat neun Versuche, das Wort zu erraten. Dabei kann er entweder einen einzelnen Buchstaben eingeben oder aber das ganze Wort. Wenn der Buchstabe im gesuchten Wort enthalten ist, wird er auf dem Bildschirm angezeigt. Ist er dagegen nicht im Wort gegeben, oder ist das vom zweiten Spieler eingegebene Wort falsch, wird nach und nach ein Galgen auf dem Bildschirm gezeichnet.

Wir wünschen Ihnen nun viel Spaß beim »Galgenspiel«.

```
10 REM GALGENSPIEL
20 GRAPHIC0,1:PRINTTAB(10);"****GALGENSPIEL****"
30 CHAR,0,10,""
40 PRINT"SPIELER 1: GEBEN SIE BITTE DAS WORT EIN"
50 PRINT
60 POKE239,0:INPUTW$:WA$=W$
70 PRINTCHR$(147)
80 COLOR0,1:COLOR1,2:GRAPHIC1,1
90 CHAR,0,19,"SPIELER 2: BITTE EINEN BUCHSTABEN ODER"
100 CHAR,11,20,"DAS WORT EINGEBEN."
110 FORI=1TOLEN(W$)
120 CHAR,I,23,"-"
130 NEXTI
140 X=0:LW$="":R=0:FORI=0TO39:CHAR,I,24," ":NEXT
150 POKE239,0:GETKEYL$
160 IFL$=CHR$(20)ANDX=0THEN150
170 IFL$=CHR$(13)ANDX=0THEN150
180 IFL$=CHR$(20)THENX=X-1:LW$=LEFT$(LW$,X):CHAR,1+X,24," ":
GOTO 230
190 IFL$=CHR$(13)THEN250
200 IFL$<"A"ORL$>"Z"THEN150
```

```
210 LW$=LEFT$(LW$,X)+L$
220 X=X+1
230 CHAR,1,24,LW$
240 GOTO150
250 IFX>1ANDLW$=W$THEN900
260 IFX>1THEN320
270 FORI=1TOLEN(W$)
280 IFMID$(WA$,I,1)=LW$THENCHAR,I,23,LW$:R=1:RG=RG+1:MID$
(WA$,I,1)="1"
290 NEXTI
300 IFRG=LEN(W$)THEN900
310 IFR=1THEN140
320 F=F+1
330 ONFGOSUB400,450,500,550,600,650,700,750,800
340 IFF<9THEN140
350 CHAR,0,21,"ALLES FALSCH!!!! DAS WORT LAUTET:"
360 CHAR,1,23,W$,1
370 POKE239,0
380 GETKEYA$:RUN
400 CIRCLE,80,140,40,20,275,85
410 DRAW,40,138TO120,138
420 PAINT,45,135
430 RETURN
450 FORI=0TO2:DRAW,79+I,120TO79+I,5:NEXT
460 RETURN
500 FORI=0TO2:DRAW,80,5+ITO140,5+I:NEXT
510 RETURN
550 FORI=0TO2:DRAW,80,40-ITO95-I,5:NEXT
560 RETURN
600 DRAW,130,5TO130,15
610 RETURN
650 CIRCLE,135,20,8,5,,,320
660 RETURN
700 CIRCLE,130,55,8,27
710 RETURN
750 CIRCLE,115,40,15,2,,,320
760 CIRCLE,145,40,15,2,,,40
770 RETURN
800 CIRCLE,120,99,20,3,,,285
810 CIRCLE,140,99,20,3,,,75
820 DRAW,133,22TO135,24
830 DRAW,135,18
840 DRAW,138,21
850 RETURN
900 CHAR,1,23,W$
910 FORI=0TO39:CHAR,I,19," ":CHAR,I,20," ":NEXT
920 CHAR,0,24,"RICHTIG GERATEN!",1
930 GETKEYA$:RUN
```

9 Die Programmerstellung auf dem Plus/4

9.1 Der Programmodus

Wenn sich im Rechnerspeicher ein Programm befindet und Sie es mit dem BASIC-Befehl RUN starten, wird der Rechner in den Programmodus versetzt, d.h. die Kontrolle über den Rechner wird an das Programm übertragen. Der Benutzer des Programms wird, vielleicht noch einige Male aufgefordert, Daten über die Tastatur einzugeben, ansonsten hat er nun Pause.

Unterbrechen kann man ein Programm durch Drücken der STOP-Taste. Sollte der Rechner auf für ihn unlogische Anweisungen treffen, wie zum Beispiel eine Division durch 0 oder ein falsch geschriebenes Befehlswort, dann bricht er mit einer Fehlermeldung das Programm selbsttätig ab. Die Kontrolle geht wieder auf den Anwender über.

Das gleiche, allerdings ohne Fehlermeldung, passiert, wenn man in einer Zeile des Programms ein END eingegeben hat, auf das der Rechner bei der Abarbeitung trifft, oder er keine weitere Programmzeile findet. In diesem Falle meldet er sich mit 'READY' zurück.

9.2 Die Startbefehle RUN und GOTO

Sie können ein Programm durch zwei Befehle starten. Fangen wir mit dem gebräuchlichsten an.

Syntax:

RUN

oder

RUN Zeilennummer

Wenn Sie RUN eingeben, dann löscht der Rechner erst einmal alle Variablen. Dann wird die erste Zeilennummer des Programms festgestellt und bei ihr mit der Abarbeitung des Programms begonnen. Wird RUN von einer Zeilennummer gefolgt, so beginnt bei dieser Zeile die Abarbeitung. Die Zeilennummer muß wirklich existent sein, da der Rechner sonst mit einer Fehlermeldung abbricht.

Ein Programm kann auch noch mit einem anderen Befehl gestartet werden.

Syntax:

GOTO Zeilennummer

Dieser Befehl verhält sich ähnlich RUN Zeilennummer, nur werden die Variablen nicht gelöscht. Die Angabe einer Zeilennummer ist unbedingt notwendig. Dieser Befehl eignet sich gut zum Austesten von Programmteilen, nachdem der Rechner den Programmablauf mit einer Fehlermeldung abgebrochen hat.

9.2.1 Das Anhalten eines Programms

Bis auf wenige Ausnahmen können Sie ein Programm jederzeit durch Drücken der RUN/STOP-Taste anhalten. Es erscheint nun die Meldung: »BREAK IN Zeilennummer«. Sie können sich jetzt z. B. wichtige Variablen ausgeben lassen oder deren Inhalt ändern. Geben Sie danach CONT ein, so fährt das Programm an der Stelle fort, an der es unterbrochen wurde.

Eine zweite Möglichkeit bietet CTRL/S. Werden diese beiden Tasten gleichzeitig gedrückt, so wird der Programmablauf nicht unterbrochen, sondern lediglich angehalten. Es erscheint keinerlei Meldung auf dem Bildschirm. Der Rechner macht erst weiter, wenn eine zeichenausgebende Taste, also nicht CTRL, SHIFT usw., gedrückt

wird. Möchten Sie, daß Ihr Programm nicht mit CTRL/S angehalten werden kann, dann müssen Sie als erste Programmzeile

POKE 2039,6

eingeben. Einschalten läßt sich diese Tastenkombination wieder mit:

POKE 2039,0

Wir kommen nun zu einem Befehl, der, in Programmen eingesetzt, ebenfalls den Programmablauf unterbricht, dem STOP-Befehl

Syntax:

STOP

Diesen Befehl können Sie in jedem Programm einsetzen. Er dient in erster Linie der Fehlereingrenzung und bewirkt dasselbe wie das Drücken der RUN/STOP-Taste. Sie können sich ebenfalls Variableninhalte anzeigen lassen oder diese ändern. Die Fortführung des Programms erreicht man ebenfalls mit dem CONT-Befehl. Das Programm macht dann mit der Abarbeitung des Befehls, der dem STOP-Befehl folgt, weiter.

ACHTUNG: Sie können mit CONT kein Programm fortsetzen, wenn es in irgendeiner Weise verändert wurde. Es erfolgt sonst die Fehlermeldung »CAN'T CONTINIUE ERROR«.

9.2.2 Das Beenden von Programmen

Ein Programm wird vom Rechner automatisch in der letzten Programmzeile beendet, wenn diese keinen Sprungbefehl zurück in das Programm enthält. Ihr Plus/4 meldet sich dann mit »READY« zurück und harrt der Dinge, die da kommen sollen. Es gibt aber auch einen Befehl, mit dem ein Programm beendet werden kann: END.

Syntax:

END

Dieser Befehl wird immer an der (den) Stelle(n) eingesetzt, an der (denen) ein Programm beendet werden soll. Er darf sich an jeder Stelle eines Programms befinden. Oft sind in einem Programm Unterprogramme, die fallweise abgearbeitet werden. Soll ein Programm in der Zeile vor der Unterroutine beendet werden, so muß ein END in ihr vorhanden sein. Da eine Unterroutine immer mit RETURN abgeschlossen wird,

würde der Rechner mit der Fehlermeldung »?RETURN WITHOUT GOSUB ERROR IN Zeilennummer« unterbrechen.

Ein END dient auch der Dokumentation eines Programms, da man jederzeit in einem Programmlisting sehen kann, an welcher Stelle und unter welchen Bedingungen ein Programm beendet wird. Sollten diesem Befehl noch weitere Programmzeilen folgen, läßt es sich mit CONT wieder starten.

9.3 Hilfen beim Programmieren

Damit der Rechner unterscheiden kann, ob er die Befehlszeile sofort abarbeiten soll oder ob es sich um die Eingabe einer Programmzeile handelt, wird der Programmzeile eine Zahl vorangestellt. Diese Zahl darf sich in einem Bereich von 0 bis 63999 bewegen. Nach der Eingabe einer solchen Zeile und Druck der RETURN-Taste wird die Zeile im Programmspeicher abgelegt. Beim Plus/4 wird die eingegebene Zeile nicht auf eventuell vorhandene Fehler überprüft, das geschieht erst während des Programmablaufes.

Der Plus/4 verfügt über einen sogenannten bildschirmorientierten Editor, was bedeutet, daß Sie den Cursor auch beim Eingeben eines Programms frei auf dem Bildschirm bewegen können. Diese Tatsache macht den Plus/4 besonders bedienerfreundlich, da auch nachträglich sehr schnell eine vorhandene Zeile verändert werden kann. Danach kann man an alter Stelle mit der Programmierung fortfahren.

In einer Programmzeile dürfen auch mehrere Anweisungen stehen. Diese werden durch einen »:« (Doppelpunkt) getrennt. Die Länge einer Programmzeile darf normalerweise max. 80 Zeichen betragen, dies entspricht zwei Bildschirmzeilen. Die Ausnahmen erfahren Sie auch in diesem Kapitel. Der Plus/4 stellt Ihnen viele komfortable Programmierhilfen zur Verfügung.

9.3.1 Der LIST-Befehl

Syntax:

LIST	oder
LIST Zeilennummer	oder
LIST Zeilennummer -	oder
LIST - Zeilennummer	oder
LIST Zeilennummer - Zeilennummer	

Beim Programmieren werden Sie diesen Befehl sicher am häufigsten brauchen. Mit ihm können Sie sich jederzeit das gesamte Programm ansehen. Nun einige Erklärungen zur Syntax:

LIST	Es wird das gesamte Programm aufgelistet.
LIST Zeilennummer	Zeigt die Programmzeile, die in Zeilennummer angegeben wurde.
LIST Zeilennummer -	Alle Zeilen ab Zeilennummer werden gelistet.
LIST - Zeilennummer	Alle Zeilen bis einschließlich Zeilennummer werden gelistet.
LIST Zeilennummer - Zeilennummer	Listet die Zeilen von bis.

Das Listen eines Programmes können Sie jederzeit mit der RUN/STOP-Taste abbrechen. Soll das Listen verlangsamt werden, dann drücken Sie die COMMODORE-Taste. CTRL/S schließlich hält das Listen an. Fortsetzen läßt sich das Listen mit der Betätigung einer zeichenausgebenden Taste.

9.3.2 Der AUTO-Befehl

Syntax:

AUTO Zeilenabstand

Mit diesem Befehl wird Ihnen die Eingabe der Zeilennummer durch den Rechner abgenommen. Wenn Sie zum Beispiel AUTO 100 eingeben, wird der Zeilenabstand auf 100 festgelegt. Der Abstand kann jederzeit durch einen anderen ersetzt werden. Abgeschaltet wird AUTO durch die Eingabe:

AUTO

RUN oder »GOTO Zeilennummer« schaltet diesen Modus ebenfalls ab.

Geben Sie bitte folgendes ein:

AUTO 10

Es erscheint »READY« auf dem Bildschirm. Nun geht es weiter mit

10 PRINT "DER AUTO-BEFEHL"

Am linken Rand des Bildschirms steht jetzt die Zahl 20. Wenn Sie noch einmal die RETURN-Taste drücken, erscheint der Cursor eine Zeile tiefer auf dem Bildschirm. Hätte sich zu diesem Zeitpunkt die Programmzeile 20 bereits im Speicher befunden, dann wäre sie gelöscht worden. Beim Speichern von Programmzeilen überprüft der Rechner, ob sich im Speicher bereits eine Zeile mit derselben Nummer befindet. Sollte dies der Fall sein, wird die vorhandene durch die neue Zeile ersetzt.

Obwohl keine neue Zeilennummer auf dem Bildschirm ausgegeben wurde, ist der AUTO-Befehl immer noch wirksam. Machen wir weiter mit:

5 PRINT "DAS IST"

Nach RETURN erscheint die Zahl 15. Das Abschalten von AUTO erfolgt durch RETURN, AUTO, RETURN.

Geben wir LIST ein, so erscheint nach RETURN unser Programm auf dem Schirm, und zwar in der richtigen Reihenfolge der Zeilennummern. Starten Sie jetzt das Programm mit RUN.

Es kommt immer wieder vor, daß man in ein Programm Zeilen einfügen möchte. Sollte es sich um mehrere Zeilen handeln, und hat man die Zeilenabstände zu klein gewählt, so stehen einem plötzlich keine Zeilennummern in der gewünschten Höhe zur Verfügung. Auch hier wurde Vorsorge getroffen.

9.3.3 Der RENUMBER-Befehl

Syntax:

RENUMBER neue Startzeile,Zeilenabstand,alte Startzeile

Dieser Befehl bewirkt ein neues Durchnumerieren der Programmzeilen. Werden die angegebenen Parameter nicht eingegeben, so erfolgt die Neunumerierung der Zeilen von der ersten bis zur letzten Zeile im 10er-Abstand, und die erste Zeile erhält die Nummer 10.

Zu den Parametern:

neue Startzeile	Dies ist nach RENUMBER die erste Programmzeile
Zeilenabstand	Ist der Abstand, den die Zeilen zueinander haben sollen
alte Startzeile	Bei dieser Zeile beginnt die Neunumerierung

Die Neuordnung der Zeilen beginnt mit dem Parameter, der in »neue Startzeile« übergeben wurde und wird erst am Programmende abgeschlossen. Wird der Parameter »neue Startzeile« nicht mit angegeben, so wird das gesamte Programm neu durchnumeriert. Die neue Startzeile darf nicht kleiner gewählt werden als eine Zeilennummer, welche nicht in das RENUMBER mit einbezogen wird.

ACHTUNG: RENUMBER arbeitet nicht ganz fehlerfrei. Beim Programmieren dürfen keine Zeilennummern größer als 63999 vergeben werden. Der RENUMBER-Befehl ignoriert diese gänzlich. Zeilennummern über 63999 und kleiner 65536 werden ebenfalls vergeben. Allerdings kann man diese weder editieren (berichtigen) noch löschen, auch ein erneutes RENUMBER bringt keinen Erfolg.

Zeilennummern über 65535 werden ebenfalls angezeigt, und zwar beginnt die Zählung wieder bei 0. Nach einem LIST stehen diese Zeilen in der richtigen Reihenfolge am Ende des Programms. Sie werden auch nach einem RUN richtig abgearbeitet. Bei diesen Zeilennummern gilt ebenfalls das oben Gesagte.

Dieser Umstand hat auch Vorteile, so kann man z. B. sein Copyright durch ein RENUMBER in diesen Bereich schieben, ohne daß jemand es ohne weiteres löschen kann. Normalerweise sollte man aber darauf achten, daß die Zeilennummer 63999 von einem RENUMBER nicht überschritten wird.

Sollte sich das oben angeführte Programm noch im Speicher befinden, dann versuchen Sie einmal:

RENUMBER 100,20

Nach LIST sehen Sie nun Ihr Programm, beginnend mit der Zeile 100, die zweite Zeile hat die Nummer 120 erhalten.

9.3.4 Das Abkürzen von Befehlen

Die meisten BASIC-Befehle lassen sich abkürzen. Sie können zum Beispiel statt PRINT auch das Fragezeichen (?) eingeben. Durch diese Abkürzungen läßt sich beim Programmieren eine Menge Arbeit sparen. Die abgekürzten Befehle werden bei einem LIST wieder als Klartext ausgegeben. Das Abkürzen kann sowohl im Großschrift-Grafik- als auch im Groß-/Kleinschriftmodus angewendet werden. Wir empfehlen Ihnen das Programmieren im zuletzt genannten Modus, da sonst bei der Abkürzung der meisten Befehle Grafikzeichen angezeigt werden und die Eingabe sehr schnell unübersichtlich wird.

Durch Abkürzung von Befehlen läßt sich aber auch Platz in einer Programmzeile sparen. Eigentlich darf eine Programmzeile nicht länger als 80 Zeichen sein, aber: Geben Sie doch einmal 10 und dann so lange ?: ein, bis zwei Bildschirmzeilen vollgeschrieben sind. Nun folgt LIST. Sie sehen, mehrere Bildschirmzeilen sind mit »PRINT:« komplett beschrieben. Auch wenn Sie mit RUN starten, werden diese PRINT-Befehle klaglos abgearbeitet. Der Plus/4 beachtet also nur die Länge der Eingabezeile und nicht die Länge, die die Befehle später benötigen. Dies liegt am Bildschirmeditor und hat daher auch einen Schönheitsfehler. Bewegen Sie nämlich den Cursor in diese aufgelistete Zeile und drücken dann die RETURN-Taste, erfolgt die Fehlermeldung »STRING TOO LONG ERROR«.

Eine Tabelle der möglichen Abkürzungen finden Sie im Anhang C dieses Buches.

9.4 Verbessern von Programmen

Über den bildschirmorientierten Editor lassen sich sehr schnell Änderungen und Berichtigungen eines Programms realisieren. Sie können, ohne eine Programmzeile neu einzugeben, Zeichen in einer Programmzeile verändern, löschen oder hinzufügen.

Gehen Sie dazu einfach mit dem Cursor auf das betreffende Zeichen und führen Sie dann den oder die entsprechenden Tastendrücke durch. Die Berichtigung der Programmzeile muß immer mit RETURN abgeschlossen werden. Was Sie tun müssen, um zum Beispiel Zeichen einzufügen oder zu löschen, das erfahren Sie im Kapitel über die Tastatur.

Beim Programmieren kommt es oft vor, daß ganze Programmzeilen oder Programmteile gelöscht werden sollen, weil man zum Beispiel eine bessere Lösung gefunden hat. Gehen Sie wie folgt vor, um eine Programmzeile oder nicht zusammenhängende Programmzeilen zu löschen:

Geben Sie die Zeilennummer, die gelöscht werden soll, in einer Leerzeile des Bildschirms ein. Drücken Sie die RETURN-Taste. Die Zeile ist nun gelöscht.

ACHTUNG: Sollten Sie versehentlich eine Zeile gelöscht haben, die sich aber noch auf dem Bildschirm befindet, so können Sie diese regenerieren, indem Sie den Cursor auf oder in die Zeile setzen und die Taste RETURN drücken. Die Zeile wird wieder an der richtigen Stelle des Programmspeichers plaziert. Auf dem gleichen Wege lassen sich auch Zeilen duplizieren. Gehen Sie dafür mit dem Cursor auf eine zu verdoppelnde Zeile, geben Sie dann die neue Zeilennummer ein und drücken Sie die RETURN-Taste. Nach LIST erscheint nun diese Programmzeile an der richtigen Stelle des Programms.

Oft ist es wünschenswert, ganze Programmbereiche zu löschen. Das BASIC des Plus/4 läßt uns auch hier nicht im Stich.

9.4.1 Der DELETE-Befehl

Syntax:

DELETE Anfangszeile - Endzeile

Mit dem Befehl können Sie bestimmte Programmbereiche löschen. Auch bei diesem Befehl müssen nicht alle Parameter übergeben werden. So entfernt ein DELETE 200 lediglich die Zeile 200 aus dem Speicher, ein DELETE 200-300 aber die Zeilen 200 bis 300. Geben Sie DELETE -200 ein, werden alle Zeilen bis einschließlich der Zeile 200 gelöscht. DELETE 200- schließlich löscht alle Zeilen beginnend mit der Nummer

200 bis zum Programmende. Alle oben genannten und beschriebenen Befehle können nur im Direktmodus angewandt werden.

Und nun stellen wir einen Befehl vor, der den Speicher vollkommen löscht.

9.4.2 Der NEW-Befehl

Syntax:

NEW

Mit diesem Befehl sollte man sehr vorsichtig umgehen, da er ein im Speicher befindliches Programm entfernt. Dieser Befehl funktioniert sowohl im Direkt- als auch im Programmodus. Es ist denkbar, daß ein Programm ein Paßwort erwartet und sich nach einer Falscheingabe selber durch NEW löscht. Übrigens: Im Kapitel 20 zeigen wir Ihnen, wie Sie ein mit NEW gelöschtes Programm zurückholen können.

9.5 Programmiervorbereitungen

Jedes Programm kann nur so gut wie sein Programmierer sein. Wenn man zehn Programmierer bittet, für das gleiche Problem ein Programm zu erstellen, so wird man aller Wahrscheinlichkeit nach zehn verschiedene Programme erhalten. Alle Programme werden das Problem sicher lösen, aber eins dieser Programme ist dann bestimmt das übersichtlichste, ein anderes vielleicht das schnellste. In diesem Buch kommt es uns insbesondere darauf an, die Vielzahl der Möglichkeiten, die Ihnen der Rechner bietet, zu erlernen. Außerdem möchten wir Ihnen natürlich Hinweise auf richtiges und übersichtliches Programmieren geben.

Am Anfang des Programmierens sollte immer die genaue Planung stehen. Sie sollten nicht nur wissen, was das Programm machen soll, sondern vor allem, wie es das machen soll. Sie müssen sich also Zeit für die Problemanalyse nehmen. Schreiben Sie sich dabei Stichworte auf, sie werden die Programmierung einfacher gestalten. Fertigen Sie eine Übersicht über den Programmablauf an, und geizen Sie dabei nicht mit Erklärungen. Verwenden Sie so wenig Sprungbefehle wie möglich, sie machen ein Programm unübersichtlich. Bedenken Sie immer, daß eine sorgfältige Planung mindestens das halbe Programm ist. Die Zeit, die Sie vorher in die Planung investiert haben, wird Ihnen beim Programmieren, spätestens aber bei der Fehlersuche, zurückvergütet. Als Lohn winkt Ihnen dann auch ein Programm, welches sich sehr schnell und mit wenig Programmieraufwand anderen Gegebenheiten anpassen läßt und das man auch noch nach Monaten versteht.

Für die übersichtliche Gestaltung stellen wir Ihnen noch einen Befehl vor.

9.5.1 Der REM-Befehl

Syntax:

REM

REM ist eine Abkürzung und kommt vom englischen Remark (auf deutsch Bemerkung).

Eingesetzt wird er, wenn ein Programmteil im Programm selbst beschrieben werden soll. REM sorgt dafür, daß das, was diesem Befehl folgt, vom Rechner überlesen wird. Durch diesen Befehl bleiben Programme übersichtlicher und werden verständlicher. Gewöhnen Sie es sich daher insbesondere bei längeren Programmen und Routinen an, diese durch ein vorgestelltes REM zu kommentieren. Es kann Ihnen sonst passieren, daß Sie Ihr eigenes Programm nach ein paar Wochen selbst kaum noch durchschauen. Bei längeren Programmen sollten die ersten Zeilen mit REM beginnen, um dahinter die im Programm vorkommenden Variablen zu dokumentieren. Das könnte wie folgt aussehen:

```
10 REM NN=NACHNAME VN=VORNAME SA=STADT
20 REM II=LAUFVARIABLE UND KANN VON  JEDEM PROGRAMMTEIL
VERWENDET WERDEN.
.
.
.
100 REM HAUPTPROGRAMM
.
.
.
300 REM ZEICHENEINGABE; ERLAUBT A,S,E
.
.
500 REM USW.
```

Wenn man sich bemüht, nur eine Anweisung oder max. zwei kurze Anweisungen in einer Programmzeile unterzubringen, dann trägt dies ebenfalls sehr zur Übersichtlichkeit bei. Zugegeben, dies ist der Geschwindigkeit sicher nicht sehr zuträglich, aber bei sehr zeitkritischen Problemen kann man oft durch den Einsatz besserer Befehle und Anweisungen, die einem das Plus/4-BASIC bietet, vieles an Zeit wieder aufholen. Es ist auch auf jeden Fall einfacher, umständliche Lösungen zu erkennen.

WICHTIGER HINWEIS: Speichern Sie auf jeden Fall Ihr Programm zwischenzeitlich ab, ein Stromausfall kann die Arbeit von Stunden zunichte machen. Das Programm sollte auf jeden Fall vor dem ersten Start abgespeichert werden, insbesondere wenn POKEs vorhanden sind. Ist der Rechner erst einmal abgestürzt, dann hilft nämlich meist nur der Druck auf die RESET-Taste. Das Programm ist dann aber völlig zerstört.

9.6 Strukturiertes Programmieren

Sicher werden Sie, wenn Sie das Buch bis hierher aufmerksam gelesen haben, jetzt auch eigene Programme entwerfen können. Es ist ja auch alles so einfach: Einige Programm-Zeilennummern und einige BASIC-Befehle dazu, und schon ist das Programm fertig. Nun, leider ist das doch nicht so einfach. Sicher, die Programme, die man Zeile für Zeile schreibt, laufen eventuell fehlerfrei und erfüllen ihre Aufgabe, so daß Sie mit Ihrem Programm zufrieden sind. Was ist nun aber, wenn Ihr Programm etwas umfangreicher ist und Sie es nach einiger Zeit noch einmal ändern möchten? Denken Sie z. B. einmal an ein Programm, das Ihnen Ihre zu zahlende Lohnsteuer für 1986 berechnet. Das Programm läuft, alles klappt prima. Natürlich speichern Sie das Programm ab, damit Sie auch 1987 wieder die Berechnung machen können, oder vielleicht sollen für Bekannte und Verwandte solche Berechnungen gemacht werden. Doch was ist, wenn 1987 neue Steuergesetze in Kraft treten, und die Berechnung daher geändert werden muß? Sie müssen also Ihr Programm entweder neu schreiben, oder die Änderungen am schon bestehenden Programm durchführen. Natürlich werden Sie kein neues Programm schreiben, sondern das alte ändern. Dazu muß aber eine sehr wichtige Bedingung erfüllt sein: Ihr Programm muß so geschrieben sein, daß Sie auch die Änderungen durchführen können, es muß übersichtlich sein, und Sie müssen auch später noch erkennen, was wann wo im Programm passiert.

Sie haben es sicher schon erkannt, es müssen bestimmte Programmierregeln eingehalten werden. Das Programm muß eine klare Ausdrucksweise haben, und es muß in kurze, übersichtliche Blöcke aufgeteilt sein. Kurz, das Programm muß strukturiert sein. Stellen Sie sich vor, Sie geben Ihr Programm an Freunde oder Bekannte weiter, die es an ihre eigenen Bedürfnisse anpassen wollen. Wenn Ihr Programm im »Spaghetti-Code« (ein großes Durcheinander) geschrieben ist, werden Sie selbst eventuell schon während des Programmierens nicht mehr »durchblicken«.

Das Programm muß folgende Bedingungen erfüllen:

1. Natürlich muß es funktionieren.
2. Es muß übersichtlich sein.
3. Es muß komfortabel (bedienungsfreundlich) sein.
4. Es muß änderbar sein.
5. Es muß wirksam, also effektiv sein.

Unter Umständen wird man Kompromisse machen müssen, denn es kann ja sein, daß Ihr zur Verfügung stehender Speicherbereich nicht ausreicht. (Beim Plus/4 müßte Ihr Programm allerdings schon sehr umfangreich sein, wenn Sie den ganzen Speicher damit belegen.)

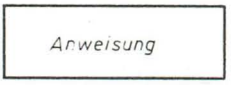
Wenn Sie die nun folgenden Regeln einhalten, werden Sie vermutlich viel Zeit zum Überlegen brauchen. Dafür ist die eigentliche »Codierung«, also das Umschreiben Ihrer Gedanken in ein lauffähiges Programm wesentlich kürzer.

Anmerkung: Bei sehr kurzen Programmen, die Sie nur einmal und auch nur für eine spezielle Aufgabe benötigen, ist eine strukturierte Programmierung nicht notwendig. Nur, wenn Programme umfangreicher werden und für längere Zeit gebraucht werden und die Programme universell sein sollen, ist eine klare Programmierung angebracht, ja sogar gefordert. Und dazu noch eine Anmerkung: Wir, die Autoren, geben zu, daß auch wir immer wieder den Fehler machen, einfach drauflos zu programmieren. Als wir anfingen, BASIC und Assembler zu erlernen, sprach niemand von »strukturiert«, man ist vielmehr selbst darauf gekommen, als Programme wieder gelöscht werden mußten, weil keiner mehr durchblickte. Nobody is perfect.

Als Hilfsmittel für die Programmierung dienen die sogenannten Programmablaufpläne oder Flußdiagramme. Dabei wird ein Problem grafisch dargestellt. Und zwar wird das Problem in kleine Schritte zerteilt, die in der richtigen zeitlichen Reihenfolge dargestellt werden. Für die Programmablaufpläne gibt es Sinnbilder, die wir hier kurz zeigen.

9.6.1 Sinnbilder für Programmablaufpläne

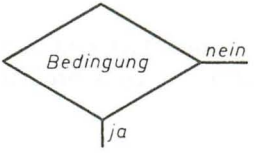
Bei uns ist bekanntlich fast alles genormt. Daher gibt es auch für die Sinnbilder eine Norm, und zwar die DIN 66001. Die wichtigsten Sinnbilder sind folgende:



Operation, allgemein

Z. B. Zuweisungen, Deklarationen, auch mehrere Anweisungen möglich.

Bild 9.1: *Operation, allgemein*

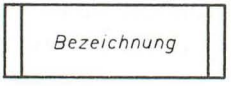


Verzweigung

Die Bedingung für die Verzweigung sollte im Symbol eingetragen werden.

Bild 9.2: *Verzweigung*

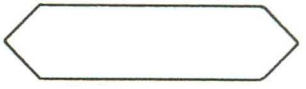
Unterprogramm



Die Bezeichnung für das Unterprogramm wird eingetragen, zusätzlich ist es empfehlenswert, auch die Zeilennummer einzutragen.

Bild 9.3: *Unterprogramm*

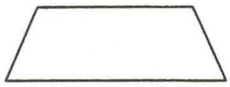
Programm-Modifikation



Grundlegende Änderungen, die das nachfolgende Programm betreffen, z.B. Umschalten des Speicherbereichs.

Bild 9.4: *Programm-Modifikation*

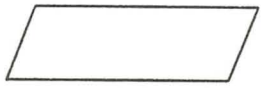
Operation von Hand



Hierzu gehören Dinge wie das Einschalten von Geräten oder das Einspannen von Papier in den Drucker.

Bild 9.5: *Operation von Hand*

Eingabe, Ausgabe



Die Ausgabe von Daten auf den Bildschirm oder andere Ausgabegeräte oder die Eingabe von Daten von der Tastatur oder anderen Eingabegeräten.

Bild 9.6: *Eingabe, Ausgabe*

Ablauflinie

Der zeitliche Ablauf im Programmablaufplan erfolgt von oben nach unten bzw. von links nach rechts. Die Symbole sind durch Ablauflinien miteinander verbunden.

Bild 9.7: *Ablauflinie*

Zusammenführung

Eine Programmstelle, die von mehreren anderen Programmteilen erreicht werden kann. Z. B. GOTO.

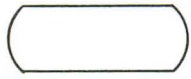
Bild 9.8: *Zusammenführung*

Übergangsstelle

Ein Programmablaufplan kann an einer Stelle unterbrochen werden, um an anderer Stelle weitergeführt zu werden. Die Übergangsstellen werden mit Buchstaben gekennzeichnet. Zusammengehörige Übergangsstellen bekommen gleiche Bezeichnungen. Eine Übergangsstelle kann von mehreren anderen erreicht werden.

Bild 9.9: *Übergangsstelle*

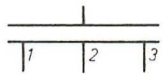
Grenzstelle



Programmstart oder Programmende.

Bild 9.10: *Grenzstelle*

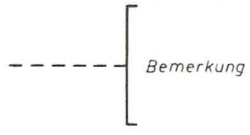
Aufspaltung



Ein Ersatz für mehrere Verzweigungen.
Beispiel: ON...GOSUB...

Bild 9.11: *Aufspaltung*

Bemerkungen



Sparen Sie nicht mit Bemerkungen zum Programm. Die gestrichelte Linie muß zu dem Symbol führen, zu dem die Bemerkung gehört. Es können auch Befehle eingetragen werden.

Bild 9.12: *Bemerkungen*

Neben diesem Programmablaufplan setzt sich immer mehr das sogenannte Struktogramm durch. Struktogramme wurden erforderlich durch die Entwicklung »höherer« Programmiersprachen wie Pascal, Comal, Fortran 77 , Elan usw., die eine Strukturierung zulassen. BASIC ist eigentlich keine strukturierte Programmiersprache, nun bietet aber der Plus/4 Befehle, die eine Strukturierung zulassen.

9.6.2 Sinnbilder im Struktogramm

Im Gegensatz zu Programmablaufplänen ist in Struktogrammen kein GOTO-Befehl darstellbar.

Struktogramme werden auch als Nassi-Shneiderman-Diagramme oder kurz als NS-Diagramme bezeichnet. Wir zeigen Ihnen nun einige wichtige Strukturen der NS-Diagramme im Vergleich zu Programmablaufplänen.

Nassi-Shneiderman **Programmablaufplan**

Sequenz

Mehrere Operationen.

Bild 9.13: *Mehrere Operationen*

Verzweigung

IF...THEN... oder
IF...THEN...ELSE

Bild 9.14: *Verzweigung*

Mehrfache Verzweigung

Z. B. ON..GOSUB..

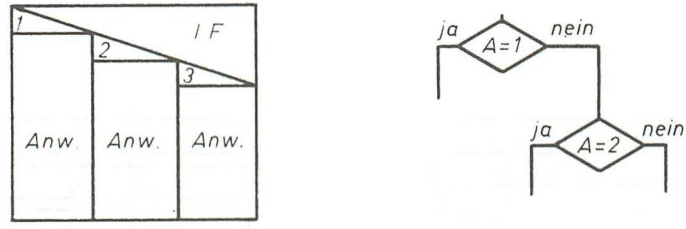


Bild 9.15: *Mehrfache Verzweigung*

Wiederholung

Ein Programm wird so lange wiederholt, bis eine Bedingung erfüllt ist.

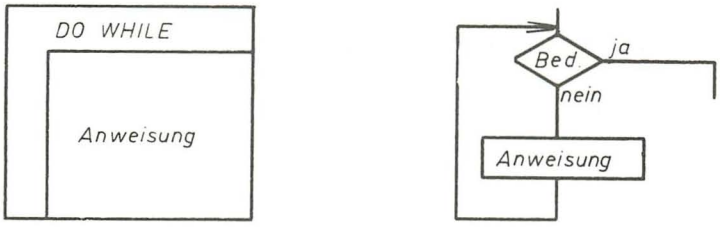


Bild 9.16: *Wiederholung*

Unterprogramm

Bild 9.17: *Unterprogramm*

9.6.3 Beispielprogramm KALENDER

Wir zeigen Ihnen nun die Entwicklung eines Programms. Wir werden das Programm mittels Programmablaufplan und Struktogramm grafisch darstellen.

Problemstellung: Ein Programm soll geschrieben werden, das für ein beliebiges Datum den entsprechenden Wochentag ausgibt. Wir richten uns dabei nach den Regeln des Gregorianischen Kalenders: Die durch vier teilbaren Jahreszahlen sind Schaltjahre, haben also 366 Tage. Die durch 100 teilbaren Jahreszahlen sind jedoch keine Schaltjahre. Dagegen sind die durch 400 teilbaren Jahreszahlen doch Schaltjahre.

Soviel zu den Regeln. Wie soll unser Programm nun aufgebaut sein? Es würde reichen, wenn das Programm aus drei Blöcken bestände:

1. Datumseingabe
2. Berechnung
3. Ergebnis ausgeben

Den ersten Block können wir noch weiter verfeinern:

1.1 Bildschirm löschen
1.2 Titelbild drucken
1.3 Datum eingeben

Der letzte Block könnte so aussehen:

3.1 Bildschirm löschen
3.2 Eigegebenes Datum ausgeben
3.3 Wochentag ausgeben
3.4 Frage, ob weitere Berechnungen durchgeführt werden sollen

Hier nun der Programmablaufplan und das Struktogramm, wie wir es entwickelt haben.

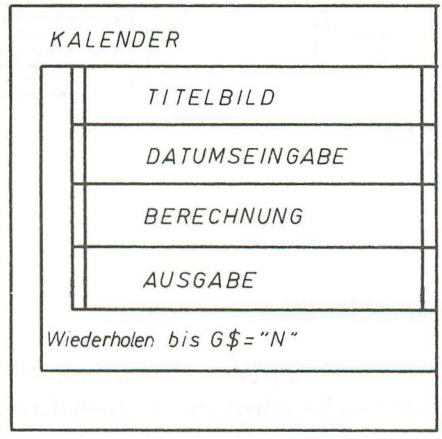


Bild 9.18: *Struktogramm des Programms KALENDER*

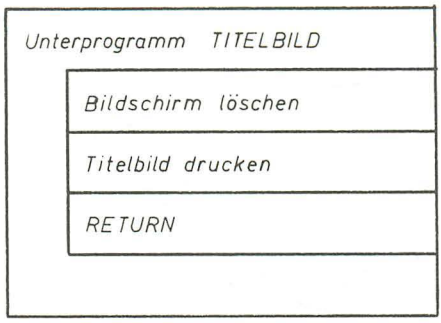


Bild 9.19: *Struktogramm des Unterprogramms TITELBILD*

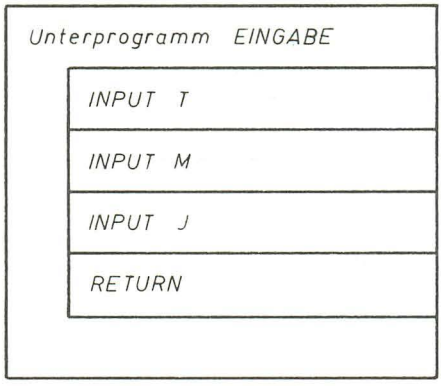


Bild 9.20: *Struktogramm des Unterprogramms EINGABE*

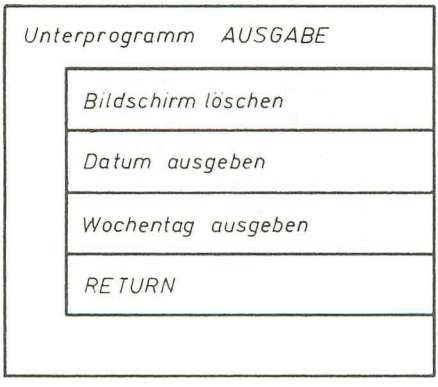


Bild 9.21: *Struktogramm des Unterprogramms AUSGABE*

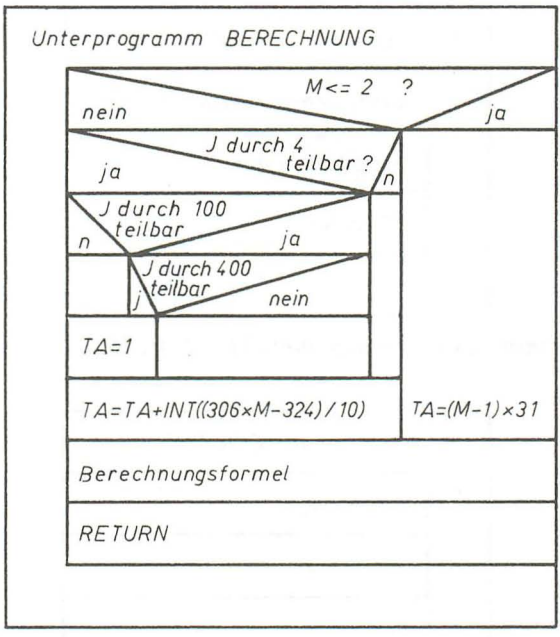


Bild 9.22: *Struktogramm des Unterprogramms BERECHNUNG*

Sie sehen, beim Struktogramm sind wir einen anderen Weg gegangen. Die Programmblöcke werden hier als Unterprogramme eingesetzt. Diese Unterprogramme werden so lange wiederholt, bis der Anwender bei der Frage zum Schluß ein »N« eingibt. Bei dem Struktogramm gibt es also eine Hauptroutine, die die Rolle eines »Managers« übernimmt und die einzelnen Unterprogramme aufruft.

Für jedes Unterprogramm gibt es ein eigenes Struktogramm. Das Struktogramm für die Eingabe erfüllt allerdings nicht ganz unsere Bedingungen. Wir haben gesagt, daß die Daten vor 1583 nicht berechnet werden dürfen. Eine entsprechende Abfrage müßte also noch eingebaut werden. Außerdem gibt es noch ein weiteres »Unterprogramm«, nämlich das Einlesen der Wochentage in ein Variablenfeld. Versuchen Sie doch einmal selbst, diese Änderungen vorzunehmen.

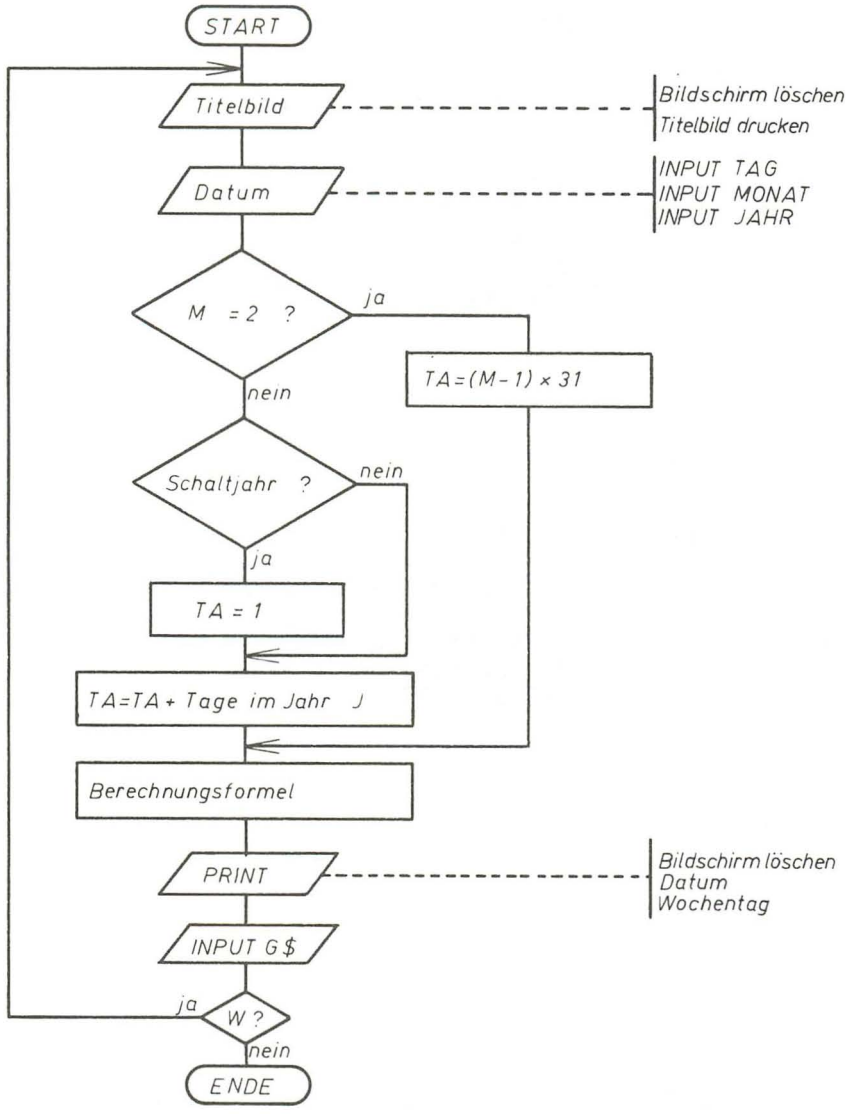


Bild 9.23: *Programmablaufplan des Programms KALENDER*

Hier nun das BASIC-Programm:

```
10 REM KALENDER
20 FOR I=0TO6:READW$(I):NEXT:REM WOCHENTAGE
30 :
40 DO
50 GOSUB 200:REM TITELBILD
60 GOSUB 400:REM EINGABE
70 GOSUB 500:REM BERECHNUNG
80 GOSUB 700:REM AUSGABE
90 PRINT:PRINT"WEITERE BERECHNUNG (J/N)";
100 GETKEY G$
110 IF G$="N"THEN EXIT
120 LOOP
130 END
200 :
210 REM TITELBILD
220 PRINTCHR$(147)
230 PRINT
240 PRINT" WOCHENTAGBERECHNUNG"
250 PRINT
260 PRINT" GEBEN SIE EIN BELIEBIGES DATUM EIN."
270 PRINT
280 PRINT" ICH WERDE IHNEN DEN WOCHENTAG"
290 PRINT" FUER IHR DATUM NENNEN. BITTE KEIN"
300 PRINT" DATUM VOR 1583 EINGEBEN."
310 PRINT
320 PRINT
330 RETURN
400 :
410 REM EINGABE
420 INPUT"TAG";T
430 INPUT"MONAT";M
440 INPUT"JAHR";J
450 IFJ<1583THEN420
460 RETURN
500 :
510 REM BERECHNUNG
520 IF M<=2 THEN TA=(M-1)*31:GOTO560
530 IF J/4 = INT(J/4) THENTA=1
540 IF J/100=INT(J/100)ANDJ/400=INT(J/400)THEN TA=1:ELSETA=0
550 TA = TA+INT((306*M-324)/10)
560 TA = TA+(J-1)*365+INT((J-1)/4)
570 TA = TA-INT((J-1)/100)+INT((J-1)/400)
580 TA = TA+T
590 TA = TA-INT(TA/7)*7
600 RETURN
700 :
710 REM AUSGABE
720 PRINT CHR$(147)
730 PRINT"DER";T;".";M;".";J;"IST EIN ";W$(TA)
740 PRINT
```

```
750 RETURN
800 :
810 REM DATAS FUER WOCHENTAGE
820 DATA SONNTAG,MONTAG,DIENSTAG,MITTWOCH
830 DATA DONNERSTAG,FREITAG,SONNABEND
```

So, wenn Sie das Programm fehlerfrei eingegeben haben, müßte es eigentlich auch einwandfrei funktionieren. Die Berechnung im Unterprogramm ab Zeile 500 ist genau so wie im Struktogramm angegeben. Die Variable TA gibt zunächst die Anzahl der Tage vom Jahr 0 bis zum eingegebenen Jahr einschließlich Schalttage an. Hierbei werden die Regeln des Gregorianischen Kalenders auch für die Jahre vor 1583 angewandt. Da wir aber die Eingabe dieser Jahre in unserer Eingaberoutine schon unterbinden, kann keine falsche Berechnung gemacht werden. Natürlich hätten wir auch die Differenz zwischen dem eingegebenen Jahr und 1583 ausrechnen können, aber das ist nicht notwendig. Es funktioniert auch so.

Nun noch ein Tip, wie Sie dieses Programm noch verbessern können: In der Eingaberoutine wird nur geprüft, ob die eingegebene Jahreszahl größer als 1583 ist. Was passiert aber, wenn Sie sich nicht an die Regel halten, und z. B. 45.13.1986 eingeben? Das Programm wird Ihnen antworten, daß der 45.13.1986 ein Sonnabend ist. Da stimmt doch was nicht! Es müßte also noch eine Überprüfung der eingegebenen Tages- und Monatszahlen erfolgen. Versuchen Sie doch einmal diese Überprüfung zu programmieren.

Dieses Programm könnte man auch dazu verwenden, einen Biorhythmus zu berechnen. Das ist aber schon etwas schwieriger. Dazu müßten die vergangenen Tage von einem Geburtsdatum bis zum aktuellen Datum errechnet werden. Und dann müßten drei Sinuskurven gezeichnet werden. Eine für die sogenannte »körperliche Aktivität«, ein 23-Tage-Rhythmus, eine für das Gefühlsleben (28 Tage) und eine Kurve für die intellektuelle Leistung (33 Tage). Die Tageszahlen beziehen sich jeweils auf die vergangenen Tage seit der Geburt. Nun, diese Berechnungen sind sehr umstritten, sie sind aber eine recht lustige Sache, wenn man das Ergebnis einmal den Freunden oder Bekannten zeigt. Der Plus/4 wird Ihnen mit der hochauflösenden Grafik eine große Hilfe geben bei der Realisierung eines solchen Programms.

Sicher werden Sie meinen, daß das Anfertigen der Struktogramme eine sehr aufwendige Arbeit ist, wenn Sie aber umfangreiche oder sehr umfangreiche Programme schreiben wollen, werden Sie letztendlich viel Zeit dadurch sparen. Denn das Umschreiben eines Struktogramms in ein lauffähiges Programm erfordert weniger Zeit, als wenn Sie einfach »drauflos« programmieren. Sie erkennen auch an unserem einfachen Beispiel, daß eine Aufteilung des Programms in Unterprogramme dieses viel übersichtlicher macht. Machen Sie sich während der Arbeit viele Notizen, und schreiben Sie alle Variablen auf, die Sie verwenden.

10 Die Eingabe von Zeichen

Die wenigsten Programme, die Sie erstellen werden, werden ohne Zeicheneingabe auskommen. Soll ein Programm flexibel sein, so kommt man um das Eingeben von Zahlen oder Buchstaben nicht herum. Dazu ein Beispiel: Angenommen, Sie möchten ein Programm schreiben, welches Ihnen Rechteckflächen berechnet, dann könnten Sie folgendes machen:

```
10 PRINT CHR$(147)
20 A=15
30 B=10
40 PRINT A*B
```

Programmbeschreibung:

Zeile 10	Hier wird der Bildschirm gelöscht und der Cursor in die linke obere Ecke gesetzt.
Zeile 20	Der Variablen A wird der Wert 15 zugewiesen.
Zeile 30	Die Variable B erhält den Wert 10.
Zeile 40	Die Variable A wird mit der Variablen B multipliziert, das Ergebnis wird auf dem Bildschirm ausgegeben.

Nach Ablauf des Programms erhalten Sie die Zahl 150 auf dem Bildschirm.

Um jetzt ein Rechteck mit der Seitenlänge A=5 und B=25 zu berechnen, müßten Sie erst das Programm ändern. Dazu muß erst einmal das Programm mit LIST auf den Bildschirm gebracht werden. Dann geht's mit dem Cursor in die Zeile 20 auf die Fünf. Mit der Taste INST/DEL wird nun die Eins gelöscht. Nun noch die RETURN-Taste gedrückt, damit der Rechner die geänderte Programmzeile übernimmt und der Cursor steht in der Zeile 30. Jetzt wird der Cursor auf die Eins gesetzt und die Zahl 25

eingegeben. Nach einem weiteren RETURN ist auch diese Programmzeile übernommen. Nun noch schnell einmal die Taste CRSR DOWN gedrückt, dann SHIFT-F3 und auf dem Bildschirm erscheint, was man sich während der ganzen Prozedur schon im Kopf errechnete, das Ergebnis 125. Na, hat's Spaß gemacht?

Vielleicht noch beim ersten Mal, aber wie sieht es aus, wenn Sie 130 Rechtecke berechnen müssen? Sie werden mit mir übereinstimmen, daß man bei Betrachtung dieses Aufwands sicher sehr schnell zu dem Schluß kommt, daß sich so ein Problem mit einem ganz normalen Taschenrechner schneller lösen läßt. Das Programm krankt daran, daß man ihm keine Daten von außen übergeben kann. Mit den folgenden Befehlen können Sie dieses Manko beseitigen.

10.1 Der INPUT-Befehl

Syntax:

INPUT "Kommentar" ; Variable,Variable,...

Der Kommentar kann auch entfallen, dann sieht die Syntax so aus:

INPUT Variable,Variable,...

Trifft ein Programm auf diesen Befehl, so passiert folgendes: Das Programm stoppt. Sollte dem INPUT-Befehl ein Kommentar folgen, so wird dieser auf dem Bildschirm angezeigt. Es erfolgt die Ausgabe eines Fragezeichens auf dem Bildschirm. Der Cursor wird eingeschaltet und erscheint hinter dem Fragezeichen. Das Programm läuft erst nach dem Drücken der RETURN-Taste weiter.

Ändern wir nun einmal das Programm wie folgt und starten es dann.

```
20 INPUT A
30 INPUT B
```

Geben wir nun hinter dem ersten Fragezeichen eine Vier ein und beenden dann die Eingabe mit der RETURN-Taste, so erscheint ein zweites Fragezeichen unter dem ersten. Nun geben wir 15 ein und beenden wiederum die Eingabe mit der RETURN-Taste. Auf dem Bildschirm erscheint nun die Zahl 60.

Sie sehen, das Programm ist wesentlich flexibler geworden, nur mit der Übersichtlichkeit ist es nicht zum besten bestellt. Wie soll ein Fremder oder auch man selbst noch nach ein paar Wochen wissen, welche Eingaben man zu tätigen hat? Aber auch dieses Problem läßt sich leicht lösen, bietet doch der INPUT-Befehl die Möglichkeit eines Kommentars. Also ändern wir einmal:

```
20 INPUT "LAENGE DER ERSTEN SEITE ";A
30 INPUT "LAENGE DER ZWEITEN SEITE ";B
```

Das Ergebnis nach dem Start werden Sie sicher erwartet haben.

Wie Sie der Syntax dieses Befehles entnehmen können, ist es auch möglich, mit einem INPUT-Befehl mehreren Variablen Werte zuzuweisen. In diesem Falle müssen die verschiedenen Variablen durch Kommata (,) getrennt werden. Die Eingaben müssen dann ebenfalls Kommata zwischen den einzelnen Werten enthalten. Bauen wir unser Programm doch einmal um.

```
20 INPUT "LAENGE ERSTE,ZWEITE SEITE ";A,B
```

Die Zeile 30 muß gelöscht werden!

Nach dem Start und der Meldung des Programms geben wir nun 30,40 ein. Wir erhalten nach RETURN dann den Wert 1200. Sie sehen, auch das funktioniert. Sollten Sie nur einen Wert eingeben, so erscheinen zwei Fragezeichen auf dem Bildschirm, die Sie auf Ihr Versäumnis aufmerksam machen. Sie werden in die Lage versetzt, den fehlenden Wert nachträglich einzugeben.

Sie können mit dem INPUT-Befehl aber nicht nur Zahlen, sondern auch Zeichenketten eingeben. Der Variablen muß dann allerdings ein $ (Dollarzeichen) angehängt werden. Näheres werden Sie im Kapitel über Strings erfahren.

Einige Besonderheiten des INPUT-Befehls

Wenn das Programm auf die Eingabe wartet, können Sie es nicht mit der STOP-Taste abbrechen. In diesem Falle hilft nur die RETURN-Taste weiter und, sobald das Programm fortfährt, die STOP-Taste.

Sollte der INPUT-Befehl einen numerischen Wert erwarten und Sie statt dessen Buchstaben eingeben, erscheint die Fehlermeldung »?REDO FROM START«. Das Programm wird nicht abgebrochen, sondern der Kommentar des INPUT-Befehls erscheint erneut und der Rechner wartet weiter auf die richtige Eingabe.

Soll mit dem INPUT-Befehl ein String eingegeben werden, so darf er kein Komma enthalten, da dieser Befehl nach dem Komma ja die Zuweisung für eine weitere Variable erwartet. Sollte diese fehlen, so erscheint die Meldung »EXTRA IGNORED«. Das Programm wird nicht abgebrochen, nur die Zeichen nach dem Komma wurden nicht mit in die Stringvariable übernommen.

In der Zeit, in der der Cursor sichtbar ist, ist auch der Bildschirmeditor wieder eingeschaltet. D. h., daß Sie zum Beispiel mit den Cursortasten den Cursor beliebig auf dem Bildschirm bewegen können. Falsch eingegebene Zeichen können Sie wie gewohnt mit der INST/DEL-Taste löschen. Auch ist es möglich, den Bildschirm mit SHIFT CLEAR/HOME zu löschen.

Diese Besonderheiten sind nicht immer erwünscht. Ein falscher Tastendruck, und der Bildschirmaufbau ist zerstört. Bei Verwendung dieses Befehls kann man als Programmierer sein Programm vor Fehlbedienung kaum schützen, auch ist es manchmal nicht angebracht, daß das Programm auf eine Eingabe wartet, sondern mit der Abarbeitung fortfährt (zum Beispiel bei Spielen). Aus diesem Grund erhielt der Plus/4 noch weitere Eingabebefehle.

10.2 Der GETKEY-Befehl

Syntax:

GETKEY Variable,Variable,...

Die Syntax dieses Befehles gestattet nicht, einen Kommentar mit auszugeben. Wird ein Kommentar erwünscht, so muß dieser vorher mit dem PRINT-Befehl auf den Bildschirm gebracht werden. Beim Antreffen des Befehls verhält sich das Programm so: Der Programmablauf wird unterbrochen. Der Cursor bleibt ausgeschaltet. Das Programm wird durch irgendeinen Tastendruck, mit Ausnahme der SHIFT-, CTRL- oder COMMODORE-Taste, fortgesetzt.

Dazu folgendes Programm zur näheren Erklärung:

```
10 PRINT"ICH WARTE AUF DEINE EINGABE "
20 GETKEY A$
30 PRINT"DU HAST "A$" EINGEGEBEN"
40 GOTO 10
```

Falls Sie beim Eingeben der Zeile 30 etwas verwirrt werden, so macht das nichts. Über den PRINT-Befehl erfahren Sie im Kapitel über die Zeichenausgabe mehr.

Wenn Sie das Programm starten, wartet es nach Ausgabe des Satzes in Zeile 10 auf Ihren Tastendruck. Nach dem Druck einer beliebigen Taste (Einschränkungen siehe oben), wird die gedrückte Taste angezeigt. In Zeile 40 wird das Programm angewiesen, mit der Zeile 10 fortzufahren, d. h. in diesem Fall, daß das Programm von neuem abläuft.

Wenn Sie die Taste SHIFT und die Taste CLEAR/HOME zur gleichen Zeit drücken, werden Sie feststellen, daß auch diesmal der Bildschirmeditor funktioniert. Der Schein trügt. Der Bildschirmeditor ist in Wahrheit abgeschaltet! Verursacht wird das Bildschirmlöschen durch den PRINT-Befehl in Zeile 30. Sie haben ja die notwendigen Tasten zum Bildschirmlöschen gedrückt. Dieses wird in der Stringvariablen A$ gespeichert. In Zeile 30 schließlich erhält der Rechner den Befehl, das gespeicherte Zeichen auszugeben, und treu wie er ist, macht er es ja auch. Sie können dies aber verhindern. Ergänzen Sie bitte einmal das Programm um folgende Zeile, starten Sie es, und versuchen Sie dann einmal, den Bildschirm zu löschen.

```
25 IF A$=CHR$(118) THEN GOTO 20
```

Nun können Sie auch den Unterschied zum INPUT-Befehl feststellen. Mit dem GETKEY-Befehl ist das »Ausschalten« einzelner Tasten möglich geworden. Natürlich wurde die Taste CLEAR/HOME nicht wirklich abgeschaltet. Das Programm bekommt vielmehr den Befehl, daß, wenn diese Taste gedrückt wird, zurück zur Eingabe verzweigt werden soll. Vielleicht haben Sie auch schon festgestellt, daß nach dem Drücken der Funktionstasten das Programm mit einer Fehlermeldung abbricht. Hier die Lösung dieses Problemes:

KEY 1,""

Mit diesem Befehl wurde die Belegung der F1-Taste gelöscht, und es ist nun nicht mehr möglich, mit dieser Taste das Programm zum »Absturz« zu bringen.

Ändern wir jetzt noch einmal das Programm.

```
20 GETKEY A$,B$
30 PRINT"DU HAST "A$" UND "B$" EINGEGEBEN"
```

Wenn Sie das Programm starten, werden Sie feststellen, daß das Programm erst nach Eingabe des zweiten Zeichens mit der Abarbeitung fortfährt. Es werden aber dann beide Zeichen ausgegeben.

Mit dem Befehl GETKEY ist es also möglich, das Programm wie beim INPUT-Befehl anzuhalten um auf eine Zeicheneingabe über die Tastatur zu warten. GETKEY nimmt aber nur soviel Zeichen an, wie Variablen, durch Kommata getrennt, vorhanden sind. Das Drücken der RETURN-Taste entfällt.

Wird das Zeichen nicht auf dem Bildschirm ausgegeben, so kann der Bildschirmaufbau nicht zerstört werden. Ist es notwendig, und das ist in den meisten Anwendungen der Fall, die eingegebenen Zeichen anzuzeigen, so kann man unbeliebte oder natürlich auch in einem Programm unerlaubte Zeichen unterdrücken.

Sicher wird jetzt der eine oder andere von Ihnen sagen: »Das ist ja alles schön und gut, nur ich möchte nicht, daß mein Programm angehalten wird, nur damit eine Eingabe getätigt werden kann.« Aber auch für dieses Problem gibt es eine Lösung:

10.3 Der GET-Befehl

Syntax:

GET Variable,Variable,...

Zuerst ein weiteres kleines Programm:

```
10 I=0
20 GET A$
30 PRINT A$
40 I=I+1
50 PRINT I
60 GOTO 20
```

Wenn Sie dieses Programm starten, dann werden Sie über den Bildschirm Zahlen nur so huschen sehen. Verlangsamen können Sie den Programmablauf durch Drücken der COMMODORE-Taste. Drücken Sie nun eine Taste, so wird deren Belegung wie beim GETKEY-Befehl sofort angezeigt. Das Programm macht also keine Pause, sondern es fährt, ganz egal, ob eine Taste gedrückt wurde oder nicht, mit der Abarbeitung des Programms fort. Etwas anderes wird Ihnen auch sehr schnell auffallen: Das Betätigen der F-Tasten verursacht keinen Programmabsturz mehr.

Die Programmbeschreibung:

Zeile 10	Hier wird die Variable I auf 0 gesetzt. Diese Zeile ist in diesem Programm, wenn mit RUN gestartet wird, zwar umsonst, wurde aber der Übersichtlichkeit zuliebe aufgenommen.
Zeile 20	Sollte eine Taste gedrückt werden oder schon vorher gedrückt worden sein, so wird deren Code in der Stringvariablen A$ festgehalten.
Zeile 30	In dieser Zeile wird A$ angezeigt oder der Code ausgeführt.
Zeile 40	Zu der Variablen I wird 1 addiert.
Zeile 50	Der Wert der Variablen I wird auf dem Bildschirm ausgegeben.
Zeile 60	Das Programm verzweigt zur Zeile 20.

Vielleicht werden Sie sich über die Beschreibung der Zeile 20 etwas wundern, aber es stimmt. Auch wenn sich das Programm zum Beispiel in der Zeile 40 befindet, wird die Tastatur abgefragt, und falls eine Taste gedrückt ist, wird diese im Rechner gespeichert.

Der Beweis:

```
10 REM
20 GET A$
30 PRINT A$
40 FOR I=0 TO 1500 : NEXT
50 GOTO 10
```

Wenn Sie dieses Programm gestartet haben, dann drücken Sie bitte einige Tasten schnell hintereinander, lehnen sich zurück und achten dann darauf, was passiert.

Das Ergebnis wird sein, daß Zeichen auf dem Bildschirm ausgegeben werden, obwohl Sie keine Taste gedrückt halten. Da der Rechner natürlich auch seine Grenzen hat, werden maximal zehn Zeichen zwischengespeichert. Alle weiteren eingegebenen Zeichen werden ignoriert.

Das eben Beschriebene gilt auch für die Befehle INPUT und GETKEY. An dieser Stelle möchte ich einmal etwas technischer werden, da bei manchem sicher auch die Frage auftauchen wird, wie die Zeichen zwischengespeichert werden.

Der Plus/4 verfügt wie alle Rechner über einen vor der Programmiersprache geschützten Speicherbereich, in dem für seine einwandfreie Funktion wichtige Werte abgelegt werden. Woher sollte er zum Beispiel sonst auch wissen, an welcher Stelle des Bildschirms sich der Cursor befindet? Ein Teil dieses Bereichs ist der sogenannte Tastaturpuffer. Er hat eine Länge von zehn Speicherzellen. Daneben existiert noch eine Speicherzelle, die dem Rechner mitteilt, wie viele Zeichen sich zur Zeit im Tastaturpuffer befinden.

Arbeitet der Rechner zum Beispiel eines Ihrer Programme ab und Sie drücken nun eine Taste, so wird das Zeichen erst einmal in den Tastaturpuffer übernommen und dann die Zählerspeicherzelle auf eins gesetzt. Dies kann, wie oben schon erwähnt, maximal zehnmal geschehen. Trifft er nun auf einen der Tastaturabfragebefehle wie INPUT, GETKEY oder GET, dann wird der Zähler überprüft. Hat dieser den Wert Null, so wartet er bei den Befehlen INPUT und GETKEY auf einen Tastendruck. Bei dem Befehl GET wird das Programm sofort weitergeführt. Sollte der Zähler aber einen Inhalt größer Null enthalten, so werden ein oder mehrere Zeichen aus dem Puffer geholt und der oder den Variablen übergeben. Der Zähler wird entsprechend zurückgezählt.

Trifft der Rechner während des Programmes auf keinen derartigen Befehl, so werden die im Puffer befindlichen Zeichen nach Beendigung des Programms im Direktmodus ausgegeben.

Es wird sicher später bei Ihren Problemlösungen vorkommen, daß während des Programmablaufs gedrückte Tasten nicht beachtet werden sollen. Hier unsere Lösung:

POKE 239,0

Wenn Sie diese Zeile in Ihr Programm aufnehmen, dann werden alle Zeichen, die nach der Abfrage der Tastatur eingegeben wurden, ignoriert. Dieser Befehl setzt den Zähler zurück und gaukelt so dem Rechner einen leeren Tastaturpuffer vor.

ACHTUNG: Dieser Befehl ist nur für GETKEY und INPUT geeignet, da GET dermaßen schnell abgearbeiter wird (kein Programmstop), daß es Ihnen sehr selten gelingen wird, ein Zeichen an Ihr Programm zu übergeben.

Ihnen wird aufgefallen sein, daß in den Beispielen über den GETKEY- und den GET-Befehl nur Stringvariablen benutzt werden, da beim Übergeben von Buchstaben an eine numerische Variable die Programme mit der Fehlermeldung »TYPE MISMATCH ERROR IN...« ausgestiegen wären. Dem kann man begegnen, Beispiele gibt es im Kapitel über die Fehlersuche und die Fehlerbehandlung, aber man sollte sich sehr früh angewöhnen, Fehler erst gar nicht zuzulassen.

Ist zum weiteren Programmablauf eine numerische Eingabe notwendig, so ist dies mit dem Plus/4 kein Problem, bietet er einem doch die Möglichkeit, Strings in numerische Variablen umzuformen. Hierzu und darüber, wie man komfortabel viele Zeichen auf einmal bei einer Eingaberoutine zuläßt oder ausblendet, gibt's im Kapitel über Strings nähere Informationen.

Wenn Sie nach dem Durchlesen dieses Kapitels Ihre eigenen Eingaberoutinen programmieren, werden Sie sehr schnell merken, daß die Befehle GETKEY und GET am besten zu handhaben sind. Der Programmieraufwand ist sicher größer als beim Befehl INPUT, dafür kann man aber unerwünschte Zeichen ausblenden und dem Bediener die Denkarbeit, ob nun numerische oder alphabetische Eingaben erforderlich sind, abnehmen.

11 Zeichenausgabe auf dem Bildschirm

Ein Programm lebt zu großen Anteilen von der Darstellung seiner Ergebnisse auf dem Bildschirm. Für den Anwender eines Programms kommt es immer darauf an, daß er sofort sieht, welche Eingaben erforderlich sind, und daß die Ausgabe der Berechnungen übersichtlich angeordnet erfolgt. Hier zuerst einmal ein Beispiel:

```
20 INPUT "GEBEN SIE EINE ZAHL EIN";ZA
30 PRINT ZA*2
40 PRINT ZA*5
```

Starten Sie das Programm und beantworten Sie dann die Fragen mit der Zahl 2. Nach RETURN erscheinen untereinander angeordnet die Zahlen 4 und 10. Die Ergebnisse stimmen, aber wie sieht nur der Bildschirm aus! Zuerst sieht man das eben geschriebene Programm, darunter die Frage nach der Zahl und dann kommen die Ergebnisse. Finden Sie dies optimal? Fangen wir an, etwas Ordnung zu machen.

11.1 Der PRINT-Befehl

Syntax:

PRINT "Text" Steuerzeichen

oder

PRINT Variable Steuerzeichen

oder Kombinationen von beiden

Mit dem PRINT-Befehl können Sie Zeichen auf dem Bildschirm ausgeben. In welcher Form dies geschieht, bestimmen die Steuerzeichen.

Steuerzeichen sind: , (Komma) und ; (Semikolon)

Zur Erläuterung ein Programm. Geben Sie vorher bitte NEW ein.

```
10 PRINT "DER"
20 PRINT "PLUS/4"
30 PRINT "COMPUTER"
```

Wird dieses Programm gestartet, so erscheinen die einzelnen Worte untereinander auf dem Bildschirm. Trifft das Programm auf einen PRINT-Befehl, so werden alle Zeichen, die sich in Anführungszeichen befinden, ausgegeben. Sollten die Anführungszeichen fehlen, so werden die Zeichen als Variablennamen interpretiert und die Werte dieser Variablen werden auf dem Bildschirm dargestellt.

Wird der PRINT-Befehl von keinem Steuerzeichen gefolgt, so löst der nachfolgende PRINT-Befehl einen Zeilenvorschub aus, d. h. der Cursor wird in die erste Spalte der nachfolgenden Zeile gesetzt. Erfolgt die Ausgabe in der letzten Zeile des Bildschirms, so wird beim nächsten PRINT der Bildschirminhalt um eine Zeile nach oben gescrollt und der Inhalt der ersten Zeile geht verloren. Der Plus/4 bietet die Möglichkeit, das Scrollen zu verhindern. Näheres darüber können Sie bei der Behandlung der ESC-Funktionen erfahren.

Oft sollen mehrere Ausgaben in einer Zeile dargestellt werden. Somit sind wir auch schon bei der Bedeutung der Steuerzeichen, die dem PRINT-Befehl folgen können.

11.1.1 Die Bedeutung des Kommas beim PRINT-Befehl

Der Cursor wird von einem Programm, obwohl er während des Programmablaufes nicht sichtbar ist, gesteuert. Der Bildschirm des Plus/4 hat 40 Spalten und 25 Zeilen. Stellen Sie sich eine Bildschirmzeile in vier gleich große Bereiche aufgeteilt vor. Das ergibt 4x10 Spalten mit 25 Zeilen. Folgt nun ein Komma dem Ausgabetext oder der Variablen, so erfolgt die Ausgabe beim nächsten PRINT-Befehl im nächsten Viertel der Ausgabezeile. Ein Beispiel:

NEW (Nur notwendig, wenn sich noch das vorherige oder ein anderes Programm im Speicher befinden sollte)

```
10 PRINT"1","2","3","4"
```

Sie sehen, die Zeichen stehen in einer Zeile und sind durch jeweils neun Leerzeichen getrennt. Möchte man aber nur die Zahlen 3 und 4 im dritten und vierten Viertel ausgeben, dann geht man wie folgt vor:

```
10 PRINT,,"3","4"
```

Merken wir uns also: Folgt ein Komma dem PRINT-Befehl, so wird der Cursor in das nächste Viertel der aktuellen Bildschirmzeile gesetzt.

Folgen mehrere Kommas dem PRINT-Befehl, so wandert der Cursor durch jedes Komma zehn Bildschirmspalten nach rechts. Versuchen Sie auch folgendes einmal:

```
10 PRINT,,"3",,,"2"
```

Trifft der Rechner auf diese Programmzeile, so nimmt er an, daß er das dem PRINT-Befehl nachfolgende Zeichen ausgeben soll. Da dieses aber ein Komma ist, setzt er den Cursor zehn Spalten weiter. Das nächste Zeichen ist ebenfalls ein Komma, also wird der Cursor noch einmal zehn Zeichen nach rechts gesetzt. Jetzt trifft er auf das erste Anführungszeichen, das heißt für ihn, daß er die nachfolgenden Zeichen bis zum Antreffen der zweiten Anführungszeichen auszugeben hat. In diesem Falle ist es nur ein Zeichen. Dieses wird auf dem Schirm ausgegeben. Danach folgen wieder Kommata. Nach dem ersten Komma wird der Cursor in die Spalte 30 gesetzt. Da der Plus/4 nur 40 Zeichen in einer Zeile darstellen kann, springt der Cursor, nachdem der Rechner auf das nächste Komma trifft, in die erste Spalte der nächsten Reihe. Das folgende Komma setzt den Cursor noch einmal um zehn Spalten nach rechts, und endlich wird die 2 auf dem Bildschirm ausgegeben. Nach Ablauf des Programms steht daher die 3 im dritten Viertel des Bildschirmes und die 2 im zweiten Viertel der nächsten Zeile. Dieses Steuerzeichen läßt sich sehr gut zur Aufstellung von Tabellen verwenden.

Auch hierfür ein Beispiel:

```
10 INPUT "ZAHL";ZA
20 PRINT
30 PRINT "2*ZAHL=",2*ZA
40 PRINT
80 GOTO 10
```

Die Erklärung:

Zeile 10: Der Rechner gibt das Wort 'ZAHL' aus und wartet auf eine numerische Eingabe. Nach der Eingabe und RETURN springt der Cursor in die nächste Zeile

Zeile 20: Da dem PRINT-Befehl weder ein Text oder ein Variablenname noch ein Steuerzeichen folgt, wird eine Leerzeile ausgegeben.

Zeile 30: Der Text »2*ZAHL=« wird ausgegeben, dann springt, weil ein Komma folgt, der Cursor in das nächste Viertel des Bildschirms. Es wird nun 2*ZA ausgerechnet und das Ergebnis auf dem Bildschirm angezeigt.

Zeile 40: Es wird abermals eine Leerzeile angezeigt.

Zeile 80: Das Programm verzweigt wieder zur Zeile 10.

Wie Sie in der Zeile 30 erkennen können, ist zur Ausgabe des Ergebnisses kein weiteres PRINT nötig. Dies gilt immer dann, wenn Zeichenketten, Variablen oder Kombinationen von beiden hintereinander stehen und nur durch Leerzeichen, Komma oder Semikolon getrennt sind.

Dieses Programm kann nur durch sehr schnelles Drücken der STOP-Taste nach RETURN unterbrochen werden. Machen wir es doch komfortabler:

```
50 PRINT "SOLL EINE WEITERE ZAHL FOLGEN ? (J)"
60 GETKEY A$
70 IF A$ <> "J" THEN END
```

Die Erklärung dieser Zeilen finden Sie bei den logischen Verknüpfungen und im Kapitel über die Eingabe von Zeichen.

Abschließend zu diesem Steuerzeichen noch ein Demonstrationsprogramm:

```
10 REM DEMO ZUM "," ALS STEUERZEICHEN BEIM PRINT-BEFEHL
20 PRINT CHR$(147): REM BILDSCHIRM LÖSCHEN
30 PRINT "DIE AUSGABE DER ZAHLEN 1 BIS 40"
40 PRINT :  REM LEERZEILE ERZEUGEN
50 PRINT "UNGERADE ZAHLEN","GERADE ZAHLEN"
60 FOR I=1 TO 40 STEP 2
70 PRINT I,,I+1
80 NEXT I
90 GETKEY A$
```

Bis auf die Zeilen 60 und 80, die im Kapitel über die Programmschleifen näher erklärt werden, müßten Sie dieses Programm eigentlich verstehen. Sollte dies nicht der Fall sein, so lesen Sie doch noch einmal von vorn.

11.1.2 Das Semikolon beim PRINT-Befehl

Vermutlich werden Sie oft Programme schreiben, die ohne eine vorherige Anleitung nicht auskommen. Sollte diese Anleitung länger als eine Programmzeile sein, kommt man ohne das Semikolon (;) bei dem PRINT-Befehl nicht aus. Es ermöglicht das Ausgeben von zusammenhängenden Sätzen auf dem Bildschirm, obwohl Teile dieser Sätze eventuell in einer ganz anderen Programmzeile stehen. Auch das Einfügen von Variableninhalten in einen Text ist nahtlos möglich. Nehmen wir zur Demonstration noch einmal das Beispiel vom Anfang dieses Kapitels und verändern es leicht.

```
10 PRINT "DER";
20 PRINT "PLUS/4";
30 PRINT "COMPUTER"
```

Nach RUN erscheinen alle Worte unmittelbar hintereinander auf dem Bildschirm. Um den Text lesbar zu machen, wird es also notwendig, dem Wort »DER« und dem Wort »PLUS/4« ein Leerzeichen (Space) anzuhängen. Nach getaner Arbeit erscheint der Satz leicht lesbar auf dem Bildschirm. Ist der darzustellende Satz länger als eine Bildschirmzeile (40 Zeichen), dann erfolgt die Ausgabe in der ersten Spalte der nächsten Zeile.

Sollen numerische Variablen in einem Text erscheinen, so ist zu beachten, daß den Zahlen vom Rechner bei der Ausgabe ein Leerzeichen vorangestellt und ein Leerzeichen angehängt werden. Ändern Sie bitte die Zeilen 5 und 20 wie folgt:

```
5  A=4
20 PRINT "PLUS/";A;
```

Die Ausgabe erfolgt, obwohl dem Wort »PLUS/« kein Leerzeichen folgt und dem Wort »COMPUTER« kein Leerzeichen vorangestellt ist, mit Leerzeichen zwischen den einzelnen Worten. Dies gilt aber wie schon gesagt nur für numerische Variablen. Stringvariablen werden wieder unmittelbar in den Text eingefügt. Der Beweis:

```
5  A$="4"
20 PRINT "PLUS/";A$;
```

Das Semikolon zwischen »PLUS/« und A$ braucht nicht mit eingegeben zu werden. Die Textausgabe funktioniert auch in diesem Falle einwandfrei. Probieren Sie's!

Möchte man ein Programm mit einem längeren Einleitungstext versehen oder sollen Texte während eines Programmablaufes auf dem Bildschirm erscheinen, so bereitet dies oft Probleme bei der Eingabe. Durch die Zeilennummer und den PRINT-Befehl kann man beim Programmieren sehr schlecht das Format erkennen, in dem der Text später angezeigt wird. Meistens müssen nach dem ersten Programmablauf noch Leerzeichen eingegeben oder Worte in andere Programmzeilen übertragen werden, um den Text auf dem Bildschirm übersichtlich zu gestalten. Ein späteres Einfügen oder Löschen von Worten und Sätzen ist nur schwer zu bewerkstelligen.

Das Programm beendet diese Misere durch eine spezielle Ein- und Ausgabetechnik. Die Texte müssen in DATA-Zeilen abgelegt werden. Das Programm liest jeweils eine DATA-Zeile und untersucht diese nach Spaces (Leerzeichen). Trifft es auf ein Leerzeichen, so wird überprüft, ob der Teil des Strings bis zum Leerzeichen noch Platz in der aktuellen Bildschirmzeile hat. Ist dies der Fall, so wird diese Position gespeichert und die Position des nächsten Leerzeichens gesucht. Dieses wird so lange wiederholt, bis entweder der gesamte String durchsucht wurde - in diesem Fall wird der noch nicht ausgegebene Text zwischengespeichert - oder der Text keinen Platz mehr in einer Bildschirmzeile findet. Ist kein Platz mehr vorhanden, so wird der Text bis zum vorherigen Space ausgegeben, der aktuelle String um dieses Zeichen verkürzt und der Rest dann wieder nach Leerzeichen durchsucht.

Wenn das Programm auf den String »-1« trifft, wird der eventuell zwischengespeicherte Text ausgegeben und das Programm beendet. Es können auch sehr einfach Absätze oder Leerzeilen eingegeben werden.

Zur Bedienung:

Wird dieses Programm als Unterprogramm verwandt, so müssen Sie in der Zeile 30 den Befehl END in RETURN abändern, damit der Rücksprung in das Hauptprogramm erfolgen kann.

Absätze und Leerzeichen werden durch Leerstrings erzeugt. Trifft das Programm auf den ersten Leerstring, so erfolgt die weitere Ausgabe der Zeichen in der nächsten Zeile. Folgt nun ein weiterer Leerstring, so wird eine Leerzeile erzeugt. Die Länge der Strings in den DATA-Zeilen können eine ganze Programmzeile umfassen (80 Zeichen). Es sind alle darstellbaren Zeichen zugelassen.

Das Programm ist im Speicher durch RENUMBER verschiebbar. Sind mehrere Texte in einem Programm auszugeben, muß vorher der Datazeiger durch ein RESTORE Zeilennummer auf die auszugebende Textpassage gesetzt werden. Jeder dieser Text-

teile muß mit »-1« beendet werden. Achten müssen Sie auch darauf, daß jede DATA-Zeile mit einem Leerzeichen beendet werden sollte, da sonst Schwierigkeiten auftreten können.

Zur Programmierung:

Die REM-Zeilen oder die Zeilen in denen lediglich ein Doppelpunkt steht, brauchen nicht mit eingegeben zu werden. Der besseren Übersichtlichkeit sowie der späteren Dokumentation wegen sollte man aber diese geringe Mehrarbeit übernehmen. Die DATA-Zeilen dienen nur der Demonstration. Sie können später ebenfalls gelöscht werden, um sie durch eigene zu ersetzen.

Das Programm:

```
1 REM  **** KOMFORTABLE AUSGABE VON TEXTEN ****
2 :
3 REM  ***** DIE VERWENDETEN VARIABLEN : ******
4 REM A$= HIERIN BEFINDET SICH DER EINGELESENE STRING
5 REM B$= DIE AUSZUGEBENDE ZEILE WIRD HIER ZWISCHENGESPEICHERT
6 REM PO= DIE POSITION DES NÄCHSTEN SPACE
7 REM XO= DIE POSITION DES LETZTEN SPACE
8 :
9 :
10 PRINT CHR$(147) : REM BILDSCHIRM LÖSCHEN
20 READ A$         : REM STRING EINLESEN
25 REM BEI UNTERPROGRAMM STATT END RETURN EINGEBEN
30 IF A$="-1" THEN PRINT B$:END
40 IF A$="" THEN PRINT B$:B$="":GOTO 20
50 PO=0
60 DO UNTIL PO+LEN(B$)>41
70 XO=PO
80 PO=INSTR(A$," ",PO+1)
90 IF PO=0 THEN EXIT
100 LOOP
110 IF PO=0 AND LEN(A$)>0 THEN XO=40-LEN (B$)
120 B$=B$+LEFT$(A$,XO)
130 A$=MID$(A$,XO+1,LEN(A$))
140 IF PO=0 AND LEN(A$)=0 THEN GOTO 20
150 PRINT B$:B$=""
160 GOTO 50
500 DATA "DIES IST EIN KOMFORTABLES AUSGABEPROGRAMM FUER TEXTE.
SIE SEHEN, "
510 DATA "KEIN TEXT WIRD GETRENNT "
520 DATA "ODER ZERSTUECKELT AUSGEGEBEN. "
530 DATA "ES KOENNEN SEHR LEICHT ","","ABSAETZE ERZEUGT
WERDEN."
540 DATA "AUCH ","","","LEERZEILEN SIND MOEGLICH "
700 DATA "-1"
```

11.2 Der PRINT USING-Befehl

Syntax:

PRINT USING "Format "; Variable/String/Text, Variable,...

Mit diesem Befehl kann sehr einfach eine formatierte Ausgabe realisiert werden.

Wie Sie der Syntax entnehmen können, benötigt PRINT USING eine Formatanweisung, die in Anführungszeichen eingeschlossen wird. Es muß ein Semikolon folgen, welches von den auszugebenden Variablen (String- oder numerische Variablen) gefolgt wird. Es können aber auch Texte ausgegeben werden, wenn sie ebenfalls in Anführungszeichen stehen.

Der PRINT USING-Befehl kennt mehrere Formatierungszeichen. Ein Zeichen gilt sowohl für die Ausgabe von Zahlen, als auch für Texte. Einige andere gelten dann jeweils nur für die formatierte Zahl- oder Textausgabe. Wir fangen mit dem Zeichen an, welches sowohl für Zahlen als auch für Texte gilt.

11.2.1 Die Raute (#) bei der Ausgabe von Zahlen mit PRINT USING

Geben Sie vor dem Weiterlesen bitte erst das nachstehende kleine Programm ein:

```
20 INPUT "FORMAT ";FO$
30 INPUT "ZAHL " ;A
40 IF A = -1 THEN END
50 PRINT USING FO$;A
60 GOTO 20
```

Wie Sie der Zeile 50 des Programms entnehmen können, kann die Formatanweisung auch durch eine Stringvariable, hier FO$ genannt, übergeben werden. Das Programm läßt sich jederzeit beenden, indem man die Frage »ZAHL ?« mit -1 beantwortet.

Starten Sie das Programm. Beantworten Sie die Frage nach dem Format mit #### (vier Rauten) und die Frage nach der Zahl mit 1234. Es erscheint die Zahl 1234 linksbündig in der nächsten Zeile. Die Formatfrage beantworten Sie bitte mit RETURN, die Frage nach der Zahl mit 123. Dieses Spiel wiederholen Sie, bis Sie nur noch die Zahl 1 eingegeben haben.

Sie können nun den Unterschied zum PRINT-Befehl sehr deutlich erkennen. Gibt der PRINT-Befehl Zeichen und Zahlen immer am Anfang einer Zeile aus, soweit nicht anders durch Steuerzeichen bestimmt, so werden die Zahlen bei PRINT USING »dezimalgerecht« ausgegeben, solange immer das gleiche Format gewählt wird. D. h. Einerstellen stehen untereinander, Zehnerstellen stehen untereinander usw.

Behalten Sie das Format bei und geben Sie als Zahl 12345 ein. Es erscheinen vier Sternchen auf dem Bildschirm, wovon jedes eine # im Format repräsentiert. Für jede in einer Zahl enthaltene Ziffer muß eine # im Formatierungsfeld stehen. Ist eine Zahl länger als im Format angegeben, so erscheinen, der Anzahl von Rauten (#) entsprechend, Sternchen auf dem Bildschirm. Der Programmierer hat also dafür zu sorgen, daß nicht mehr Ziffern ausgegeben werden sollen, als # im Format vorhanden sind. Geben wir bei unverändertem Format einen Dezimalbruch ein und beantworten die Frage »ZAHL ?« mit 123.5 . Das Ergebnis ist 124. Der Rechner hat also erst die Zahl aufgerundet und dann ausgegeben.

Geben Sie 123.4 ein. In diesem Fall hat der Rechner abgerundet. Diese Tatsache macht diesen Befehl sehr komfortabel, wird einem doch zusätzliche Programmierarbeit abgenommen.

ACHTUNG: Der Plus/4 verarbeitet lediglich Zahlen mit nicht mehr als neun Ziffern. Geben Sie zum Beispiel 11 Rauten und dann eine 11stellige Zahl ein, so erscheinen die letzten Ziffern als Null! Wie Sie mit dem PRINT USING größere Zahlen darstellen können, erfahren Sie bei den Exponentialzeichen.

11.2.2 Der Dezimalpunkt bei PRINT USING

Jetzt wird das Format geändert:

###.## (dreimal Raute, Punkt, zweimal Raute)

Als Zahl geben wir ein:

123.345

Das Ergebnis ist:

123.35

Dem Dezimalpunkt dürfen also mehr Ziffern folgen, als Formatzeichen vom Typ # angegeben wurden. Es wird auch in diesem Fall auf- oder abgerundet. Folgen dem Dezimalpunkt keine weiteren Ziffern, so werden, der Anzahl der Rauten entsprechend, Nullen ausgegeben. Geben Sie einmal willkürliche Zahlen ein. Sie werden feststellen, daß die Dezimalpunkte immer untereinander stehen. Was passiert aber bei der Eingabe einer negativen Zahl? Probieren wir es bei unverändertem Format aus.

-220.14 ergibt sechs Sternchen auf dem Bildschirm; drei für den ganzzahligen Teil, eins für den Dezimalpunkt und zwei für den Dezimalbruch.

-22.14 wird angezeigt. Das Minuszeichen beansprucht also eine Raute. Dies kann umgangen werden.

11.2.3 Plus- und Minuszeichen bei PRINT USING

Geben wir nun das Format mit -###.## (Minuszeichen, dreimal Raute, Dezimalpunkt, zweimal Raute an, und dann die Zahl 123.45 ein. In der nächsten Zeile erscheint in der zweiten Spalte 123.45. Es wurde also ein Zeichen freigelassen. Dieses Zeichen ist für das im Formatfeld angegebene Minuszeichen reserviert. Geben wir nämlich -123.45 ein, so erscheint das Minuszeichen in der ersten Spalte der Zeile.

Ähnlich verhält es sich bei dem Pluszeichen. Ein Beispiel: Das Format +###.## und dann als Zahl 123 ergibt bei der Ausgabe +123.00. Es wird jeder positiven Zahl ein Pluszeichen vorangestellt. Ein bei der Eingabe der Zahl ebenfalls eingegebenes Pluszeichen wird nicht mit angezeigt. +123 ergibt ebenfalls »nur« +123.00.

Auch negative Zahlen können in diesem Format dargestellt werden. Beispiel: -123 ergibt -123.00. Plus- und Minuszeichen können aber auch dem Format nachgestellt werden. Hierfür ein Beispiel: Das Format ###.##+ ergibt bei der Zahl 123 die Ausgabe 123.00+. Die Zahl -123 ergibt 123.00-. Sie sehen, bei der Ausgabe wird Plus oder Minus nachgestellt.

ACHTUNG: Ein Format wie -###.##- ist VERBOTEN! Trifft der Rechner auf dieses Format, so wird das Programm mit »SYNTAX ERROR IN ...« abgebrochen.

11.2.4 Die Exponentialzeichen bei PRINT USING

Trifft der Rechner bei PRINT USING auf vier Exponentialzeichen (^^^^), dann zeigt er die folgende Zahl im wissenschaftlichen Format an. Für all diejenigen, denen dieses Format nicht geläufig ist, folgt eine kurze Erklärung dieses Formats.

Im Dezimalsystem können Zahlen verschieden dargestellt werden. Man kann die Zahl 1000 zum Beispiel so ausdrücken, oder auch als 1*10^3, sprich eins mal zehn hoch drei. Die Drei ist in diesem Fall der Exponent, die Zahl zehn die sogenannte Basis. 10^3, sprich zehn hoch drei, kann auch durch 10*10*10 ausgedrückt werden. Das Ergebnis dieser Multiplikation ergibt nun 1000. Wie wird die Zahl 2000 in Exponentialform ausgedrückt? Teilen wir 2000 durch 10, erhalten wir als Ergebnis 200. Nun wird noch zweimal durch 10 geteilt und wir haben das Ergebnis 2. Da wir insgesamt drei-

mal durch 10 geteilt haben, steht im Exponenten der Basis 10 die Zahl drei. Also lautet das richtige Ergebnis: 2*10^3.

Die Zahl 2300 läßt sich auch als 2.3*10^3 und die Zahl 2340 als 2.34*10^3 beschreiben. Der Plus/4 drückt diese Zahl als 2.34E+3 aus. Das »E« steht hier für die Zahl 10.

Das Ganze läßt sich aber auch umdrehen. So ergibt 0.003 im wissenschaftlichen Format 3+10^-3, sprich drei mal zehn hoch minus drei. Das Minuszeichen heißt hier nichts anderes, als daß man die Zahl drei dreimal hintereinander durch 10 dividieren muß, um die im wissenschaftlichen Format dargestellte Zahl in dezimaler Schreibweise zu erhalten. Der Plus/4 stellt diese Zahl als 3E-3 dar. Dieses Format ist nur bei sehr großen oder sehr kleinen Zahlen sinnvoll, da es bei diesen beim Ergebnis oft nicht auf den Inhalt der letzten Stelle ankommt.

Machen wir nun mit dem PRINT-USING-Befehl weiter. Als Format geben wir #^^^^ ein. Die Zahl soll 1000.22 sein. Das Ergebnis lautet: 1E+03. Sie können feststellen, daß der Rechner von sich aus gerundet hat, da die 0,22 nicht mit ausgegeben wurde. Er hat die Zahl aber nur für die Ausgabe gerundet, in der Variablen ist sie noch so, wie wir sie eingegeben haben, vorhanden.

Die Zahl 1.5 ergibt 15E-1. Das Format läßt zwei Zeichen vor und, bedingt durch die Exponentialzeichen, drei Zeichen nach diesen (Vorzeichen und zwei Ziffern) zu.

Mit diesem Format sind Sie in der Lage, Zahlen mit der maximalen Stellenzahl, die der Rechner noch verarbeiten kann, darzustellen. Diese Zahlen sind aber ungenau. Es können Zahlen von 1E+38 bis 1E-38 mit dem Plus/4-BASIC verarbeitet werden. Zahlen die größer sind, verursachen ein »?OVERFLOW ERROR«, Zahlen die kleiner sind, sind für den Rechner Null. Der Exponent wird stets, auch bei PRINT USING, mit seinem Vorzeichen dargestellt.

Ein anderes Format: ###.##^^^^ und die Zahl 123456 ergibt 123.46E+02. Multiplizieren Sie einmal diese Zahl mit 10^2 = 100, so erhalten Sie 12346. Sie sehen, auch jetzt wird gerundet. Alles oben Gesagte gilt weiterhin.

ACHTUNG: Daß man die vier Exponentenzeichen sowohl vor als auch nach der Raute plazieren kann, stimmt nicht! Es kommt zwar keine Fehlermeldung, dafür sieht aber die Ausgabe recht kurios aus.

11.2.5 Das Dollarzeichen bei PRINT USING

Dieses Zeichen wird, wenn es in einem Formatstring steht, mit auf dem Bildschirm ausgegeben. Es ermöglicht einem nicht nur die Angabe der Währung (Dollar), sondern einfach die formatierte Ausgabe von Hexadezimalzahlen auf dem Bildschirm.

Wählen wir nun das folgende Format: $#### und geben dann die Zahl 12 ein. Das Dollarzeichen steht in der ersten Spalte der nächsten Zeile, die Zahl 12 steht in den Bildschirmspalten vier und fünf. Nehmen Sie nun das Format: ##$## und wiederum die Zahl 12. Das Ergebnis sieht nun ganz anders aus. Die ersten beiden Spalten sind leer, dann folgt das Dollarzeichen und die Zahl 12.

Merken wir uns: Steht das Dollarzeichen vor der ersten Raute, so erscheint es, der Anzahl von Rauten vor einem Dezimalpunkt entsprechend, vor der auszugebenden Zahl. Steht es irgendwo zwischen Rauten und einem Dezimalpunkt, so erscheint es direkt vor der darzustellenden Zahl.

11.2.6 Das Komma bei PRINT USING

Wird ein Komma in einem Formatstring verwendet, so erscheint es an der Stelle auf dem Bildschirm, an der es auch im Formatstring steht. Steht es an erster Stelle, so wird es ebenfalls mit ausgegeben.

Die Demonstration der nachfolgenden Beispiele ist mit unserem Programm leider nicht mehr möglich, da der INPUT-Befehl eingegebene Kommata bekanntlich nicht in Strings übernimmt. Geben Sie daher bitte im Direktmodus ein:

PRINT USING "##,##";1234

Das Ergebnis: 12.34

Als nächstes folgt:

PRINT USING ",##,##.##,##";1234.5678

Die Ausgabe: 12,34.56,78

Das Komma darf also, wie man im zweiten Beispiel sehr gut sehen kann, nicht mit dem Dezimalpunkt gleichgesetzt werden. Sie haben sicher auch festgestellt, daß das oben Angeführte auch wirklich stimmt. Natürlich gilt auch weiterhin das Gesagte zum Runden der Zahlen.

11.2.7 Formatierte Textausgabe mit PRINT USING

Für die Textausgabe stehen weniger Formatzeichen als bei der formatierten Zahlenausgabe zur Verfügung. Das ist aber kein Nachteil, kann man doch durch die richtige Kombination dieser Formatierungszeichen nahezu jeden Wunsch der Darstellung auf dem Bildschirm erfüllen. Um PRINT USING bei der Textausgabe genauso leicht wie bei der formatierten Ausgabe von Zahlen kennenzulernen, müssen wir unser Programm etwas ändern:

```
30 INPUT "TEXT ";A$
40 IF A$ = "-1" THEN END
50 PRINT USING FO$;A$
```

Die Vorarbeiten sind geleistet, und wir können mit dem ersten Formatierungszeichen für die Textausgabe beginnen.

11.2.8 Die Raute (#) bei der Textausgabe mit PRINT USING

Wie immer fangen wir mit einer Formatangabe an. Diesmal nehmen wir: #### (viermal Raute). Der Text soll heißen: PLUS.

Den Unterschied zur Zahlenformatierung werden Sie sicher sofort erkannt haben. Der Text wird linksbündig ausgegeben. Geben wir nun als Text ein: PLUS/4.

Es erscheinen keine Sternchen, sollte der Text auch mehr Zeichen umfassen als Rauten bei der Formatierung angegeben sind. Der Text wird um die rechten Zeichen gekürzt, die nicht mehr in das definierte Format passen. Machen Sie auch folgendes:

Format: ######.###### (sechs Rauten, Dezimalpunkt, sechs Rauten)

Text: DER COMPUTER

Auf dem Bildschirm sehen Sie, daß Zeichen wie der Dezimalpunkt bei der Ausgabe ignoriert werden.

11.2.9 Das Gleichheitszeichen bei PRINT USING

Wir haben oben festgestellt, daß die Textausgabe in der ersten Spalte beginnt und so viele Zeichen, wie Rauten bei dem Formatstring vorhanden sind, umfaßt. Geben Sie bitte als Format ein: = und dann 39 Rauten. Als Text nehmen wir: DER PLUS/4 COMPUTER. So einfach ist es, Texte in die Mitte des Bildschirms zu bringen. Das Gleichheitszeichen (=) bewirkt, daß der Text innerhalb der Formatzeichen zentriert wird.

Ein anderer Fall mit Format =###### (Gleichheitszeichen, sechs Rauten) und Text PLUS/4. Der Text erscheint rechtsbündig auf dem Bildschirm. Auch dies ist mit diesem Formatbefehl, allerdings nur bedingt und auch etwas umständlich, möglich. Wäre hier der Text länger als sieben Zeichen, so würden die rechten Zeichen nach dem siebten abgeschnitten, d. h. nicht mitangezeigt.

An welcher Stelle des Formatstrings das Gleichheitszeichen steht, spielt genausowenig eine Rolle wie die Anzahl. Bedenken Sie aber, daß das Gleichheitszeichen ebenfalls einen Platz für ein Zeichen repräsentiert und daß mindestens eine Raute im Formatstring vorhanden sein muß.

11.2.10 Das Größer-als-Zeichen (>) bei PRINT USING

Wir fangen gleich mit dem Format an:

>####### (Größer-als-Zeichen, siebenmal Raute)

Der Text: PLUS/4

Sie sehen, der Text erscheint in einem Abstand von zwei Bildschirmspalten vom linken Rand des Schirms. Belassen wir es einmal bei diesem Format und geben als neuen Text: DER PLUS/4 ein. Es wird nur DER PLUS angezeigt, die anderen Buchstaben wurden wieder rechts abgeschnitten. Über die Anzahl und die Positionierung dieses Steuerzeichens gilt das bei der Beschreibung des Gleichheitszeichens Gesagte.

11.3 Erweiterte Ausgabemöglichkeiten mit PRINT USING

Bis jetzt wurden Ihnen die verschiedenen Formatierungszeichen dieses Befehls erläutert. Nun möchten wir Ihnen noch einige Hilfestellungen bei der Verwendung von PRINT USING geben. Trifft der Befehl auf mehrere auszugebende Texte oder Zahlen, so wird das vorher definierte Format beibehalten. Ein Beispiel:

```
PRINT USING "####.##";1234.56," ",7890.125
```

ergibt:

```
1234.56     789.13
```

Sie sehen, die beiden Zahlen erscheinen im gleichen Format und sind durch sieben Leerzeichen getrennt. Das siebte Leerzeichen wird durch den Dezimalpunkt bewirkt. Trifft der Befehl bei der Ausgabe eines Textes auf ein Formatierungszeichen, welches nur der Formatierung von Zahlen dient, verarbeitet er es wie die Raute (#). Genauso verhält er sich bei der Ausgabe von Zahlen und bei Auftreten von Formatierungsanweisungen für Texte.

```
PRINT USING "=###.##";1234.56
```

ergibt:

```
1234.56
```

Obwohl eigentlich eine Raute fehlt und deshalb Sternchen auf dem Bildschirm erscheinen müßten, wird die Zahl richtig dargestellt. Das Gleichheitszeichen wurde wie eine Raute verarbeitet.

Auch die Ausgabe von Begleittexten ist unkompliziert möglich.

```
PRINT USING " SIN ##.####";SIN(123)
```

ergibt:

```
SIN -0.4599
```

Die Zeichen werden, ihrer Stelle im Formatstring entsprechend, auf den Bildschirm gebracht.

Sollen nach PRINT USING in der gleichen Zeile Zeichen durch einen PRINT-Befehl ausgegeben werden, muß dem letzten auszugebenden Wert von PRINT USING ein Semikolon (;) folgen. Ein Beispiel:

```
PRINT USING "######";"PLUS/4";:PRINT " COMPUTER"
```

Ein Komma als Steuerzeichen ist nach PRINT USING nicht erlaubt, da der Rechner nach diesem Zeichen einen weiteren Text oder eine Zahl für die formatierte Ausgabe erwartet. Trifft er auf ein Komma, ohne daß es von auszugebenden Zeichen gefolgt wird, bricht er mit einer Fehlermeldung die weitere Bearbeitung ab.

Aber der PRINT-USING-Befehl bietet noch mehr. Sie können einige Formatzeichen dieses Befehles selbst umbenennen. Wie es gemacht wird, erfahren Sie im nächsten Kapitel.

11.4 Der PUDEF-Befehl

Syntax:

PUDEF "Vier-Zeichen-String"

Der eine oder andere unter Ihnen findet ein Komma an Stelle eines Dezimalpunktes vielleicht übersichtlicher, andere möchten statt des Dollarzeichens lieber ein F für Franc ausgeben lassen, ohne auf die Vorteile, die das Dollarzeichen bei der Positionierung bietet, zu verzichten. Für diese Problemlösungen bietet das BASIC des Plus/4 den **PUDEF**-Befehl. Dieser Befehl wird von einem vier Zeichen langen String gefolgt. Nach dem Einschalten des Rechners oder nach einem RUN sieht der String folgendermaßen aus:

```
" ,.$"
```

Diesen String können Sie sich nicht anzeigen lassen; der Rechner behandelt PRINT USING aber so, als sei dieser String mit PUDEF so definiert worden.

Auf die Bedeutung dieser Zeichen wurde schon bei der Behandlung des PRINT-USING-Befehls ausführlich eingegangen. Sollte Ihnen die Bedeutung dieser Zeichen noch unklar sein, lesen Sie bitte dort noch einmal nach.

Sehen wir uns diesen String einmal genauer an: Das erste Zeichen ist ein Leerzeichen, das zweite ein Komma, das dritte ein Dezimalpunkt und das vierte das Dollarzeichen. Mit dem Befehl PUDEF kann nun jedes dieser Zeichen ein anderes Aussehen erhalten. Geben Sie bitte folgendes ein:

```
PUDEF"#.,F"
PRINT USING "#$###,###.###";123456.78
```

Sie erhalten folgende Ausgabe:

#F123.456,78

Wie kommt diese Ausgabe zustande? Der String wurde mit PUDEF umgeformt. Als erstes Zeichen wurde statt eines Leerzeichens die Raute aufgenommen. Soll der Rechner bei der Ausgabe von Zahlen Leerzeichen setzen, nimmt er statt dessen Rauten. Dies gilt übrigens nur für die Zahlenausgabe. Bei Strings werden auch weiterhin Leerzeichen ausgegeben. Versuchen Sie's!

Als zweites Zeichen wurde ein Dezimalpunkt anstatt des Kommas definiert, welches ja ursprünglich in diesem String enthalten war. Dann wurde der Dezimalpunkt durch ein Komma ersetzt und das Dollarzeichen durch den Buchstaben F. Jedesmal, wenn PRINT USING bei der Zahlenausgabe Leerzeichen, Kommata, Dezimalpunkte oder Dollarzeichen ausgeben soll, werden jetzt die Ersatzzeichen: Raute(#), Dezimalpunkt (.), Komma (,) und der Buchstabe F, ausgegeben. Die zu ändernden Zeichen können PUDEF auch durch eine Stringvariable übergeben werden. Stellen Sie aber sicher, daß diese nur vier Zeichen lang ist.

ACHTUNG: Beim Plus/4 wird das Dollarzeichen nur dann gegen ein anderes definiertes Zeichen ersetzt, wenn ihm im Formatstring von PRINT USING mindestens eine Raute vorangestellt wird. Wird das mißachtet, erscheint es weiterhin auf dem Bildschirm.

Sie brauchen nicht immer den gesamten String neu zu definieren. Soll zum Beispiel das Leerzeichen durch ein Sternchen ersetzt werden, so reicht:

```
PUDEF "*"
```

Alle anderen Zeichen, auch die eventuell durch PUDEF geänderten, bleiben weiterhin erhalten. Sie müssen nur darauf achten, daß der String nicht mehr als vier Zeichen enthält, da sich sonst der Rechner mit der Fehlermeldung »?ILLEGAL QUANTITY ERROR IN...« meldet und ein laufendes Programm abbricht.

Dies sollte zu dem Komplex PRINT USING und PUDEF reichen. Wir empfehlen Ihnen, mit diesen beiden Befehlen noch ein wenig weiter zu experimentieren. Sollten diese beiden Befehle auch etwas komplex wirken, so sind sie in Programmen sehr oft gut einsetzbar.

11.5. Die Positionierung des Cursors

Das 3.5-BASIC des Plus/4 bietet auch Befehle zur direkten Positionierung des Cursors an. Mit ihnen kann der Cursor vor einer Ausgabe auf die gewünschte Stelle des Bildschirms gesetzt werden.

11.5.1 Die Positionierung des Cursors mit dem CHAR-Befehl

Syntax:

CHAR , Spalte, Zeile, ""

Der Plus/4 beginnt die Zeilen- und Spaltenzählung mit Null. Daher geht der zulässige Bereich für die Zeilen von 0 bis 24 und für die Spalten von 0 bis 39.

Mit diesem Befehl ist die Positionierung des Cursors nicht nur im Grafik-, sondern auch im Textmodus möglich.

CHAR, 0,0, "" setzt den Cursor in die linke obere Ecke.

Eine nachfolgende Ausgabe mit **PRINT** erfolgt ab dieser Stelle. Der Cursor kann überall frei auf dem Bildschirm positioniert werden, auch dann, wenn sich dort bereits Zeichen befinden. CHAR kann auch von einem Text gefolgt werden.

CHAR, 5, 4, "DER COMPUTER PLUS/4"

gibt den Text beginnend in der sechsten Spalte der fünften Zeile aus. Im Gegensatz zum Grafikmodus werden auch alle Bildschirmsteuerzeichen, die ein String enthalten können, ausgeführt. Man kann den Text also auch revers oder blinkend in verschiedenen Farben ausgeben lassen. Mehr zum **CHAR**-Befehl erfahren Sie im Kapitel über die Grafik.

11.5.2 Der SPC(X)-Befehl

Syntax:

PRINT SPC (Zahl oder numerische Variable)

Dieser Befehl setzt den Cursor spaltenweise weiter. Die Anzahl der Spalten, um die der Cursor weitergesetzt wird, muß in Klammern hinter SPC angegeben werden. Die größte Zahl, die in diesem Argument stehen darf, ist 255; die kleinste 0. Zahlen, die größer oder kleiner sind, verursachen eine Fehlermeldung. Auch Dezimalbrüche wie

12.45 oder 0.12 sind erlaubt, der Cursor wird dann aber nur um den ganzzahligen Teil weitergesetzt. Vor **SPC(X)** muß immer ein PRINT-Befehl stehen. Ein Beispiel zu SPC(X):

```
10 PRINT CHR$(147)
20 FOR I=1 TO 3
30 PRINT SPC(5) "PLUS/4";
40 NEXT I
```

Sie sehen, das Wort PLUS/4 erscheint nach dem Programmstart fünf Spalten vom Rand entfernt; das zweite Wort fünf Zeichen vom ersten Wort, usw.. Dieser Cursorsprung wurde durch SPC(5) verursacht. Möchten Sie zum Beispiel in einem Programm die Ausgabe von Zeichen immer um eine flexible Spaltenanzahl von den vorherigen Ausgaben trennen, so ist dies mit SPC(X) sehr leicht zu erreichen.

11.5.3 Der TAB-Befehl

Syntax:

PRINT TAB (Zahl oder numerische Variable)

Die Zahl des Arguments darf nicht kleiner als 0 und nicht größer als 255 sein, da sonst eine Fehlermeldung erfolgt. Das Argument kann auch hier ein Dezimalbruch sein, wobei wieder nur der ganzzahlige Teil verarbeitet wird.

TAB (X) muß immer ein PRINT-Befehl vorangestellt sein. Wie Sie schon wissen, können in jeder Zeile des Bildschirmes 40 Zeichen angezeigt werden. Mit TAB ist es möglich zu bestimmen, in welcher Bildschirmspalte die Ausgabe von Zeichen beginnen soll.

Geben Sie bitte das nachstehende kleine Programm ein:

```
10 PRINT CHR$(147)
20 FOR I=1 TO 6
30 PRINT TAB (16) "PLUS/4";
40 NEXT I
```

Nach dem Ablauf dieses Programms sehen Sie den Unterschied zu PRINT SPC(X) sehr deutlich. Die erste Ausgabe erfolgte in der 16. Spalte der ersten Bildschirmzeile, die nächsten drei Ausgaben unmittelbar dahinter. Erst die fünfte Ausgabe erfolgte wieder in der 16. Spalte der zweiten Bildschirmzeile.

Trifft der Rechner auf PRINT TAB(X), so überprüft er, in welcher Spalte der aktuellen Bildschirmzeile der Cursor steht. Ist die Spaltennummer größer als die Zahl des

Arguments, so bleibt der Cursor in der derzeitigen Position stehen. Ist die Zahl kleiner, wird der Cursor entsprechend gesetzt. Der Rechner zählt übrigens die Spalten von 0 bis 39. Ist das Argument größer als 39, wird der Cursor auf jeden Fall bewegt.

11.5.4 Der POS(X)-Befehl

Syntax:

PRINT POS (Dummy Argument)

Wenn Sie vor einer weiteren Ausgabe die Cursorposition in der aktuellen Bildschirmzeile erfahren möchten, so ist dies mit POS(X) möglich. Für die direkte Ausgabe der Position auf dem Bildschirm reicht:

PRINT POS(X)

Soll aber auf die Position reagiert werden, so kann sie auch einer Variablen übergeben werden, zum Beispiel so:

A= POS(X)

In der Variablen A ist nun die derzeitige Position gespeichert und kann für ein anderes Positionieren des Cursors verwandt werden. Denken Sie auch hier daran, daß der Plus/4 die Zählung der Bildschirmspalten mit Null beginnt.

11.5.5 Positionierung mit Bildschirmsteuerzeichen

Im Direktmodus können Sie mit den Cursortasten jeden Punkt des Bildschirms erreichen. Dies klappt auch im Programmodus, wobei hier die Steuerung des Cursors nicht durch die Tasten, sondern durch Steuerzeichen, die in Anführungszeichen stehen müssen, erfolgt. Zur Demonstration ein Beispiel:

```
10 PRINT CHR$(147)
20 PRINT "DER";
30 Die Eingabe dieser Zeile wird im Text beschrieben
40 PRINT "COMPUTER"
```

Zeile 30: Geben Sie PRINT ein. Nun drücken Sie bitte achtmal die CRSR-DOWN-Taste. Bei jedem Druck dieser Taste muß ein reverses Q auf dem Bildschirm erscheinen; wenn nicht, haben Sie die Anführungszeichen vergessen. Es folgen noch Anführungszeichen und ein Semikolon.

Sie sehen, zuerst erscheint das Wort DER und acht Bildschirmzeilen tiefer das Wort COMPUTER. Der Rechner behandelt in Anführungszeichen eingeschlossene Steuerzeichen genauso, als seien sie direkt über die Tastatur eingegeben.

Das reverse Q erhält man auch, wenn man den Reversemodus mit CONTROL und RVS ON einschaltet und dann das Q eingibt. Das Ergebnis ist das gleiche, nur ist das Eingeben umständlicher. Der Plus/4 verfügt über eine große Anzahl derartiger Steuerzeichen. So sind mit diesen Zeichen auch die Zeichenfarben wählbar oder der Reversemodus ein- oder abschaltbar. Auch FLASH ON oder FLASH OFF ist so leicht zu programmieren. Um Steuerzeichen beim Eingeben darzustellen, muß vorher der Anführungszeichenmodus durch die Eingabe eines Anführungszeichens eingeschaltet werden. Drücken Sie nach einem Anführungszeichen zum Beispiel die SHIFT- und die CLEAR/HOME-Taste, erscheint ein reverses Herz. Trifft der Rechner auf diese Zeichen, wird der Bildschirm gelöscht. Soll der Cursor in die erste Bildschirmzeile gesetzt werden, so reicht:

```
PRINT " "
    I
    hier Taste CLEAR/HOME drücken
```

Soll die nächste Ausgabe aber auch in der ersten Zeile erfolgen, so muß noch ein Semikolon folgen.

Die Steuerung des Cursors mit den Sonderzeichen trägt nicht gerade zur Übersichtlichkeit eines Programms bei, vor allem dann, wenn mehrere Steuerzeichen hintereinander stehen. Es geht aber auch anders.

11.5.6 Die Ausgabe mit CHR$

Syntax:

PRINT CHR$ (Zahl oder Variable)

oder

A$ = CHR$ (Zahl oder Variable)

Wie Sie sicher wissen, kann ein Computer nur Zahlen verarbeiten. Mit Buchstaben kann er nichts anfangen. Darum wandelt er jeden Buchstaben, den Sie eingeben, in eine Zahl um.

Der Plus/4 besitzt, wie wir eben gesehen haben, auch diverse Bildschirmsteuerzeichen. Dem Zeichen für das Bildschirmlöschen ist zum Beispiel die Zahl 147 zugeordnet,

dem REVERSE ON die Zahl 10. Die Art und Weise, in der Zahlen den Buchstaben und Steuerzeichen zugeordnet sind, nennt man ASCII-Code. Dieser Code wird von den meisten Rechnern benutzt, wobei Abweichungen schon zwangsläufig durch die verwendeten Steuercodes auftreten. Nicht jeder Rechner verfügt über so vielfältige Möglichkeiten der Bildschirmgestaltung wie unser Plus/4. Die kleinste Zahl, die verwendet werden darf, ist Null, die größte 255.

Geben Sie bitte folgendes ein:

```
PRINT "A"
```

Das Ergebnis ist klar, es erscheint ein »A« auf dem Bildschirm. Das gleiche können wir aber auch durch:

```
PRINT CHR$(65)
```

erreichen. CHR$ teilt dem Rechner mit, daß er die in Klammern folgende Zahl als Zeichen auf dem Bildschirm ausgeben soll. Bei der Ausgabe muß CHR$ ein PRINT-Befehl vorangestellt werden. Der auszugebende Code ist in Klammern einzuschließen, da der Rechner sonst mit einer Fehlermeldung »aussteigt«.

Einige Steuerzeichen sind auf normalem Wege durch Bildschirmsteuerzeichen nicht erreichbar. In diesem Fall muß das Steuerzeichen mit »PRINT CHR$ (Codezahl)« ausgegeben werden. Ein weiteres Beispiel:

```
PRINT CHR$(14)
```

Der Code 14 schaltet also den Groß-/Kleinschreibmodus ein. Den Großschriftmodus können Sie jetzt durch Drücken der SHIFT- und COMMODORE-Taste wieder einschalten. In einem Programm macht man dies aber sinnvollerweise mit:

```
PRINT CHR$(142)
```

Die Umschaltung durch die Tastatur kann aber auch ganz unterbunden werden:

```
PRINT CHR$(8)
```

Versuchen Sie jetzt einmal mit der SHIFT- und COMMODORE-Taste den Modus zu wechseln. Einschalten können Sie diese Tastenkombination wieder mit:

```
PRINT CHR$(9)
```

Mit CHR$ können auch Zeichen an eine Stringvariable übergeben werden. Das sieht dann so aus:

A$=CHR$(X)

Das Zeichen, dessen Code in X steht, wird als ASCII Zeichen A$ übergeben. Dies soll an dieser Stelle über CHR$ reichen.

11.5.7 Der ASC-Befehl

Syntax:

PRINT ASC ("Zeichen")

oder

X = ASC ("Zeichen")

Obwohl dieser Befehl nicht so gut in dieses Kapitel paßt, soll er hier erklärt werden, da er die Umkehrfunktion von CHR$ darstellt. Wie Sie der Syntax entnehmen können, verarbeitet dieser Befehl Zeichen, und nicht wie der CHR$-Befehl Zahlen.

Geben Sie einmal ein:

PRINT ASC("A")

Sie erhalten die Zahl 65 auf dem Bildschirm. Die Zahl 65 stellt den ASCII-Code des Buchstaben A dar. Das Zeichen, dessen ASCII-Code ermittelt werden soll, muß in Klammern stehen und durch Anführungszeichen eingeschlossen sein. Es können auch mehrere Zeichen zwischen den Anführungszeichen stehen, es wird dann aber nur der Code des ersten Zeichens ausgegeben, die anderen Zeichen werden ignoriert.

Auch folgendes ist möglich:

X=ASC("A") oder X=ASC(A$) oder PRINT ASC(A$)

Bitte beachten, daß Stringvariablen nicht in Anführungszeichen stehen dürfen, sonst wird der Code für den ersten Buchstaben des Stringvariablennamens ausgegeben.

11.6 Bildschirmausgabe mit dem POKE-Befehl

Syntax:

POKE Speicheradresse, Inhalt

Der Befehl wird an dieser Stelle nicht weiter beschrieben, da dies im Kapitel 17 erfolgt. Sollten Sie diesen Befehl und seine Tücken noch nicht kennen, so lesen Sie bitte erst einmal dort nach.

ACHTUNG: Übernehmen Sie bitte alle Zahlenwerte im Laufe dieses Kapitels. Es kann sonst passieren, daß der Rechner »abstürzt«. Sie können ihn dann oft nur noch durch einen **RESET** neu starten.

11.6.1 Der Bildschirmaufbau im Textmodus

Der Bildschirm des Plus/4 besteht aus 25 Zeilen mal 40 Spalten. Es könnten also insgesamt 1000 Zeichen auf einmal angezeigt werden. Jedes dieser Zeichen würde einen Platz des Bildschirmes belegen. Wie Sie wissen, findet man in jedem Rechner, der programmierbar sein soll, RAM-Speicher (RAM = Schreib-Lesespeicher). Ein Teil dieses RAMs nennt man beim Plus/4 Bildschirmspeicher, ein anderer Teil wird Farbspeicher genannt. Wie Sie wissen, sind die einzelnen Speicherplätze eines Rechners durchnumeriert; man spricht von Adressen. Die jeweiligen Anfangs- und Endadressen vom Bildschirmspeicher und dem Farbspeicher lauten:

BILDSCHIRMSPEICHER:	Dezimal 3072 - 4071	Hex $OCOO - $OFE7
FARBSPEICHER:	Dezimal 2048 - 3071	Hex $O8OO - $0BFF

Die Numerierung beginnt in der linken oberen Ecke des Bildschirms. Dies ist der Ursprung. Die Zählung erfolgt von links nach rechts. Ist eine Zeile zu Ende, fährt man mit der Zählung in der linken Spalte der nächsten Zeile fort. Ein Beispiel für die Zählung. Welche Adresse hat die Speicherzelle in der 3. Zeile und 25. Bildschirmspalte? Rechnen wir einmal:

Der Ursprung in der linken oberen Ecke hat die Adresse 3072. Addieren wir nun hierzu 40 (jede Bildschirmzeile besteht ja aus 40 Zeichen), so haben wir den Anfang des Speichers der zweiten Zeile. Zum Ergebnis werden noch 40 addiert. Wir erhalten 3152. Diese Zahl repräsentiert die Anfangsadresse im Bildschirmspeicher für die 3. Zeile. Hierzu muß noch 24 für die 25. Bildschirmspalte addiert werden. Wir erhalten als Ergebnis die Zahl 3176. Da 3152 schon die Adresse für die erste Bildschirmspalte der 3. Zeile ist, dürfen nur noch 24 addiert werden.

Aus der obigen Berechnung läßt sich sehr einfach eine Formel ableiten.

Bildschirmadresse = Ursprung + 40 * (Zeile - 1) + Spalte - 1

Rechnen Sie mit dieser Formel einmal unser Beispiel nach. In jeder dieser 1000 Speicherzellen steht ein Wert. Jeder dieser Werte repräsentiert das Zeichen, welches an der entsprechenden Stelle abgebildet wird. In jeder Speicherzelle können Werte von 0 bis 255 stehen.

Erst einmal ein Beispiel. Hierzu nehmen wir die oben ausgerechnete Speicherzelle 3176. Löschen Sie nun bitte den Bildschirm und geben dann folgendes ein:

POKE 3176,1

Es erscheint an der vorberechneten Stelle auf dem Bildschirm ein großes A. Sie sehen, die Werte für den Bildschirmspeicher sind n i c h t gleich dem ASCII-Code. Auch die reverse Darstellung läßt sich mit POKE erreichen. Sie brauchen nur zu dem Wert, in diesem Fall eins, die Zahl 128 addieren. Probieren Sie es aus.

Mit dieser Methode können Sie direkt an jeder Stelle des Bildschirms alle Zeichen, die der Plus/4 kennt, ausgeben. Eine Ausnahme bilden lediglich die Bildschirmsteuerzeichen. Sie können also nicht mit einem POKE in den Bildschirmspeicher zum Beispiel die Groß-/Kleinschreibung einschalten.

Die Zeichen, die auf dem Bildschirm angezeigt werden, können verschiedene Farben besitzen. Deshalb gibt es für jede Bildschirmposition auch noch einen zweiten Speicher, den Farbspeicher. Die Berechnung für die einzelnen Positionen erfolgt ebenfalls nach dem oben genannten Schema, nur die Anfangsadresse ist eine andere.

Der Plus/4 hat 16 Farben, die auch noch in der Leuchtkraft verändert werden können. Soll in den Farbspeicher gepoket werden, muß man darauf achten, daß dort die Farben mit Null beginnen. Null ist die Farbe schwarz. Die letzte Farbe, hellgrün, besitzt die Farbnummer 15. Die Luminanz der Farben wird durch die Addition von jeweils 16 erhöht. Der letzte Wert, der hier möglich ist, beträgt 127. Dies stellt die Farbe hellgrün in seiner höchsten Luminanzstufe dar. Durch die Luminanzabstufungen, insgesamt sind 8 Stufen möglich, ist es auch möglich, Grautöne mit der Farbe Weiß zu erzeugen. Weiß ist erst in seiner höchsten Luminanzabstufung »richtig Weiß«. Die Farbnummern brauchen hier nicht extra aufgeführt werden, Sie können sie selbst leicht erreichen. Da Sie die Farben auf den Zahlentasten der Tastatur aufgedruckt finden, brauchen Sie vom aufgedruckten Wert nur jeweils eins abziehen. Schwarz hat demnach den Wert Null, Blau den Wert 6 und Orange den Wert 8 (8+1-1).

Veränderungen der Farben werden Sie, je nach Bildschirmart, erst in den höheren Luminanzstufen feststellen können. Ein Beispiel zur Berechnung der Luminanz. Die Farbe Hellblau hat die Farbzahl 13 (8+6-1). Die zweite Luminanzstufe liegt bei 29 (13+16). Die höchste Stufe bei dieser Farbe wird durch die Zahl 125 (13+7*16) dargestellt. Löschen Sie jetzt bitte den Bildschirm und geben Sie dann folgendes ein:

POKE 3472,1
POKE 2448,13+4*16

In der zehnten Zeile erscheint nun ein A in hellblauer Farbe der Luminanzstufe fünf.

Sollten Sie dies auf Ihrem Bildschirm nicht erkennen können, so müssen Sie die Luminanz durch Änderung des Multiplikators vier in fünf oder größer erhöhen.

Ändern Sie einmal den Multiplikator in 11. Die Farbe hat sich nicht verändert, aber der Buchstabe blinkt. Werden in den Farbspeicher Werte größer als 127 gepoket, wird das zugehörige Zeichen blinkend dargestellt. Man braucht also nur zum gewünschten Farbwert 128 addieren.

Übrigens, Sie können den Farbspeicher nach Belieben verändern. Steht in der entsprechenden Speicherzelle des Bildschirmspeichers kein Zeichen, erscheint der Bildschirm unverändert.

Zeichen können Sie auch mit dem POKE-Befehl löschen. Poken Sie einfach ein Leerzeichen (32) in den Bildschirmspeicher. Ein Shift-/Leerzeichen (96) bewirkt dasselbe; aber gehen Sie dann einmal mit dem Cursor an die Stelle, wo vorher das gelöschte Zeichen stand und drücken dann die RETURN-Taste. Es erscheint ein »READY«. Dies liegt am Bildschirmeditor.

Es wird viele Fälle geben, in denen Sie das direkte Zeichenpoken benutzen werden. Es ist manchmal zu aufwendig zu programmieren, mit einem PRINT- oder einem PRINT-USING-Befehl an einer bestimmten Bildschirmstelle Zeichen auszugeben. Bedenken Sie aber stets, viele solcher Pokes machen ein Programm unübersichtlich. Sollen viele Zeichen auf einmal ausgegeben werden, dann ist der PRINT-Befehl wesentlich schneller.

Wir sind nun am Ende dieses Kapitels angelangt. Manch einem mag es stellenweise etwas zu ausführlich gewesen sein. Wir geben Ihnen aber lieber ein Zuviel als Zuwenig an Informationen, gerade in solch einem wichtigen Kapitel. Experimentieren Sie mit den vorgestellten Befehlen ruhig noch weiter, Sie werden sicher noch einiges entdecken, was hier nicht erwähnt wurde.

12 Die Sprungbefehle

Wie Sie wissen, werden die Programmzeilen von der niedrigsten zur höchsten hin abgearbeitet. Dies trifft so lange zu, wie der Rechner nicht zum Verlassen dieses Weges angewiesen wird. Die folgenden Befehle dienen alle dazu, den Programmablauf an einer anderen Stelle des Programms fortzusetzen. Sie unterscheiden sich nicht nur in der Schreibweise, sondern auch in ihren Eigenarten. Lesen Sie dieses Kapitel bitte sorgfältig durch, da die meisten längeren Programme nicht ohne Sprungbefehle auskommem.

12.1 Der GOTO-Befehl

Syntax:

GOTO Zeilennummer

Sehr oft muß ein Programm auf Ereignisse, die während des Programmablaufs auftreten, reagieren. Die genaue Analyse und die Reaktion auf ein Ereignis nehmen aber meist mehr Platz in der Programmzeile ein, in der es auftritt, als in dieser zur Verfügung steht. In diesem Fall muß das Programm verzweigen. Der Syntax des GOTO-Befehls können Sie entnehmen, daß dieser immer von einer Zeilennummer gefolgt werden muß. Diese Zeilennummer repräsentiert das Sprungziel. Die Nummer muß auch wirklich existieren, da sonst das Programm mit einer Fehlermeldung abgebrochen wird.

Trifft der Rechner auf GOTO gefolgt von einer Zeilennummer, so wird der Programmspeicher, beginnend bei der ersten Zeilennummer, nach dieser Zeile durchsucht. Wird sie gefunden, so wird das Programm an dieser Stelle fortgesetzt. Ein Beispiel:

```
10 PRINT "VERZWEIGUNG ZUR ZEILE 60 (J/N)"
20 GETKEY A$
30 IF A$="J" THEN GOTO 60
40 PRINT "ES ERFOLGT VERZWEIGUNG ZUR ZEILE 20"
50 GOTO 20
60 PRINT "DIE VERZWEIGUNG IST ERFOLGT"
70 PRINT "VERZWEIGUNG ZUR ZEILE 10":GOTO 10
```

In dieses Programm wurden gleich mehrere Verzweigungen eingebaut, um Ihnen den GOTO-Befehl näherzubringen. Man kann gut erkennen, daß bei einem GOTO ganze Programmteile übersprungen werden. Auch der Rücksprung zum Programmanfang ist mit diesem Befehl möglich.

Ändern Sie nun bitte die Zeile 30

```
30 IF A$="J" THEN 60
```

Der Ablauf zeigt, auch diese Syntax wird vom Rechner verstanden. Sie können also in einigen Fällen auf das GOTO verzichten. Der Rechner »denkt« sich einfach ein GOTO vor die Zeilennummer. Die Zeilen 50 und 70 lassen sich nicht so ändern. Hier erwartet der Rechner nach wie vor ein GOTO.

12.2 Der ON...GOTO-Befehl

Syntax:

ON Variable GOTO Zeilennr.,Zeilenr.,.....Zeilennr.

Sie möchten ein Programm schreiben, in dem es möglich sein soll, mit Hilfe eines Auswahlmenüs zwischen drei verschiedenen Programmpunkten zu wählen. Man könnte das Menü derart gestalten, daß die Wahl durch die Eingabe einer Zahl erfolgt. Im Programm könnte man dann durch entsprechende IF...THEN-Zeilen auf die Eingabe reagieren.

Es gibt aber eine bessere Lösung. Geben Sie das nachstehende Programm ein:

```
10 PRINT CHR$(147)
20 PRINT "PUNKT EINS (1)"
30 PRINT "PUNKT ZWEI (2)"
40 PRINT "PUNKT DREI (3)"
50 GETKEY A$
60 I=VAL(A$)
80 ON I GOTO 100,200,300
100 PRINT "1.PUNKT" :GOTO 20
200 PRINT "2.PUNKT" :GOTO 20
300 PRINT "3.PUNKT" :GOTO 20
```

Starten Sie das Programm und geben Sie dann Zahlen von eins bis drei ein. Das Programm verzweigt an die jeweils richtige Stelle. Hat die Variable I den Wert Eins, so wird zur ersten Zeilennummer verzweigt, hat sie den Wert Zwei, zur zweiten Zeilennummer und beim Wert Drei zur dritten Zeilennummer, die dem GOTO-Befehl folgt. Geben Sie einmal vier ein. Obwohl keine vierte Zeilennummer folgt, wird ein Sprung zur Zeile 100 ausgeführt. Das gleiche passiert, wenn Sie einen Buchstaben oder eine Null eingeben.

Merken wir uns: Steht in der Variablen ein Wert, der größer ist als die Anzahl der angegebenen Sprungziele oder eine Null, so erfolgt keine Verzweigung. Das Programm wird in der nächsten Zeile weiter ausgeführt. Diese Reaktion ist nicht immer erwünscht. Es wurde aber auch noch Platz für die Zeile 70 gelassen.

```
70 IF I>3 OR I<1 THEN 50
```

Der GOTO-Befehl eignet sich recht gut zum Fortsetzen eines Programms an anderer Stelle. Soll ein Programm aber nach der Abarbeitung einiger Programmzeilen wieder zurückkehren, dann ist dies nur mit einigem Aufwand möglich. Es folgt jetzt ein Befehl, der diesen Aufwand auf ein Minimum reduziert.

12.3 Der GOSUB-Befehl

Syntax:

GOSUB Zeilennummer

In vielen Programmen werden einige Programmzeilen immer wieder benötigt. Nehmen wir als Beispiel die Tastaturabfrage. Da ein Programm absturzsicher programmiert werden soll, ist dies meist eine etwas längere Routine. Oft werden aber an mehreren Stellen des Programms Daten von der Tastatur erwartet. Möglich wäre nun zum Beispiel, diese Routine immer an diesen Stellen zu programmieren. Dies verlängert aber nicht nur das Programm beträchtlich und macht es dadurch unübersichtlich, sondern kostet auch noch viel Speicherplatz und viel Tipparbeit.

Sinnvoller ist es daher, diese Routine nur einmal zu programmieren und das Programm immer dann, wenn Daten eingegeben werden sollen, zu ihr verzweigen zu lassen. Nach der Dateneingabe sollte es zurückkehren, um die Daten dort, wo sie gebraucht werden, zu verarbeiten. Dies genau macht der GOSUB-Befehl.

GOSUB ist die Abkürzung von »GO SUBROUTINE«; auf deutsch »verzweige zum Unterprogramm«. Auch der GOSUB-Befehl braucht eine Zeilennummer, damit der Computer erfährt, an welcher Stelle des Programms er mit der Arbeit weitermachen soll. Bekommt er diesen Sprungbefehl, so merkt er sich, an welcher Stelle des Pro-

gramms er die Arbeit abgebrochen hat. Er verzweigt zur angegebenen Zeilennummer und fährt mit der Abarbeitung der dortigen Anweisungen so lange fort, bis er auf den Befehl RETURN trifft. Nun erfolgt der Rücksprung zu dem Befehl, der dem GOSUB folgt.

Die Gliederung eines Programms könnte so aussehen:

```
HAUPTPROGRAMM
.
GOSUB UNTERPROGRAMM
.
.
GOSUB UNTERPROGRAMM
.
.
END
UNTERPROGRAMM
.
.
RETURN
```

Ein konkretes Beispiel:

```
10 REM HIER BEGINNT DAS HAUPTPROGRAMM
20 GOSUB 100
30 PRINT" HIER IST WIEDER DAS HAUPTPROGRAMM"
40 IF A$="J" THEN GOTO 10
50 PRINT "DAS PROGRAMM WIRD BEENDET"
60 END : REM ENDE DES PROGRAMMS
100 REM HIER BEGINNT DAS UNTERPROGRAMM
110 PRINT "DIES IST DAS UNTERPROGRAMM"
120 PRINT "NOCH EINMAL ?  J/N"
130 GETKEY A$
140 IF A$ <>"J" OR A$ <>"N" THEN 130
150 RETURN :REM ENDE DES UNTERPROGRAMMS
```

Nach dem Start wird sofort zum Unterprogramm verzweigt. Das Unterprogramm wartet auf einen Tastendruck. Wird eine andere Taste als J oder N gedrückt, so wird wieder zur Zeile 130 verzweigt. Ist es eine dieser Tasten, erfolgt der Rücksprung zur Zeile 30. Die Antwort wird nun ausgewertet, und wenn mit J geantwortet wurde, beginnt das Spiel nach einem Sprung in Zeile 10 von vorn.

Man darf auch von einem Unterprogramm aus ein anderes Unterprogramm aufrufen, dieses dann ebenfalls wieder ein Unterprogramm usw. Bei unseren Versuchen kamen wir auf 39 zur gleichen Zeit aufrufbare Unterprogramme. Diese Zahl ist aber auch von den zur gleichen Zeit aktiven Schleifen abhängig. Danach erfolgte die Fehlermeldung »OUT OF MEMORY ERROR«. Die Fehlermeldung basiert auf dem begrenzten Speicherplatz, der für die Speicherung der Rücksprungadressen vorgesehen ist.

Beachten müssen Sie, daß der Rechner nicht auf ein RETURN treffen darf, bevor nicht mit GOSUB ein Unterprogramm aufgerufen wurde. Er findet keine Rücksprungadresse und bricht das Programm mit der Meldung »RETURN WITHOUT GOSUB ERROR« ab. Das Programm läßt sich dann auch nicht mehr mit CONT fortsetzen.

Vermeiden Sie nach Möglichkeit das Aufrufen einer Unterroutine durch eine andere. Ein Programm wird durch diese Verschachtelung sehr schnell unübersichtlich und schlecht erweiterbar.

12.4 Der ON...GOSUB-Befehl

Syntax:

ON Variable GOSUB Zeilennr., Zeilennr.,....,Zeilennr.

Dieser Befehl verhält sich wie der schon besprochene ON-GOTO-Befehl. Lesen Sie dort bitte den Umgang mit der Syntax nach. Da mit diesem Befehl Unterprogramme aufgerufen werden, muß jedes Sprungziel, das GOSUB folgt, mit RETURN enden.

13 Programmschleifen

Oft sollen einige Programmteile mehrmals hintereinander ablaufen. Um diese Programmierung zu erleichtern, besitzt unser Plus/4 Befehle zur Programmierung von Schleifen. Fangen wir gleich mit dem ersten an.

13.1 Der FOR...NEXT-Befehl

Syntax:

FOR Variable = Zahl TO Zahl STEP Zahl

Geben wir den einzelnen Variablen oder Zahlen einmal andere Namen. Das sieht dann so aus:

FOR Laufvariable = Startwert TO Endwert STEP Schrittweite

- FOR muß immer ein Variablennamen folgen.
- TO kann ein Variablennamen oder eine Zahl folgen.
- STEP braucht nicht mitangegeben zu werden, der Rechner setzt dann als Schrittweite Eins fest.

Erst einmal ein kleines Programm:

```
10 FOR I = 0 TO 5
20 PRINT I:". SCHLEIFENDURCHLAUF"
30 NEXT I
40 PRINT:PRINT "  I HAT AM ENDE DER SCHLEIFE DEN WERT";I
```

Nach dem Programmablauf sehen Sie die einzelnen Durchläufe und den Schlußsatz mit der Zahl 6 als den Wert von I auf dem Bildschirm.

I ist die Laufvariable, 5 der Endwert, und da kein STEP folgt, beträgt die Schrittweite Eins. Trifft der Rechner diese Schleifenart an, so sucht er sich erst einmal den Startwert und vergleicht ihn mit dem Endwert. Nun sucht er sich die Schrittweite. Der erste Schleifendurchlauf kann beginnen. Trifft er auf NEXT, wird die Laufvariable um den Betrag der Schrittweite erhöht. Das Ergebnis wird wiederum mit dem Endwert verglichen. Sollte es größer sein als der Endwert, so macht das Programm mit dem Befehl weiter, der NEXT folgt. Ist er kleiner, wird die Schleife noch einmal durchlaufen, und das Vergleichen beginnt aufs neue.

Auch ein Rückwärtszählen ist möglich. Ändern Sie dazu das Programm:

```
10 FOR I = 5 TO 1 STEP -1
```

Die erscheinenden Aussagen treffen zwar nicht mehr zu, es sollte aber nicht stören.

Hatte im ersten Programmbeispiel die Laufvariable I am Ende des Programmes den Wert 6, so hat sie jetzt den Wert 0. Es liegt daran, daß eine Schleife erst dann beendet wird, wenn der Wert der Laufvariablen entweder größer oder wie im zweiten Beispiel kleiner als der Endwert ist. Dies ist beim Programmieren unbedingt zu beachten, wenn in einem anderen Programmteil wieder mit der Laufvariablen gearbeitet werden soll.

Es dürfen auch mehrere Schleifen gleichzeitig aktiv sein. Ein NEXT bezieht sich dann immer auf die zuletzt geöffnete Schleife. Auch jetzt ein Beispiel:

```
10 FOR I=1 TO 2
20 PRINT "WERT DER ERSTEN SCHLEIFE =";I
30 FOR A=1 TO 5
40 PRINT "WERT DER ZWEITEN SCHLEIFE =";A
50 NEXT A
60 NEXT I
```

Die erste Schleife wird zweimal durchlaufen, die zweite fünfmal. Sie erhalten durch diese Verschachtelung zehnmal den Ausdruck für die zweite Schleife. Die Angabe der Variablen in den Zeilen 50 und 60 dürfen auch weggelassen werden. Man sollte aber auf diese Angabe nicht verzichten, da hierdurch schnell erkannt wird, an welcher Stelle eines Programms eine Schleife zu Ende ist.

Wir ändern nun das Programm, um Ihnen eine schnelle Bildschirmgestaltung mit FOR...NEXT-Schleifen zu demonstrieren.

```
10 PRINT CHR$(147);
20 PRINT " ****************************************"
30 FOR I=1 TO 23
40 PRINT "*                                      *"
50 NEXT I
60 PRINT " ****************************************"
70 GETKEY A$
```

Die Zeilen 20 und 60 bestehen aus einem Leerzeichen und 38 Sternchen. In der Zeile 40 sind 38 Leerzeichen zwischen den beiden Sternchen. Nach dem Aufbau des Bildschirms wartet der Rechner auf einen Tastendruck. Das Programm wird so beendet. Wählt man einen solchen Bildschirmaufbau, dann bietet es sich an, ein Fenster in der Größe des Freiraums zu definieren. Die Umrandung kann dann nicht mehr so einfach zerstört werden. Wie man das macht, können Sie bei der ESC-Taste nachlesen.

Diese Art der Schleifen dürfen Sie nicht durch Sprungbefehle verlassen, da sie aktiviert bleiben, bis die oben genannten Bedingungen erfüllt sind. Es gibt im Plus/4-BASIC aber auch Schleifen, die mit besonderen Befehlen verlassen werden können.

13.2 Schleifen mit DO..UNTIL/WHILE

Syntax:

DO UNTIL mathematischer Ausdruck

DO WHILE mathematischer Ausdruck

Wie immer dazu ein Beispiel:

```
10 X = 5
20 DO UNTIL X=0
30 PRINT "X IST ";X
40 X = X-1
50 LOOP
60 PRINT "X HAT AM ENDE DER SCHLEIFE DEN WERT";X
```

In der Zeile 10 wird der Variablen X der Wert 5 zugewiesen. In der Zeile 20 beginnt die Schleife. Sie heißt sehr frei übersetzt: Durchlaufe die Schleife, bis der Wert von X Null ist. In der Zeile 30 wird der aktuelle X-Wert auf dem Bildschirm ausgegeben. In der Zeile 40 wird von X die Zahl 1 abgezogen. Zeile 50 stellt das Ende der Schleife dar. Der Rechner verzweigt wieder zur Zeile 20. Dort wird X überprüft. Sollte X kleiner oder größer als Null sein, wird die Schleife fortgesetzt. Sollte X=0 sein, so fährt das Programm mit dem Befehl fort, der LOOP folgt. In unserem Beispiel ist dies die Zeile 60. UNTIL sorgt dafür, daß diese Schleife erst verlassen wird, bis das, was nach UNTIL definiert wurde, eintritt.

Kommen wir zu WHILE.

```
10 A$ = "J"
20 DO WHILE A$ = "J"
30 PRINT "WEITERMACHEN ? (J)"
40 GETKEY A$
50 LOOP
60 PRINT "A$ = ";A$
```

Solange Sie die Frage mit J beantworten, läuft dieses Programm immer weiter.

Wird WHILE verwandt, wird die Schleife erst verlassen, wenn das Argument, das While folgt, nicht mehr »wahr« ist. Auch diese Art der Schleifen dürfen verschachtelt werden. Wie beim FOR-Befehl, dem ein NEXT folgen muß, muß für jede dieser Schleifen ein LOOP vorhanden sein. Das erste LOOP bezieht sich auf die zuletzt geöffnete Schleife.

Wie oben schon erwähnt, können diese Schleifen jederzeit verlassen werden. Dies darf aber nicht mit einem Sprungbefehl wie GOTO oder GOSUB erfolgen. Die Schleife bliebe dann immer noch wirksam. Zum Verlassen gibt es einen eigenen Befehl.

13.3 Der EXIT-Befehl

Man kann mit DO...LOOP auch unendliche Schleifen aufbauen. Lassen Sie einmal folgendes Beispiel laufen.

```
10 DO
20 A = A+1
30 PRINT A
40 LOOP
```

Das Programm läßt sich nur noch mit der STOP-Taste abbrechen. Diese Schleifen können mit dem EXIT-Befehl verlassen werden. Es folgt ein kleines Programm, mit dem wir EXIT näher beschreiben werden.

```
10 X = 0
20 DO UNTIL X = 500
30 GET A$
40 IF A$ = "S" THEN EXIT
50 X=X+1
60 LOOP
70 PRINT "X=";X;"A$
```

Wenn Sie dieses Programm starten, kann es auf zweierlei Arten beendet werden. Die erste Möglichkeit wäre, X in der Schleife auf den Wert 500 aufaddieren zu lassen. Die zweite Möglichkeit besteht darin, daß während des Programmablaufs die Taste S gedrückt wird. Dies wird durch die Zeile 40 ermöglicht. Drückt man die Taste, so wird die Aussage »wahr«, und das Programm arbeitet den nun folgenden Befehl ab. Er heißt EXIT.

Für den Rechner bedeutet dieser Befehl, daß er die Schleife als abgearbeitet zu betrachten hat und er zum Befehl, der LOOP folgt, verzweigen muß. In diesem Fall ist es die zweite Zeile 70, die die Ausgabe der Variableninhalte bewirkt.

Eine der vielfältigen Anwendungen dieser Schleifen ist das Programmieren sehr komfortabler Eingaberoutinen. Oft sollen Eingaben eine bestimmte Länge nicht überschreiten, andererseits sollen sie auch vorzeitig mit der RETURN-Taste beendet werden können. Das nachfolgende Programm soll lediglich eine Anregung sein, um ein solches Problem mit dem Plus/4 lösen zu können. Zu einer komfortablen Eingaberoutine gehört noch etwas mehr, aber Sie sollen sich ja auch einmal im Programmieren üben.

```
10 REM EINGABEROUTINE
20 L=5
30 DO UNTIL LEN (A$)=L
40 GET E$
50 PRINT E$;
60 IF E$ = CHR$(13) THEN EXIT
70 A$=A$+E$
80 LOOP
90 PRINT:PRINT A$
```

In Zeile 20 wird die zulässige Länge der Eingabe definiert. Wird sie erreicht, endet die Schleife ganz normal. Hat man aber die RETURN-Taste gedrückt, wird die Schleife in der Zeile 60 mit EXIT beendet und gleichzeitig verlassen.

Wir sind am Ende dieses Kapitels angelangt und hoffen, Ihnen die Schleifenprogrammierung verständlich gemacht zu haben. Sicher werden Sie oft feststellen, daß man ohne Schleifen in einem Programm nur sehr selten auskommen kann. Dies trifft im besonderen zu, wenn ein Programm kurz und übersichtlich bleiben soll.

14 Die Stringbefehle

In diesem Kapitel werden Sie sicher auch später einige Male nachlesen. Das BASIC 3.5 bietet starke Befehle, wenn es um die Bearbeitung von Strings geht. Um einzelne Zusammenhänge nicht trennen zu müssen, erfolgt die Erklärung der Befehle nicht in alphabetischer Reihenfolge. Für einige Befehle gibt es auch Umkehrfunktionen. Diese werden ebenfalls im richtigen Zusammenhang erklärt.

Ein String ist eine zusammenhängende Zeichenkette. Er kann aber auch aus nur einem Zeichen bestehen. Zahlen in Strings werden vom Rechner wie Buchstaben behandelt. Das Rechnen ist also mit ihnen nicht möglich. Im Kapitel über die Variablen wurden schon die Stringvariablen behandelt. Daher werden Sie auch schon wissen, daß eine Stringvariable immer durch ein nachgestelltes Dollarzeichen ($) gekennzeichnet wird. Ordnen wir einer solchen Variablen einen Inhalt zu:

```
A$="COMMODORE PLUS/4"
```

Ansehen können Sie sich den Inhalt mit:

```
PRINT A$
```

Der Inhalt kann auch an eine andere Stringvariable übergeben werden:

```
B$ = A$
```

PRINT B$ zeigt nun denselben Inhalt wie PRINT A$

Verknüpfen lassen sich die Stringvariablen ebenfalls:

```
C$ = A$+B$
```

In C$ steht jetzt der Inhalt von A$ und B$. C$ hat also die doppelte Länge. Es ist auch folgendes zulässig:

C$ = C$+C$

Der Inhalt von C$ wurde verdoppelt. Sehen Sie ihn doch einmal an.

PRINT C$

Beachten Sie bitte, ein String darf nicht länger als 255 Zeichen sein.

Interessiert es Sie vielleicht, wie viele Zeichen der String enthält? Natürlich kann man nachzählen, aber warum sollte man es tun, es gibt hierfür eine Funktion. Bei der Erklärung der Befehle wird in diesem Kapitel überwiegend mit Stringvariablen gearbeitet. Es können aber auch Strings verwendet werden, diese müssen nur zwischen Anführungszeichen stehen.

14.1 Der LEN-Befehl

Syntax:

PRINT LEN (Stringvariable)

oder

X = LEN (Stringvariable)

Dieser Befehl gibt Ihnen die Länge eines Strings an. Bei der Stringverarbeitung ist es oft sehr wichtig, die genaue Länge eines Strings zu erfahren, denn oft soll ein String gezielt verändert werden.

14.2 Der LEFT$-Befehl

Syntax:

A$ = LEFT$ (A$, Zahl oder Variable)

oder

PRINT LEFT$ (A$; ?ZAHL ODER VARIABLE)

Mit diesem Befehl können Sie einer Stringvariablen die Anzahl der Zeichen zuordnen, die in der Variablen oder mit der Zahl festgelegt wurde. Die Zählung beginnt mit dem linken Zeichen.

Ein Beispiel:

```
10 INPUT "GEBEN SIE EINEN STRING EIN ";A$
20 INPUT "WIEVIELE ZEICHEN SOLLEN B$ UEBERGEBEN WERDEN ";A
30 PRINT "DER STRING IST";LEN(A$);"ZEICHEN LANG"
40 B$= LEFT$(A$,A)
50 PRINT B$
60 PRINT "WEITER ? (J)"
70 GETKEY K$
80 IF K$ = "J" THEN 10
```

Geben Sie als String »COMPUTER« ein; die Anzahl der Zeichen soll vier betragen. Die Länge des Strings beträgt acht Zeichen. Diese Länge wird mit LEN(A$) festgestellt und Ihnen mitgeteilt. Nun werden die ersten vier Zeichen in B$ übernommen und Ihnen B$ dann ebenfalls angezeigt. Die Länge und der Inhalt von A$ wurden nicht verändert; die ersten vier Zeichen wurden lediglich in B$ kopiert. Es gibt aber auch die Möglichkeit, Strings gezielt zu verändern. Dazu kommen wir etwas später in diesem Kapitel.

Geben Sie eine Zahl ein, die größer ist als die Anzahl der Zeichen in A$. Es kommt keine Fehlermeldung und die Stringvariable B$ enthält ebenfalls den Inhalt von A$. Bitte achten Sie unbedingt darauf, daß die Anzahl der abzutrennenden Zeichen nicht kleiner als Null oder größer als 255 ist, sonst erfolgt eine Fehlermeldung.

14.3 Der RIGHT$-Befehl

Syntax:

B$ = RIGHT$ (A$, Variable oder Zahl)

oder

PRINT RIGHT$ (A$, Variable oder Zahl)

Mit diesem Befehl lassen sich die rechten Zeichen übergehen oder auf dem Bildschirm darstellen. Ändern wir nun das Programm wie folgt:

```
40 B$=RIGHT$ (A$,A)
```

Wird nun eine Zeichenkette eingegeben und dann die Anzahl der in B$ zu übernehmenden Zeichen, so erscheinen sie in der richtigen Reihenfolge auf dem Bildschirm.

Die Ausgabe beginnt also nicht beim äußerst rechten Zeichen, sondern bei dem, welches von rechts gezählt mit A definiert wurde. Auch bei RIGHT$ darf die Zahl größer als die im String vorhandene Anzahl von Zeichen sein. Es wird auch diesmal lediglich der gesamte String ausgegeben. Die Zahl der zu übernehmenden Zeichen darf aber auch hier nicht kleiner als Null oder größer als 255 sein.

Bieten einem die eben beschriebenen Befehle schon sehr viel für die Stringverarbeitung, gibt es dennoch einen sehr viel stärkeren.

14.4 Der MID$-Befehl

Syntax:

B$= MID$(A$,V,A)

PRINT MID$(A$,V,A)

- V steht für »Von«
- A steht für »Anzahl«

Mit diesem Befehl lassen sich Teile von Strings übergeben. Es werden A Zeichen, beginnend beim Vten Zeichen übernommen. Die Zählung erfolgt von links.

Sie können selbstverständlich den Variablen V und A jederzeit andere Namen geben. Auch Zahlenangaben sind erlaubt. Die Variable V darf den Wert von Eins nicht unterschreiten, eine Fehlermeldung wäre das Ergebnis. Die Variable A darf nicht kleiner als Null sein. Beide Variablen dürfen auch nicht den Wert 255 überschreiten. Wir ändern noch einmal das Programm vom Beginn dieses Kapitels.

```
25 INPUT "GEBEN SIE DEN ANFANG EIN ";V
40 B$=MID$ (A$,V,A)
```

Die Zeile 25 ist hinzugekommen. Die Zeile 40 muß entsprechend geändert werden.

Wenn Sie mit dem Programm einige Versuche unternommen haben, werden Sie leicht feststellen können, daß auch tatsächlich immer nur der definierte Teilstring übernommen und dann später ausgegeben wird. Wird für V eins eingegeben und für A die Länge des eingegebenen Strings, wird der komplette String übergeben.

14.5 Die Änderung eines Strings mit MID$

Mit MID$ können Strings sehr einfach verändert werden. MID$ darf auch links vom Gleichheitszeichen (=) stehen. Das würde zum Beispiel so aussehen:

```
A$ = "1234567890"
B$ = "ABCD"

MID$ (A$,5) = B$
```

Nach dieser Operation sieht der Inhalt von A$ so aus:

1234ABCD90

Ein Teil des Strings wurde also durch den Inhalt der Stringvariablen B$ überschritten. Das Ersetzen begann, wie im MID$-Befehl festgelegt, bei dem fünften Zeichen. Die Angabe der Länge braucht nicht zu erfolgen, da diese von der Länge des Strings in B$ bestimmt wird.

Das gleiche Ergebnis ergibt auch

```
MID$ (A$,5) ="ABCD"
```

ACHTUNG: Die Länge des zu ändernden Strings darf durch die Manipulation nicht geändert werden, der Rechner meldet sich sonst mit einer Fehlermeldung.

Es ist natürlich auch möglich, Teile eines Strings durch Teile eines anderen zu ersetzen. Rechts vom Gleichheitszeichen müssen dann lediglich die Befehle wie LEFT$, RIGHT$ oder auch ein weiteres MID$ stehen.

14.6 Der INSTR-Befehl

Syntax:

```
X = INSTR (A$, S$, Startposition)
```

Mit diesem Befehl können Sie Zeichenketten, repräsentiert durch S$ in einem bestimmten String, hier AS, suchen lassen. Wird das Zeichen gefunden, wird seine Position im String der Variablen X übergeben. Ist das Zeichen oder die Zeichenkette nicht im definierten String enthalten, so enthält X nach der Suche den Wert Null. Wird keine Startposition angegeben, so beginnt die Suche am Anfang des Strings, sonst bei der Startposition. Ein Beispiel:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 GETKEY B$
30 X=INSTR (A$,B$)
40 IF X=0 THEN 20
50 PRINT B$;X
60 GOTO 20
```

Wenn Sie dieses Programm starten, werden nur die Zeichen berücksichtigt, die in der Variablen A$ definiert wurden. Diese Zeichen und deren Position im String werden angezeigt. Mit diesem kleinen Programm ist es also möglich, auf einfache Art und Weise zur Eingabe zuzulassen oder zu sperren. Man spart so bei einer Eingaberoutine einige IF THEN... Abfragen und somit Platz und Zeit.

14.7 Strings in Zahlen umwandeln mit VAL

Syntax:

X = VAL (X$)

Bekanntlich kann man mit Zahlen, die in Strings enthalten sind, nicht rechnen, da sie als Zeichen interpretiert werden. Mit dem Befehl VAL ist es möglich, diese Zeichen in Zahlen umzuwandeln. In der Variablen X steht dann die Zahl, die aus X$ gewonnen wurde.

Der String X$ wird von links nach rechts nach Zahlen durchsucht. Trifft der Rechner auf andere Zeichen, so wird die Suche abgebrochen. Die Zahlen müssen im String in der richtigen Schreibweise vorhanden sein. Soll eine Zahl vollständig generiert werden, ist zum Beispiel ein Komma statt eines Dezimalpunktes nicht zulässig. Da die Suche beim Antreffen unerlaubter Zeichen abgebrochen wird, muß die zu generierende Zahl am Anfang eines Strings stehen.

Einige Beispiele:

Befehl	Ergebnis
PRINT VAL ("ABC123")	0
PRINT VAL ("123.45")	123.45
PRINT VAL ("123AB4")	123
PRINT VAL ("-1.2A1")	-1.2

14.8 Zahlen in Strings umwandeln mit STR$

Syntax:

A$ = STR$ (A)

Mit diesem Befehl werden Zahlen in Strings umgewandelt. Sie können dann mit den String-Befehlen weiter verarbeitet werden. Dieser Befehl ist die Umkehrung von VAL. Im String sind nach diesem Befehl die Ziffern als Zeichen vorhanden. Es kann also jetzt auch mit diesem String gerechnet werden.

Das Beispiel:

```
A$ = STR$ (1234.56)
PRINT A$
```

ergibt:

1234.56

Hiermit ist dieses Kapitel beendet. Sie werden beim Programmieren immer wieder Strings bearbeiten müssen. Der Plus/4 liefert Ihnen ein leistungsfähiges Werkzeug, man muß es nur richtig anwenden. Die Grundlagen dazu haben wir Ihnen hoffentlich leicht verständlich geliefert. Also, auf in die Welt der Stringverarbeitung.

15 Die hochauflösende Grafik des Plus/4

Eine besondere Stärke des Commodore Plus/4 ist zweifellos seine Grafikfähigkeit. Er ist in der Lage 320 x 200, also 64000 Bildpunkte auf dem Bildschirm einzeln anzusteuern. Der Plus/4 zaubert 16 Farben auf den Bildschirm, und diese auch noch in acht Helligkeitsstufen.

Das ist noch nicht alles. Es ist auch möglich, Text und Grafik gleichzeitig darzustellen, eine Fähigkeit, die viele weitaus größere und teurere Computer nicht beherrschen. Außerdem gibt es noch die Darstellung der »Shapes«. Diese werden an anderer Stelle eingehend beschrieben. Für einen Rechner dieser Preisklasse sind diese Fähigkeiten ganz enorm!

15.1 Bildschirmaufbau

Nachdem Sie Ihren Plus/4 eingeschaltet haben, befindet sich der Rechner im sogenannten Textmodus. Das heißt, auf dem Bildschirm sind nur die Zeichen darstellbar, die Sie auf der Tastatur sehen.

Eine Zeile besteht aus 40 Zeichen, und 25 Zeilen passen untereinander auf den Bildschirm. Also 25 x 40 = 1000 Zeichen. Jedes Zeichen besteht aus 8 x 8 Bildpunkten. Diese Punkte können auch einzeln angesteuert werden.

Diese Möglichkeit besteht in der hochauflösenden Grafik. Dann ist der Bildschirm in 40 x 8 = 320 Punkte in der Waagerechten (X-Richtung) und 25 x 8 = 200 Punkte in der Senkrechten (Y-Richtung) eingeteilt. Also 320 x 200 Bildpunkte, 64000 einzeln ansteuerbare Punkte. Damit lassen sich ganz ausgezeichnete Grafiken darstellen. Wir zeigen Ihnen ein Beispiel so einer hochauflösenden Grafik.

Manchmal ist es wünschenswert, Grafik und Text gleichzeitig darzustellen. Auch das ist mit dem Plus/4 möglich. Der dritte Modus: Hochauflösende Grafik und Text. Dabei wird der Bildschirm aufgeteilt in einem oberen Bereich von 320 x 160 Bildpunkten und einen unteren Bereich für fünf Zeilen Text.

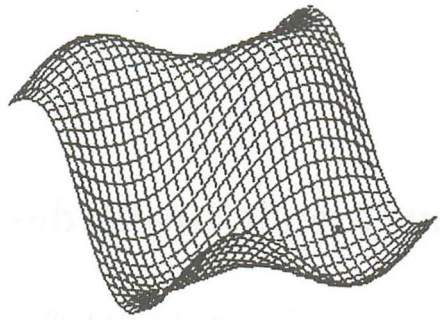


Bild 15.1: *Beispiel*

Die hochauflösende Grafik hat leider eine Schwäche: Es sind pro 8 x 8 Punkte-Matrix (vergl. Textmodus) nur zwei Farben darstellbar. Auch der Plus/4 hat seine Grenzen. Wir können mit unserem Computer trotzdem großartige farbige Grafiken erzeugen.

Außer der hochauflösenden Grafik mit nur zwei Farben pro 8 x 8 Punkte-Matrix kennt der Plus/4 noch die sogenannte Mehrfarbengrafik. Dabei sind pro 8 x 8 Punkte-Matrix vier Farben darstellbar. Die zusätzlichen Farben müssen aber durch eine geringere Auflösung des Bildschirms »bezahlt« werden. Die Auflösung im Mehrfarbenmodus beträgt 160 x 200 Bildpunkte. Diesen Mehrfarbenmodus kann man ebenfalls mit Text kombinieren, der vierte Modus: Mehrfarbengrafik und Text.

15.2 Die Grafikmodi des Plus/4

Modus	normale	scalierte Auflösung	Bezeichnung
Modus 0	25 x 40 Zeichen (Text)		Textmodus
Modus 1	320 x 200	1024 x 1024 Punkte	Hochauflösende Grafik
Modus 2	320 x 160	1024 x 816 Punkte und fünf Zeilen Text	Hochauflösende Grafik und Text
Modus 3	160 x 200	1024 x 1024 Punkte	Mehrfarbengrafik
Modus 4	160 x 160	1024 x 816 Punkte und fünf Zeilen Text	Mehrfarbengrafik und Text

Das Einschalten der Grafik

Genaugenommen befindet sich der Plus/4 immer im Grafikmodus. Der sogenannte Textmodus (der Einschaltzustand) ist nämlich der Grafikmodus 0. Um die hochauflösende Grafik einzuschalten, brauchen wir nur den Befehl

GRAPHIC 1

einzugeben und schon ist auf dem Bildschirm die hochauflösende Grafik zu sehen. Abgebildet wird der Grafikspeicher. In diesem Speicherbereich befinden sich irgendwelche Daten, die sich als Punkte oder Farben bemerkbar machen können. Daher sehen Sie jetzt auf Ihrem Bildschirm eventuell nur ein buntes Durcheinander. Wir müssen den Grafikspeicher also löschen. Das kann entweder schon beim Einschalten der Grafik geschehen oder durch einen speziellen Befehl. Beim Einschalten der Grafik müssen wir zum Löschen einen zweiten Parameter eingeben. Zum Beispiel:

GRAPHIC 4,1

Durch GRAPHIC 4 wird die Mehrfarbengrafik mit fünf Zeilen Text eingeschaltet, und durch die angehängte »1« wird dieser Grafikspeicher gelöscht (siehe auch unsere Zusammenstellung der Grafikbefehle).

Die zweite Möglichkeit, den Bildschirm zu löschen, besteht in dem Befehl

SCNCLR

Beide Befehle funktionieren auch im Grafikmodus 0, also im Textmodus. Geben Sie einmal GRAPHIC 0,1 ein. Der Bildschirm wird sofort gelöscht.

Fassen wir zusammen: Die Grafik wird eingeschaltet durch den Befehl

GRAPHIC Modus (eventuell Parameter für das Löschen)

Die Grafik wird wieder ausgeschaltet durch das Umschalten in den Textmodus:

GRAPHIC 0 (eventuell Parameter für das Löschen)

Beachten Sie bitte, daß der Plus/4 für die hochauflösende Grafik oder für die Mehrfarbengrafik viel Speicherplatz benötigt. Für jeden Punkt auf dem Bildschirm wird ein Bit reserviert. Dazu kommen noch die Farbinformationen. Zusammen wird für die hochauflösende bzw. Mehrfarbengrafik 10 Kbyte Speicher reserviert.

Die reservierten 10 Kbyte für die Grafik stehen Ihnen auch nach Ausschalten der Grafik nicht wieder zur Verfügung. Das muß auch so sein, denn Sie sollen die Möglichkeit haben, ein einmal erstelltes Grafikbild immer wieder ein- und ausschalten zu können. Erst wenn Sie das Grafikbild tatsächlich nicht mehr benötigen, sollten Sie die 10 Kbytes Speicher »zurückholen«. Und das machen Sie mit dem Befehl:

GRAPHIC CLR

Jetzt steht Ihnen auch wieder der ganze Plus/4-Speicher zur Verfügung. Tatsächlich ist das Grafikbild jetzt immer noch nicht gelöscht. Erst wenn Sie ein Programm in den Speicherbereich laden oder eintippen, indem sich der Grafikspeicher befand, ist das Bild verfälscht oder gelöscht. Das gleiche bewirken Variablen, die Sie eventuell in Ihrem Programm brauchen.

15.3 Der Pixel-Cursor

Was ein »Cursor« ist, haben Sie ja schon kennengelernt. Es handelt sich dabei um das kleine blinkende Rechteck auf Ihrem Bildschirm. Der Cursor blinkt genau dort, wo das nächste Zeichen gedruckt wird.

So etwas gibt es auch bei unserer hochauflösenden Grafik. Allerdings ist der Pixel-Cursor (PC) stets unsichtbar. Die Bildpunkte in der Grafik bezeichnet man auch als »Pixel«, daher der Name Pixel-Cursor.

Wieso ist der Pixel-Cursor eigentlich unsichtbar? Sie werden es sich schon denken können: Während Sie bei der Texteingabe in Ihren Computer natürlich erkennen müssen, wo das nächste Zeichen erscheint, wird die hochauflösende Grafik doch nur von einem Programm erstellt. Die Grafik wird ja nicht direkt von der Tastatur einge-

geben. Es gibt dagegen Anwendungen für Ihren Computer, denken Sie an ein Zeichenprogramm, wo der Pixel-Cursor eventuell sichtbar gemacht werden muß.

Die Position des Pixel-Cursors ist im Computer gespeichert. Wir können mit einem BASIC-Befehl diese gespeicherte Position abfragen. Der Befehl dazu lautet:

RDOT (n)

Für n können die Werte 0, 1 oder 2 stehen.

n = 0 ergibt die X-Position
n = 1 ergibt die Y-Position
n = 2 ergibt die zuletzt in einem Befehl angegebene Farbzone.

Sie sehen, auch die Farbzone ist im Zusammenhang mit dem Pixel-Cursor abgespeichert. Wozu nun das Ganze?

Wenn in einem Programm viele Grafik-Befehle nacheinander abgearbeitet werden, müssen jedesmal Farbzone, X- und Y-Anfangsposition, X- und Y-Endposition usw. angegeben werden. Diese Arbeit kann erheblich vereinfacht werden, da im PC stets die Endwerte gespeichert werden. Diese Werte können beim nächsten Befehl wieder als Anfangsposition verwendet werden. Das macht man, indem die Werte in den Befehlen einfach weggelassen werden. Dazu ein Beispiel:

```
10 GRAPHIC1,1
20 DRAW1,10,10TO20,20
30 DRAW1,20,20TO30,10
40 DRAW1,30,10to40,20
```

RUN

Sie schalten die Grafik wieder aus durch

GRAPHIC 0

Ändern Sie nun die Zeilen 30 und 40 wie folgt:

```
30 DRAWTO30,10
40 DRAWTO40,20
```

RUN

Sie sehen die gleiche Grafik wie im ersten Beispiel. Wir haben aber die Anfangspositionen und die Farbzone einfach weggelassen.

Der Befehl DRAW bietet, nebenbei bemerkt, eine weitere Vereinfachung. Sie können unser Beispiel auch wie folgt eingeben:

```
10 GRAPHIC1,1
20 DRAW1,10,10TO20,20TO30,10TO40,20
```

Jetzt werden nur die Endpositionen angegeben.

Kommen wir noch einmal zurück zum Pixel-Cursor. Wir können den Pixel-Cursor auch (unsichtbar) an jede beliebige Stelle des Bildschirms setzen. Dazu dient der Befehl:

LOCATE x,y

Mit x und y werden die Positionen für den PC angegeben. Wir probieren das einmal an einem Programm aus:

```
10 GRAPHIC1,1
20 LOCATE10,10
30 DRAWTO20,20TO30,10TO40,20
```

RUN

Durch LOCATE 10,10 wurde der PC an die Position X=10 und Y=10 gesetzt. Deshalb konnten wir im DRAW-Befehl auf die Anfangsposition verzichten.

Wir zeigen Ihnen ein kleines Beispielprogramm, das die großartigen Möglichkeiten des Commodore Plus/4 ausnutzt. Sie kennen sicher die Uhren, die im Fernsehprogramm vor den Nachrichten gezeigt werden. Das wäre doch einmal eine interessante Programmieraufgabe für uns. Für den Plus/4 ist das natürlich überhaupt kein Problem, ihm fehlt nur das entsprechende Programm. Dem helfen wir jetzt ab. Stellen Sie sich einmal vor, wie Ihre Gäste staunen würden, wenn Ihr Fernsehgerät ständig die Uhrzeit anzeigt - ein herrlicher Gag!

15.4 Beispielprogramm FERNSEHUHR

Bild 15.2: *Beispiel »Fernsehuhr«*

Das Problem erscheint auf den ersten Blick einfacher, als es ist: Das Programm soll in BASIC realisiert werden. Als Uhr wird die eingebaute Uhr des Plus/4 verwendet, die natürlich vorher eingestellt werden muß. Das erste Programm sollte irgendwie einen Sekundentakt bekommen, damit der Sekundenzeiger auch schön regelmäßig und genau weitergeht. Leider klappt das in BASIC nicht sehr gut. Mit der Maschinensprache könnte man sehr gut die Interruptmöglichkeiten des Plus/4 ausnutzen (Interrupt = Unterbrechung). Wir programmieren aber in BASIC, also müssen wir in unserem Programm ständig die Uhr abfragen, um festzustellen, ob schon eine Sekunde vergangen ist. Das ist zwar nicht ganz so schön, geht aber auch.

Das zweite Problem besteht darin, die Uhr auf dem Bilschirm darzustellen. Der Plus/4 kennt zwar sehr starke Grafikbefehle, es ist aber nicht ganz einfach, die 60 Minutenstriche und die 12 Stundenstriche zu zeichnen.

Das dritte Problem ist im Gegensatz zu den ersten tatsächlich etwas schwieriger. Und zwar geht es um das »Timing«, die Geschwindigkeit unseres Programms.

Nun würden wir dieses Programm hier nicht abdrucken, wenn es nicht einwandfrei laufen würde. Geben Sie also das Programm jetzt erst einmal ein. Es beinhaltet teilweise sehr umfangreiche Befehlszeilen und sehr viele Variablen, passen Sie daher bei der Eingabe auf. Tippen Sie das Programm Zeichen für Zeichen genau ein, und speichern Sie es ab, bevor Sie es starten.

```
 10 REM FERNSEHUHR
 20 DIM HX%(60),HY%(60),MX%(60),MY%(60),SX%(60),SY%(60)
 30 X=150:Y=100:RA=0:VOL8
 40 COLOR0,1
 50 COLOR1,2
 60 COLOR4,1
 70 PRINTCHR$(147)
 80 INPUT"UHRZEIT (HHMMSS)";TI$
 90 PRINT"MOMENT BITTE ..."
100 REM BERECHNEN DER DATEN FUER DIE UHRZEIGER
110 FOR W=450TO96STEP-6
120 RE=40:GOSUB330
130 HX%(T)=X2:HY%(T)=Y2
140 RE=60:GOSUB330
150 MX%(T)=X2:MY%(T)=Y2
160 RE=70:GOSUB330
170 SX%(T)=X2:SY%(T)=Y2
180 T=T+1:NEXT
190 REM ZEICHNEN DER UHR
200 GRAPHIC1,1
210 RA=71:RE=80:FORW=450TO96STEP-6:GOSUB330:DRAW,X1,Y1TOX2,Y2:NEXT
220 RA=81:RE=85:FORW=450TO96STEP-30:GOSUB330:DRAW,X1,Y1TOX2,Y2:NEXT
230 DRAW0,X,YTOMX%(M%),MY%(M%):M%=VAL(MID$(TI$,3,2)):DRAW,X,YTOMX%(M%),MY%(M%)
240 DRAW0,X,YTOHX%(H%),HY%(H%)
250 H%=(5*VAL(LEFT$(TI$,2)))+INT(M%/12):IFH%>59THENH%=H%-60
260 DRAW,X,YTOHX%(H%),HY%(H%)
270 S%=VAL(MID$(TI$,5,2)):IFS%<>S1%THEN280:ELSE270
280 DRAW0,X,YTOSX%(S1%),SY%(S1%)
290 DRAW,X,YTOSX%(S%),SY%(S%):SOUND1,900,1
300 IFS%=0ORS1%=M%THENS1%=S%:GOTO230
310 IFS1%=H%THENS1%=S%:GOTO240:ELSES1%=S%:GOTO270
320 REM WINKELFUNKTIONEN
330 X1=X+INT(COS(W*3.14/180)*RA)
340 X2=X+INT(COS(W*3.14/180)*RE)
350 Y1=Y-INT(SIN(W*3.14/180)*RA)
360 Y2=Y-INT(SIN(W*3.14/180)*RE)
370 RETURN
```

Wenn Sie das Programm abgespeichert haben, starten Sie es mit RUN. Sie werden nun nach der Uhrzeit gefragt. Dazu einige Anmerkungen: Die Uhrzeit muß mit jeweils zwei Stellen für die Stunden, Minuten und Sekunden eingegeben werden. Zum Beispiel:

Uhrzeit 9.15 Uhr und 0 Sekunden	- Eingabe: 091500
Uhrzeit 21.46 Uhr und 20 Sekunden	- Eingabe: 214620

Anschließend wird der Computer ca. 30 Sekunden rechnen, dann erscheint die Uhr mit den Zeigern auf dem Bildschirm.

Stoppen können Sie das Programm mit der RUN/STOP-Taste. Geben Sie dann den Befehl »GRAPHIC0« ein.

Wir haben gesagt, daß unser Programm schnell genug sein muß, um rechtzeitig den Sekundenzeiger zu zeichnen. Aber damit ist es noch nicht getan, der Minuten- und Stundenzeiger muß auch innerhalb einer Sekunde bewegt werden. Das muß so schnell geschehen, daß Verzögerungen zumindest mit dem Auge nicht wahrgenommen werden können. Hinzu kommt, daß es trotz der Grafikbefehle des Plus/4 nicht ganz einfach ist, von einem Mittelpunkt gleichlange Linien im Abstand von 6 Grad zu zeichnen. Am einfachsten ist, die 60 Positionen der Zeiger schon vor dem eigentlichen Uhrenprogramm zu errechnen. Dann kann man nämlich mit dem (schnellen) DRAW-Befehl die Zeigerstellungen vom Mittelpunkt aus zu den gespeicherten Positionen zeichnen. Das haben wir in unserem Programm auch genau so gemacht. Daher braucht der Computer zunächst auch die 30 Sekunden »Bedenkzeit«. Für die X- und Y-Werte verwenden wir Integervariablen (gekennzeichnet mit dem Prozentzeichen). Die Integervariablen haben den Vorteil der schnelleren Verarbeitung.

In der Zeile 20 werden die Integervariablen für die Zeigerpositionen dimensioniert. Für den Stundenzeiger stehen die Variablen HX% und HY%, für den Minutenzeiger MX% und MY% und für den Sekundenzeiger SX% und SY%. Da die Uhr 60 Einteilungen hat, dimensionieren wir jeweils mit 60.

In Zeile 30 bestimmen wir die Mittelpunktkoordinaten der Uhr. Die Variable RA beinhaltet den inneren Radius des Kreisringes - hier zunächst für die Zeiger, die natürlich im Mittelpunkt des Kreises beginnen; daher der Radius 0.

Die Zeilen 40 bis 60 setzen die Farben für den Hintergrund, Vordergrund und den Rahmen des Bildschirms.

Durch die Zeile 70 wird der Bildschirm gelöscht.

In Zeile 80 wird die aktuelle Uhrzeit in die Systemvariable TI$ übernommen.

Da die Berechnungen ca. 30 Sekunden beanspruchen, haben wir die Zeile 90 mit in das Programm aufgenommen.

Wir benötigen im Unterprogramm (ab Programmzeile 330) die Winkelfunktionen SIN und COS, um die 60 Zeigerpositionen auf dem Kreisumfang zu berechnen.

In der Zeile 110 wird der entsprechende zu berechnende Winkel in die Variable W übernommen. Lassen Sie sich nicht irritieren durch die Winkelwerte von 450 bis 96, die um jeweils -6 zurückgezählt werden. Das hat den einfachen Grund, daß die Uhr rechts herum geht und daß 12 Uhr auch tatsächlich oben ist.

Unbedingt benötigt wird für die Berechnung die Länge der Zeiger, gespeichert in den Variablen RE minus RA. RA hat den Wert 0, daher können wir direkt an RE die Länge ablesen: Der Stundenzeiger soll 40 Bildpunkte, der Minutenzeiger 60 und der Sekundenzeiger 70 Punkte lang sein.

Die Indexnummer der Integervariablen steht in der Variablen T, die in Zeile 180 hochgezählt wird.

Nun wird die hochauflösende Grafik eingeschaltet (Zeile 200).

Zeile 210 zeichnet die 60 Minutenstriche, wieder unter Verwendung der Winkelfunktionen.

In der Zeile 220 werden die 12 Stundenstriche gezeichnet.

Jetzt folgt die eigentliche Uhr. In den Zeilen 230 bis 310 wird die Systemvariable TI$ abgefragt und die Zeiger gezeichnet.

Anschließend wartet das Programm in Zeile 270 darauf, daß die Uhr eine Sekunde »weitergeht«.

Dann wird in Zeile 280 der alte Sekundenzeiger gelöscht und in Zeile 290 der neue gezeichnet. Damit die Uhr auch »tickt«, steht in dieser Programmzeile auch ein SOUND-Befehl.

In Zeile 300 und 310 wird festgestellt, ob der Sekundenzeiger auf der »12«, steht. Dann muß der Minutenzeiger weitergesetzt werden.

Anmerkung:

Wenn Sie später den BASIC-Befehl CIRCLE kennenlernen werden, werden Sie sich eventuell fragen, wieso in diesem Uhrenprogramm so »umständlich« mit den Winkelfunktionen gearbeitet wird. Einerseits haben Sie recht, denn mit CIRCLE kann man auch einzelne Punkte auf dem Kreisumfang zeichnen. Wir könnten also unsere Uhr auch mit mehreren CIRCLE-Befehlen darstellen. Nur wäre die Programmierung genauso kompliziert und auch nicht schneller. Deshalb diese Lösung des Problems.

Wir können die Darstellung der Uhr leicht ändern. Warum muß die Uhr denn eigentlich rund sein. Probieren Sie doch einmal folgende Änderungen:

```
330 X1=X+INT(COS(W*3.14/180)*RA*2)
340 X2=X+INT(COS(W*3.14/180)*RE*2)
```

oder

```
330 X1=X+INT(COS(W*3.14/180)*RA*2)
340 X2=X+INT(COS(W*3.14/180)*RE*1.5)
```

oder

```
330 X1=X+INT(COS(W*3.14/90)*RA)
340 X2=X+INT(COS(W*3.14/90)*RE)
350 Y1=Y-INT(SIN(W*3.14/90)*RA)
360 Y1=Y-INT(SIN(W*3.14/90)*RE)
```

oder

```
340 X2=X+INT(COS(W*3.14/180)*RE*2)
```

oder

```
330 X1=X+INT(COS(W*3.14/180*2)*RA)
340 X2=X+INT(COS(W*3.14/180*2)*RE)
350 Y1=Y-INT(SIN(W*3.14/180/2)*RA)
360 Y2=Y-INT(SIN(W*3.14/180/2)*RE)
```

Probieren Sie ruhig selbst einmal einige Veränderungen aus. Sie können Ihrer Phantasie freien Lauf lassen. Wir wünschen Ihnen viel Spaß mit der »Fernsehuhr«.

15.5 Die Skalierung

Eine Besonderheit bietet der Plus/4 mit dem Befehl SCALE. Durch diesen Befehl wird der Bildschirm feiner eingeteilt in 1024 Punkte horizontal und vertikal. Natürlich ist das nur rein rechnerisch möglich, die Bildschirmauflösung beträgt weiterhin »nur« 320 x 200 Punkte.

Einschalten der Scalierung durch:

SCALE 1

Ausgeschaltet wird die Scalierung durch:

SCALE 0

Damit Sie sehen, was SCALE bewirkt, geben Sie bitte folgendes ein:

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 CIRCLE 1,100,100,50
50 CHAR 1,0,24,"TASTE DRUECKEN",1
60 GETKEY A$
70 GRAPHIC 0
80 END
```

Auf dem Bildschirm sehen Sie nach dem Programmstart einen einfachen Kreis. Drükken Sie eine Taste, und fügen Sie die folgende Programmzeile ein:

```
35 SCALE 1
```

Wenn Sie nun das Programm erneut starten, wird der Kreis sehr viel kleiner dargestellt. Durch die Einteilung des Bildschirms in 1024 Punkte in X- und Y-Richtung erscheint unser Kreis mit dem Radius 100 nun im linken oberen Bildschirmbereich.

Hier wieder unsere kurze Programmbeschreibung:

In der Zeile 10 haben wir die Farbe für den Hintergrund bestimmt (hier Schwarz).

In der Zeile 20 wird die Vordergrundfarbe, also die Farbe der Bildpunkte, eingestellt (hier Weiß).

Zeile 30 schaltet die hochauflösende Grafik ein und löscht den Bildschirm.

Zeile 40 zeichnet den Kreis. Der Kreis hat die Mittelpunktkoordinaten X = 100 und Y = 100. Der Radius hat die Größe 50.

In der Zeile 50 wird ein Text in die hochauflösende Grafik geschrieben. Dieser Text erscheint in der Zeile 24 und beginnt in Spalte 20.

Zeile 60 erwartet einen Tastendruck.

Nach dem Tastendruck wird in Zeile 70 unsere Grafik wieder ausgeschaltet.

Durch das Einfügen der Zeile 35 wurde die Scalierung eingeschaltet. Der Kreis wird dann viel kleiner dargestellt.

Um auf dem scalierten Bildschirm genau die gleichen Koordinaten zu erreichen wie im Normalmodus, müssen wir die X- und Y-Koordinaten umrechnen:

X x 3,2 = scalierter X-Wert
Y x 5,12 = scalierter Y-Wert
X x 6,4 = scalierter X-Wert im Mehrfarbenmodus

Im Mehrfarbenmodus ist X mit 6,4 zu multiplizieren, da in dem Modus nur 160 Punkte in der X-Richtung darstellbar sind. Die Erstellung der Grafik wird nach unserer Erfahrung mit SCALE etwas langsamer (ca. 0,5% bis 2,5%).

Soviel zum SCALE-Befehl. Probieren Sie doch einmal diesen Befehl bei unserer »Fernsehuhr« aus. Sie müssen dazu die Scalierung einschalten, bevor die Uhr gezeichnet wird. Fügen Sie also die Programmzeile 205 ein:

```
205 SCALE 1
```

15.6 Der Grafikbefehlssatz

Einige Befehle des Plus/4 haben wir nun schon in unseren Beispielprogrammen kennengelernt. Der gesamte Befehlssatz ist aber noch sehr viel umfangreicher. Für die Grafik gibt es viele starke Befehle. Leider ist das Bedienungshandbuch des Plus/4 in Hinsicht auf den Grafikbefehlssatz sehr unübersichtlich. Wir möchten daher an dieser Stelle alle Befehle des Plus/4 auflisten und kurz beschreiben, die mit der Programmierung der hochauflösenden Grafik oder der Mehrfarbengrafik zusammenhängen.

Bei einigen Befehlen muß zusätzlich noch ein oder mehrere Parameter übergeben werden. Parameter, das sei an dieser Stelle bemerkt, sind zusätzliche Informationen zu den Befehlen, die der Computer zur Ausführung benötigt. Parameter können auch durch Variablen angegeben werden.

Bei einigen Befehlen können bestimmte Parameter weggelassen werden. Der Computer setzt dann für diese fehlenden Parameter selbst Werte ein (meist 0 oder 1).

In der folgenden Beschreibung werden die Parameter als Buchstaben angegeben. Wir brauchen hier die gleichen Bezeichnungen wie in Ihrem Plus/4-Handbuch, denn wir wollen Sie natürlich nicht durch abweichende Bezeichnungen verwirren. Wir erklären nach jedem Befehl, welche Werte die Parameter haben müssen. Parameter, die nicht unbedingt notwendig sind, haben wir in Klammern geschrieben. Bei einigen Befehlen muß oder kann die sogenannte Farbzone mitangegeben werden.

15.7 Die Farbzonen

Es gibt fünf Farbzonen:

0 = Hintergrundfarbe
1 = Vordergrundfarbe (Farbe der Bildpunkte)
2 = Mehrfarben 1
3 = Mehrfarben 2
4 = Bildschirmrand

In jeder Farbzone können Sie eine Farbe abspeichern. Dies geschieht durch den Befehl COLOR. Mit COLOR können wir die Farben unserer Grafik bestimmen.

Syntax:

COLOR Farbzone,Farbe(,Helligkeitswert)

Beispiel:

COLOR 1,2,4

(Farbzone 1, Farbe 2, Helligkeit 4)

Durch diesen Befehl wird der Bildschirmvordergrund grau eingefärbt. Das sehen Sie unmittelbar durch die Farbe des Cursors.

COLOR 4,3,7

färbt den Bildschirmhintergrund hellrot ein.

Mit dem folgenden Beispielprogramm können Sie das gesamte Farbspektrum des Plus/4 ansehen. Gleichzeitig sehen Sie einmal, wie gut (oder schlecht) Ihr Fernsehgerät bzw. Monitor die Farben wiedergibt.

```
10 FOR LU = 0 TO 7
20 FOR T = 1 TO 2
30 FOR CO = 1 TO 16
40 COLOR 1,CO,LU
50 PRINT CHR$(18);"  ";
60 NEXT CO
70 PRINT
80 NEXT T
90 NEXT LU
100 COLOR 1,2
```

Die Zeilen 10, 20, 30, 60, 80 und 90 sind FOR-TO-NEXT-Schleifen, die die Helligkeit, die Farbe und die Anzahl der zu druckenden Zeilen pro Helligkeitswert angeben.

In der Zeile 40 wird die Vordergrundfarbe (Zeichenfarbe) bestimmt.

Zeile 50 druckt zwei reverse Leerzeichen (Space). Die Reversdarstellung wird durch CHR$(18) eingeschaltet.

Zeile 100 schaltet wieder die normale Farbe ein.

Das Plus/4-BASIC bietet Ihnen zwei Befehle, mit denen Sie die momentan eingestellte Farbe der Farbzonen erfahren können.

Die Befehle RCLR und RLUM

RCLR (Farbzone)

Die Klammern müssen mit eingegeben werden. Durch diesen Befehl erfahren Sie die Farbnummer der angegebenen Farbzone. Beachten Sie bitte, im Direktmodus würde der Befehl ein

```
?SYNTAX ERROR
```

hervorrufen. Wenn Sie ihn ausprobieren möchten, geben Sie also vor dem Befehl noch ein PRINT ein.

RLUM (Farbzone)

Mit diesem Befehl können Sie die Helligkeit der Farbe in der angegebenen Farbzone erfahren.

WICHTIG: Bei den Befehlen COLOR, RCLR und RLUM müssen Sie immer die Farbzone angeben. Bei allen anderen Befehlen können Sie den Parameter »Farbzone« weglassen. Der Computer setzt dann dafür den Wert 1 ein.

Der Befehl BOX

Syntax:

BOX(fz),x1,y1,x2,y2(,wi)(,fü))

BOX zeichnet ein Rechteck.

fz	Farbzone. Ohne Angabe = Farbzone 1. Das Rechteck wird mit der Farbe gezeichnet, die durch den Befehl COLOR bestimmt wurde.
x1,y1	Koordinate links oben.
x2,y2	Koordinate rechts unten.
wi	Das Rechteck wird im Winkel von wi Grad (um den Mittelpunkt) gedreht gezeichnet. Keine Angabe = 0 Grad.
fü	Füllen. Das Rechteck kann entweder nur als Umriß (wenn fü weggelassen wird oder fü = 0 ist) gezeichnet werden oder voll ausgemalt werden (bei fü = 1).

Ein Beispiel zum BOX-Befehl:

```
10 GRAPHIC1,1
20 BOX,100,50,200,150
30 BOX,100,50,200,150,45,1
```

Das erste Rechteck wird normal an die angegebenen Koordinaten gezeichnet. Es wird nicht gefüllt.

Das zweite Rechteck dagegen wird um den Mittelpunkt, der sich aus den Koordinaten ergibt, um 45 Grad gedreht und ausgefüllt gezeichnet.

Der Befehl CHAR

Mit CHAR können Sie in jedem Grafikmodus (auch im Textmodus) Texte an jede Stelle des Bildschirms drucken. Verwechseln Sie CHAR nicht mit CHR$!

Syntax:

CHAR (fz),x,y,text(,re)

fz	Farbzone
x	X-Koordinate wie im Textmodus (Werte von 0 bis 39)
y	Y-Koordinate von 0 bis 24
Text	Text in Anführungsstrichen oder als String-Variable
re	1 = Text revers drucken, 0 = normal, ohne Angabe = 0

Dieser Befehl ist eine große Hilfe bei der Ausgabe von Text. Und zwar nicht nur im Grafikmodus, sondern auch im Textmodus. Um Text an eine beliebige Stelle des Bildschirms auszudrucken, muß normalerweise umständlich mit Cursor-Steuerzeichen oder mit vielen PRINT-Befehlen gearbeitet werden. Diese Arbeit kann mit CHAR wesentlich vereinfacht werden. Sie geben hinter CHAR die Koordinaten an, an denen der Text erscheinen soll, und schon wird der Text ausgegeben. Nun kann man den CHAR-Befehl aber nicht so anwenden wie den PRINT-Befehl. Es lassen sich zum Beispiel keine numerischen Variablen ausdrucken (außer mit STR$). Hierzu gibt es einen kleinen Trick, der dieses Problem beseitigt:

Positionieren Sie mit dem CHAR-Befehl einfach den Cursor an die gewünschte Stelle und arbeiten dann mit PRINT weiter. Beispiel:

```
CHAR,5,5,"":PRINT"ZEILE FUENF SPALTE FUENF"
```

Hiermit wird der Cursor an die gewünschte Position (X=5 Y=5) gebracht. In den Anführungsstrichen hinter CHAR haben wir nichts eingesetzt, es wird also auch nichts ausgedruckt. Der Cursor bleibt aber an der angegebenen Stelle stehen. Mit PRINT kann dann der Text weiter ausgegeben werden.

Der letzte Parameter hinter dem CHAR-Befehl, das sogenannte Reverse-Flag, kann im Grafikmodus 0, also im Textmodus, nicht angewandt werden. Dafür lassen sich im Textmodus alle Steuerzeichen (also auch REVERSE ON) im String hinter dem CHAR-Befehl einsetzen. Zum Beispiel:

```
CHAR,5,5,CHR$(18):PRINT"ZEILE FUENF SPALTE FUENF"
```

oder

```
CHAR,5,5,CHR$(18)+"ZEILE FUENF SPALTE FUENF"
```

Der Befehl CIRCLE

Mit CIRCLE können Kreise, Ellipsen, Kreissegmente oder Vielecke gezeichnet werden.

Syntax:

CIRCLE (fz),(x,y),xr,(yr),(wa),(we),(wi),(ws)

fz	Farbzone
x,y	Mittelpunktkoordinaten. Bei fehlender Angabe gelten die Werte des Pixel-Cursors.
xr	X-Radius
yr	Y-Radius, bei fehlender Angabe wird der gleiche Wert wie xr genommen.
wa	Anfangswinkel des Kreissegmentes. Ohne Angabe = 0 Grad.
we	Endwinkel des Kreissegmentes. Ohne Angabe = 360 Grad.
wi	Drehen des Kreises um wi Grad.
ws	Winkel zwischen zwei Segmenten. Ohne Angabe = 2 Grad.

Zum Anfangswinkel des Kreissegmentes: 0 Grad, bzw. 360 Grad ist oben. Der Kreis wird rechts herum gezeichnet. Wenn Sie also als Anfangswinkel eine Zahl größer als 0 angeben, fehlt also der entsprechende Winkel rechts vom Anfang. Dementsprechend verhält sich der Endwinkel. Wenn Sie nun für wi eine Zahl größer als 0 eingeben, wird der 0-Grad-Punkt, also der Anfangspunkt unseres Kreises, um den Wert wi rechts herum gedreht.

Einige Beispiele zum CIRCLE-Befehl:

CIRCLE 1,100,100,100,100

zeichnet einen Kreis an die Koordinaten X = 100, Y = 100 in der Farbe der Farbzone 1 mit dem X-Radius 100 und dem Y-Radius 100.

CIRCLE,100,100,100

zeichnet genau den gleichen Kreis, da Sie die Farbzone nicht anzugeben brauchen, wenn Sie Farbzone 1 benötigen. Außerdem muß der Y-Radius nicht mitangegeben werden, wenn dieser genauso groß sein soll wie der X-Radius.

CIRCLE,100,100,100,,180

zeichnet ein Kreissegment um den Mittelpunkt X=100 und Y=100 mit dem Radius 100 beginnend bei dem Winkel 180 Grad.

CIRCLE,100,100,100,50,270,90

zeichnet den Teil einer Ellipse von 270 bis 90 Grad.

CIRCLE,100,100,100,50,,,,90

zeichnet einen Rhombus.

CIRCLE,100,100,100,,,,45,90

zeichnet ein Quadrat.

Sie sehen auf jeden Fall, daß CIRCLE ein äußerst starker und vielseitiger Befehl ist. Die vielen Parameter machen CIRCLE zwar ein wenig kompliziert, dieser Befehl entschädigt uns aber reichlich.

15.8 Beispielprogramm für hochauflösende Grafik

Wenn Ihnen das folgende Programm nicht zu lang ist, können Sie es ja einmal eintippen und mit RUN starten.

```
10 COLOR0,3,0
20 COLOR1,8
30 GRAPHIC1,1
40 RA=30
50 XA=120
60 GOSUB100
70 XA=200
80 GOSUB100
90 END
100 FORKR=0TO360STEP5
110 X=XA+INT(COS(2*3.14*KR/360)*3*RA)
120 Y=100-INT(SIN(2*3.14*KR/360)*2*RA)
130 CIRCLE,X,Y,RA
140 NEXTKR
150 RETURN
```

In Zeile 10 und 20 werden die Farben des Vordergrunds und des Hintergrunds bestimmt.

Zeile 30 schaltet die hochauflösende Grafik ein.

Die Zeilen 40 und 50 übergeben der Variablen RA den Wert 30 und XA den Wert 120 (RA ist der Radius der Kreise, XA ist der Mittelpunkt der ersten Ellipse).

Zeile 60 ruft das Unterprogramm in Zeile 100 auf.

Nach der Rückkehr aus dem Unterprogramm wird in Zeile 70 ein neuer Wert für XA eingesetzt.

In Zeile 80 springt das Programm wieder in das Unterprogramm ab Zeile 100.

Zeile 90 beendet das Programm.

Die Zeilen 100, 110, 120 und 140 bewirken praktisch das gleiche wie der Befehl CIRCLE,120,100,90,60,,,,5. Sie sehen, wie schwer es ist, ohne CIRCLE eine Ellipse zu zeichnen.

Zeile 130 zeichnet die einzelnen kleinen Kreise.

Zeile 150 ist der Rücksprung aus dem Unterprogramm.

Auch dieses Programm verwendet wieder die Winkelfunktionen, um die Ellipsen zu berechnen. Es wäre auch hier wieder theoretisch möglich, CIRCLE zu verwenden. Das allerdings nur mit erhöhtem Programmieraufwand.

Der Befehl DRAW

Diesen Befehl haben wir schon einige Male gesehen. Mit DRAW lassen sich Linien in der hochauflösenden Grafik zeichnen.

Syntax:

DRAW (fz) (,x1,y1) (TOx2,y2) (TOx3,y3)...

fz	Farbzone
x1,y1	X- und Y-Koordinate des Anfangspunktes
TOx2,y2	X- und Y-Koordinate des Endpunktes, wenn eine Linie gezeichnet werden soll.
TOx3,y3	X- und Y-Werte einer Linie mit den Anfangspunkten x2 und y2

Sie können mit DRAW also einzelne Punkte darstellen, einzelne oder mehrere Linien zeichnen.

Der Befehl PAINT

Mit dem PAINT-Befehl können wir geschlossene Flächen, das heißt Flächen, die durch Bildpunkte umschlossen sind, mit der Farbe der angegebenen Farbzone »füllen«.

Syntax:

PAINT (fz),x,y(,m)

fz	Farbzone
x,y	Der Punkt in der geschlossenen Fläche, um den herum ausgefüllt werden soll.
m	Modus: 0 oder keine Angabe = Der Bildschirm wird mit der Farbe gefüllt, die in der Farbzone steht, 1 = wenn die angegebene Farbzone nicht die Hintergrundfarbe ist, wird der Bildschirm zwar gefüllt, die gesetzten Bildpunkte bekommen aber die Hintergrundfarbe, so daß sie sich farblich vom Hintergrund abheben.

Beispiel:

```
10 GRAPGIC1,1
20 CIRCLE,100,100,100
30 PAINT1,100,100
```

Der Befehl SCNCLR

Mit SCNCLR wird der Bildschirm gelöscht. Dabei ist es egal, in welchem Grafikmodus sich der Computer befindet. Der gerade angezeigte Bildschirm wird gelöscht. Das funktioniert also auch im Textmodus.

15.9 Shapes

Wir sind nun bei den sogenannten Shapes angelangt. Was ist ein Shape? Übersetzt heißt »Shape« Form oder Körper. Es handelt sich dabei um Rechteckformen. Ein Shape ist eine rechteckige Fläche des Grafikbildschirms, die als Stringvariable im Computerspeicher abgespeichert wird. Diese Stringvariable kann nun jederzeit wieder ausgelesen werden, so oft man will.

Noch einmal: Wir sind in der Lage, Teile unseres Grafikbildes in eine Variable zu übernehmen und können diese dann wieder auslesen und auf dem Bildschirm sichtbar machen.

Zwei Befehle ermöglichen uns, das vorher Gesagte zu erreichen:

SSHAPE und **GSHAPE**

Wir werden uns hierzu gleich ein Beispielprogramm ansehen.

```
10 COLOR1,1
20 COLOR0,2
30 GRAPHIC1,1
40 CIRCLE1,20,20,10
50 DRAW1,10,10TO30,30
60 DRAW1,10,30TO30,10
70 SSHAPE A$,10,10,30,30
80 SCNCLR
90 FORA=0TO4
100 CHAR1,0,20,"GSHAPE A$,X,Y,"+STR$(A)
110 DRAW1,0,0TO100,100
120 FORX=1TO100STEP15
130 FORY=1TO100STEP10
140 GSHAPE A$,X,Y,A
150 NEXTY:NEXTX:SCNCLR:NEXTA
160 GRAPHIC0
```

In Zeile 10, 20 und 30 setzen wir die Farben und schalten die Grafik ein.

In Zeile 40, 50 und 60 zeichnen wir ein Gebilde aus einem Kreis und zwei Geraden.

Zeile 70 speichert dieses Gebilde in die Variable A$ ab. Das geschieht durch den Befehl SSHAPE.

In den Zeilen 80 bis 160 wird der Bildschirm gelöscht und die Variable A$ mit dem Befehl GSHAPE in den verschiedenen Modi wieder auf den Bildschirm gebracht.

Zeile 110 zeichnet eine Diagonale auf den Bildschirm, damit Sie sehen, wie die Shapes den vorhandenen Bildschirminhalt verändern.

Unser Gebilde ist also in A$ gespeichert. Wir können es an jeder Stelle des Bildschirms wiedergeben. Wir erkennen aber auch, daß die Wiedergabe unseres Shapes verschieden ist. Dazu sehen wir uns die Befehlssyntax von SSHAPE und GSHAPE an.

Der Befehl SSHAPE

Syntax:

SSHAPE Stringvariable,x1,y1(,x2,y2)

Stringvariable	kann jeden Variablennamen annehmen.
x1,y1	Die Koordinaten einer Ecke unseres Shapes.
x2,y2	Die Koordinaten der anderen Ecke. Wird diese Angabe weggelassen, werden die Koordinaten des Pixel-Cursors genommen.

Mit dem Befehl SSHAPE läßt sich ein Shape, also ein Bereich unseres Grafikbildschirms abspeichern. Da das Shape in einer Stringvariablen gespeichert ist und eine Stringvariable maximal 255 Zeichen beinhalten darf, ist der abzuspeichernde Bildschirmbereich natürlich begrenzt.

Ein Zeichen (ein Byte) beinhaltet acht Bildpunkte. Sie wissen, daß der Grafikbildschirm aus 64000 Bildpunkten besteht, im Mehrfarbenmodus aus 32000 Punkten. Diese große Menge kann natürlich nicht in 255 Byte abgespeichert werden. Hinzu kommt, daß in unsere Stringvariable genaugenommen nur 251 x 8 Bildpunkte passen, da die letzten 4 Zeichen die Länge und die Breite unseres Shapes angeben. Wir können also theoretisch 251 mal 8 Bildpunkte als Shape speichern. Praktisch geht das leider auch nicht. Das Maximum sind 2000 Punkte. Dabei spielt das Format, also das Verhältnis zwischen Länge und Breite, eine große Rolle. So ist es zwar möglich, ein Shape mit X = 200 und Y = 10 Punkte abzuspeichern. Dagegen kann ein Shape, das 100 Punkte lang und 20 Punkte breit ist, nicht abgespeichert werden. Der Computer würde die Meldung

```
?STRING TOO LONG ERROR
```

ausgeben. Das liegt daran, daß im String nicht nur Punkt für Punkt abgespeichert ist, sondern auch noch weitere Informationen, die das Format betreffen. Dazu hat Commodore im Bedienungshandbuch zwei Formeln angegeben. Anhand dieser Formeln können Sie feststellen, ob Ihr Shape in einen String paßt oder nicht.

Wir sind der Meinung, daß das ewige Ausprobieren mit diesen langen Formeln keine Arbeitserleichterung bringt. Wir brauchen diese Formel nicht, sondern probieren einfach aus, wie lang und wie breit das Shape sein darf. Man kann dafür natürlich auch ein Programm nutzen. Der Plus/4 bietet durch den Befehl TRAP die Möglichkeit, auf einen Fehler wie den oben angegebenen »STRING TOO LONG ERROR« gezielt zu reagieren. Das heißt, daß ein Programm bei einem Fehler nicht unbedingt abbricht, sondern weiterläuft. Dazu ein Beispiel mit SHAPE:

```
10 YE=20
20 GRAPHIC1
30 TRAP60
40 SHAPE A$,0,0,200,YE
50 GRAPHIC0:PRINT YE;"IST DER GROESSTE WERT":END
60 YE=YE-1:GOTO30
```

Durch TRAP60 springt das Programm bei einem eventuell auftretenden Fehler in die Programmzeile 60. Dort wird der Wert für YE um 1 erniedrigt und zur Programmzeile 30 gesprungen.

Wenn Ihnen das Programm zu lang sein sollte und Sie trotzdem feststellen möchten, ob Ihr Shape in eine Stringvariable paßt, dann probieren Sie es doch einfach im Direktmodus aus.

Sie möchten z. B. ein Shape abspeichern, das in der X-Richtung 30 Bildpunkte lang ist. Dann könnte Y ca. 2000 dividiert durch 30, also etwa 66 Punkte lang sein. Das probieren wir einmal aus. Geben Sie dazu im Direktmodus ein:

SSHAPE A$,0,0,30,66

Sollte nun die Fehlermeldung

```
?NO GRAPHICS AREA ERROR
```

erscheinen, dann müssen Sie die Grafik einmal einschalten und wieder ausschalten. Der Rechner muß also den Speicherbereich für die Grafik reservieren, sonst können wir kein Shape speichern. Wenn alles richtig läuft, müßte der Computer jetzt ein

```
?STRING TOO LONG ERROR
```

ausgeben. Unser Shape ist also zu lang. Wir benutzen die Cursor-Tasten und gehen auf die Befehlszeile (SHAPE...) und ändern die letzte Zahl. Ändern wir die Zahl zunächst in 63. Drücken Sie die RETURN-Taste. Wir bekommen wieder eine Fehlermeldung. Wenn wir aber den Wert 62 einsetzen und RETURN drücken, meldet der Rechner »READY«. Das Shape mit X=30 kann also nur 62 Punkte in Y-Richtung haben (= 1860 Punkte). Uns erscheint dieses Probieren als einfachste Möglichkeit, den Bereich eines Shapes zu testen.

Der Befehl GSHAPE

Mit dem Befehl GSHAPE... kann das Shape wieder sichtbar gemacht werden.

Syntax:

GSHAPE Stringvariable,x,y,Wiedergabemodus

x und y sind die Koordinaten der linken oberen Ecke des Bereiches, an dem unser Shape dargestellt werden soll. Die Angaben für x,y und den Wiedergabemodus können auch weggelassen werden. Dann nimmt der Computer die Koordinaten des Pixel-Cursors. Der Wiedergabemodus ist dann 0. Es gibt fünf Möglichkeiten für den Wiedergabemodus bei GSHAPE:

Modus 0	wie aufgenommen
Modus 1	revers
Modus 2	Wiedergabe ODER-verknüpft
Modus 3	UND-verknüpft
Modus 4	EXCUSIV-ODER-verknüpft

Erinnern Sie sich bitte an die logischen Verknüpfungen im fünften Kapitel dieses Buches. In unserem Beispielprogramm haben wir alle vier Modi des GSHAPE-Befehls dargestellt. Dazu gehören auch die bitweise logischen Verknüpfungen des Shape-Inhalts mit dem Bildschirminhalt. Leider ist die Geschwindigkeit, in der die Shapes gezeichnet werden, sehr langsam. Sie können die Shapes daher nicht mit den beim C64 oder C128 bekannten Sprites vergleichen. Bewegte Grafiken mit Shapes sind einfach zu langsam. Man wird sie daher bei Spielen kaum so einsetzen können, wie es mit den Sprites so hervorragend möglich ist.

15.10 Das Abspeichern einer Grafik

Wir haben schon angesprochen, daß Programme möglichst auf Kassette oder Diskette abgespeichert werden sollten. Sie haben dann die Möglichkeit, Ihre Programme jederzeit wieder zu laden. Dadurch ersparen Sie sich natürlich die mühevolle Tipparbeit. Außerdem schleichen sich beim Eintippen der Programme immer wieder Tippfehler ein. Berechtigterweise werden Sie sich fragen, wieso eine Grafik abspeichern, wenn doch das Programm, das die Grafik erstellt, abgespeichert werden kann.

Sie haben recht, in den meisten Fällen wäre es wirklich einfacher, eine Grafik neu zu erstellen, als die gleiche Grafik von einem Speichermedium einzuladen.

Es gibt aber durchaus Situationen, in denen es ratsam ist, die fertige Grafik zu laden. Denken Sie daran, daß es möglicherweise sehr lange dauern kann, bis ein Programm

eine Grafik erstellt hat. Wenn im Programm umfangreiche und komplizierte Berechnungen durchgeführt werden müssen, kann es mitunter Minuten dauern, bis die Grafik auf dem Bildschirm ist. Das Laden der Grafik kann dann eventuell schneller vonstatten gehen als das Erstellen der Grafik.

Es kann auch erforderlich sein, daß eine vorhandene Grafik durch ein neues Programm verändert werden soll. Dazu kann es ebenfalls notwendig sein, die Grafik im unveränderten Zustand abzuspeichern.

Eine weitere Anwendung: Sie haben ein Programm, in dem auf dem Grafikbildschirm nacheinander verschiedene Grafiken erscheinen sollen. Es müssen nicht unbedingt ganze Grafikseiten eingeladen werden, auch kleinere Ausschnitte sind möglich.

Wie Programme gespeichert oder geladen werden, wissen Sie. Es gibt dafür die Befehle SAVE und LOAD bzw. DSAVE und DLOAD. Eine Grafik dagegen läßt sich nicht so leicht abspeichern oder laden. Diese Prozedur wird vom BASIC leider nicht unterstützt. Doch was wäre unser Plus/4, wenn es nicht doch eine Möglichkeit gäbe. Wir müssen uns, um dieses Problem zu lösen, zunächst ansehen, wie die Grafik im Computerspeicher dargestellt ist. Außerdem muß man auch wissen, welche Speicherzellen von einer Grafik »belegt« werden.

Um es einmal vorwegzunehmen, eine Grafik läßt sich abspeichern, indem man einfach Byte für Byte aus dem Speicher ausliest und diese Bytes auf die Kassette oder Diskette abspeichert. Zum Abspeichern einer Grafik brauchen wir das eingebaute Monitorprogramm. Wir starten dieses Programm mit dem BASIC-Befehl:

MONITOR

Im Monitorprogramm gibt es nun die Möglichkeit, einzelne oder beliebig viele Bytes abzuspeichern. Dazu verwenden wir den Befehl:

S "Name",n,aaaa,eeee

Name	der Name, unter dem das File abgespeichert werden soll.
n	Gerätenummer (8 für Diskette, 1 für Kassette).
aaaa	Anfangsadresse in hexadezimaler Schreibweise.
eeee	Endadresse des zu speichernden Bereichs plus 1.

Um den gesamten Grafikspeicher abzuspeichern (z. B. auf Diskette), geben Sie folgendes ein:

S "GRAFIK",8,1800,4000

Das sehen wir uns an einem Beispiel an. Dazu müssen wir zunächst eine Grafik erstellen.

```
10 COLOR0,1
20 COLOR1,2
30 GRAPHIC1,1
40 FORX=0TO300STEP10
50 DRAW,X,0TOX+10,199TOX+10,0
60 NEXT
70 GRAPHIC0
```

RUN

Geben Sie, nachdem das Programm beendet ist, den Befehl

MONITOR

ein, und speichern Sie die Grafik mit dem oben angegebenen Befehl ab. Wenn Sie einen Kassettenrecorder benutzen, müssen Sie natürlich statt einer 8 eine 1 eingeben. Bei der Kassette dauert das Speichern fast 4 Minuten, bei der Diskette nur ca. 30 Sekunden. Sie können die Grafik aber auch ohne Farbinformationen abspeichern. Die Farbe läßt sich auch nachträglich durch einfache COLOR-Befehle ändern. Im Monitor geben wir also ein:

S "GRAFIK",8,2000,4000

15.11 Das Laden der Grafik

Zum Wiedereinladen einer Grafik muß zunächst der Grafikspeicher vorbereitet werden. Das geschieht mit dem Befehl GRAPHIC, zum Beispiel mit:

GRAPHIC1,1

Dann laden Sie die Grafik wie ein Maschinenprogramm:

LOAD "Name",g,1

Unbedingt angeben müssen Sie die Sekundäradresse 1, um das File auch an die richtige Adresse zu laden. Die Grafik läßt sich auch von einem Programm aus laden. Sie muß dann ebenfalls wie ein Maschinenprogramm geladen werden. Übrigens muß der Grafikbildschirm nicht unbedingt eingeschaltet sein. Es muß nur sichergestellt sein, daß der Speicherbereich für die Grafik reserviert ist. Lesen Sie bitte zum Thema »Laden von Maschinenprogrammen« auch das Kapitel 16.

15.12 Funktionen plotten mit dem Plus/4

Wir werden Ihnen nun anhand eines Programms die ausgezeichneten Grafikfähigkeiten des Plus/4 zeigen. Es handelt sich dabei um ein Programm zur Darstellung von Funktionen jeder Art. Das Programm ist zwar etwas umfangreich, dafür ist es aber auch recht gut.

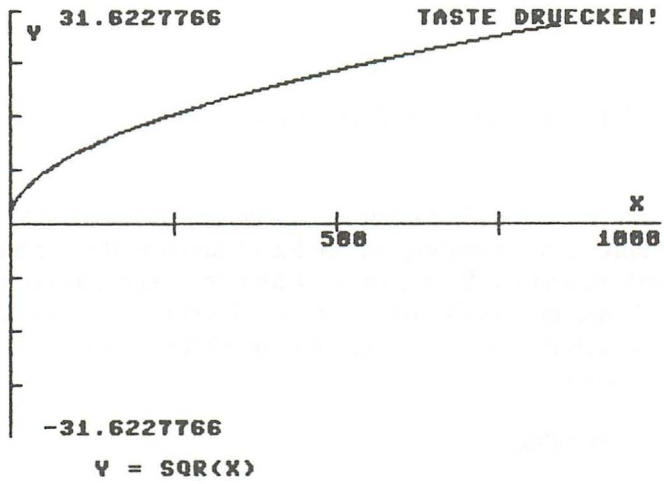


Bild 15.3: *Funktionsplot*

Zur Bedienung des Programms:

Sie müssen Ihre Funktion in der Programmzeile 540 eintragen.

```
540 DEFFNA (X) = Funktion
```

Achten Sie auf die genaue Syntax der Befehle. Um die Werte brauchen Sie sich keine Sorgen zu machen, der Rechner wird eventuell auftretende Fehler (z. B. Division durch Null) nicht mit einem Programmabbruch ahnden, sondern weiterrechnen. Wir haben hier wieder den Befehl TRAP gebraucht.

Nach dem Start des Programms müssen Sie den Maximalwert für die X-Achse eingeben. Geben Sie dazu eine positive Zahl größer Null an, bei Winkelfunktionen z. B. die Werte 90, 180 oder 360. Der Rechner berechnet den Maximalwert für Y und teilt

teilt daraufhin den Bildschirm entsprechend ein, so daß das Bildschirmformat (200 Punkte in Y-Richtung) immer optimal genutzt wird.

```
10 REM FUNKTIONEN PLOTTEN
20 REM FUNKTION STEHT IN ZEILE 540
30 GOSUB540
40 REM FEHLERBEHANDLUNG
50 TRAP520
60 PRINTCHR$(147):PRINT"          FUNKTION  PLOTTEN"
70 PRINT"----------------------------------------"
80 PRINT"ACHTUNG: DER BILDSCHIRM WIRD FUER EINIGE SEKUNDEN AUSGESCHALTET"
90 PRINT:PRINT
100 INPUT"MAXIMALWERT FUER X";XM
110 REM BILDSCHIRM AUSSCHALTEN
120 POKE65286,PEEK(65286)AND239
130 REM KOORDINATENKREUZ WIRD UNSICHTBAR GEZEICHNET
140 GRAPHIC1,1
150 DRAW1,0,100TO319,100
160 DRAW1,0,0TO0,199
170 FORZ=0TO319STEP80
180 DRAW1,Z,95TOZ,105
190 NEXTZ
200 FORZ=0TO199STEP25
210 DRAW1,0,ZTO5,Z
220 NEXTZ
230 REM Y-MAXIMALWERT WIRD ERMITTELT
240 FORF=0TOXMSTEPXM/320
250 X=F/180*3.14
260 Y=FNA(X)
270 IFY>YMTHENYM=Y
280 NEXT
290 REM DER BILDSCHIRM WIRD EINGESCHALTET
300 POKE65286,PEEK(65286)OR16
310 REM PIXEL-CURSOR AUF X=0, Y=100
320 LOCATE0,100
330 REM BERECHNUNG DER FUNKTION
340 FORF=0TOXMSTEPXM/320
350 REM UMRECHNUNG IN DAS BOGENMASS
360 X=F/180*3.14
370 Y=FNA(X):Y=100-Y*(100/YM)
380 DRAW TOF*(320/XM),Y
390 NEXT
400 REM BESCHRIFTEN
410 CHAR1,38,11,"X"
420 CHAR1,18,13,STR$(INT(XM/2))
430 CHAR1,35,13,STR$(INT(XM))
440 CHAR1,1,1,"Y"
450 CHAR1,2,0,STR$(YM)
460 CHAR1,2,24,STR$(-(YM))
470 CHAR1,25,0,"TASTE DRUECKEN!",1
480 GETKEYA$
490 GRAPHIC0,1
500 LIST540
510 END
520 RESUME NEXT
530 REM HIER STEHT DIE FUNKTION
540 DEFFNA(X)=SIN(X)/SIN(2*X)
550 RETURN
```

Eine kurze Beschreibung des Programms

In der Zeile 30 springt unser Programm in die Zeile 540. Dort steht die Funktion, die mit der Anweisung DEFFN dem Namen FNA zugewiesen wird.

Durch den Befehl TRAP in der Zeile 50 wird bei einem auftretenden Fehler zur Zeile 520 gesprungen. Dort steht der Befehl RESUME, der bewirkt, daß das Programm den Fehler ignoriert und an der Stelle weiterarbeitet, wo der Fehler aufgetreten ist.

In Zeile 100 erwartet das Programm den Maximalwert für X.

Die Programmzeile 120 ist eine Besonderheit. Hier wird der Bildschirm ausgeschaltet. Das hat folgende Bewandtnis: Um den Y-Maximalwert zu erhalten, muß unser Programm die gesamte Berechnung einmal komplett durchführen. Das ist natürlich sehr zeitaufwendig. Um diese Berechnung etwas zu beschleunigen, wird die Bildschirmausgabe abgeschaltet. Dadurch wird der Rechner schneller, da die aufwendige Organisation des Bildschirmaufbaus wegfällt.

Die Zeilen 140 bis 220 zeichnen ein Koordinatenkreuz. Das geschieht natürlich auch unsichtbar, denn der Bildschirm ist ja noch immer abgeschaltet.

In den Zeilen 240 bis 280 wird nun der Maximalwert für Y berechnet.

Zeile 300 schaltet den Bildschirm wieder ein. Dazu bekommt in der Speicherzelle 65286, das ist ein Register des TED, das Bit 4 den Wert 1.

Die Zeilen 410 bis 470 beschriften die Grafik, auch mit den Werten für X-Max. und Y-Max.

Die Zeile 480 erwartet einen Tastendruck.

Zeile 490 schaltet die Grafik wieder aus.

Durch Zeile 500 wird die Programmzeile mit der Funktion aufgelistet. Das ist sehr praktisch, da Sie dann sofort die Funktion verändern können. Der LIST-Befehl kann also auch in einem Programm gebraucht werden. Das Programm läuft übrigens nach einem LIST-Befehl nicht weiter. Sie können daher Zeile 500 weglassen.

Zeile 510 beendet das Programm (kann weggelassen werden).

Durch RESUME NEXT in Zeile 520 läuft das Programm bei einem Fehler weiter.

Wenn Sie keine Winkelfunktion darstellen möchten, dann entfernen Sie bitte die Zeilen 250 und 360. In diesen Zeilen wird eine Umrechnung in das Bogenmaß vorgenommen, da der Computer die Winkelangaben im Bogenmaß erwartet.

Hier noch ein Beispiel einer fertigen Grafik:

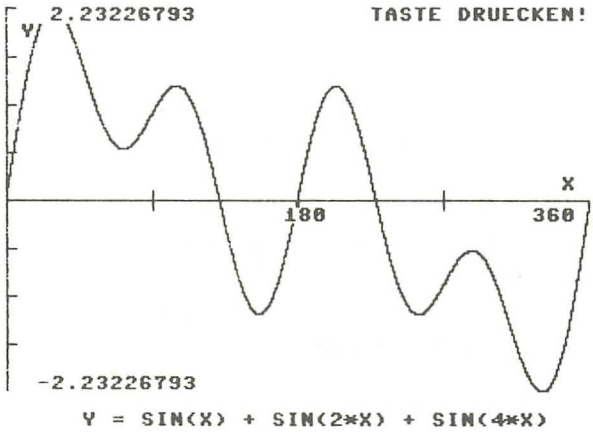


Bild 15.4: *Funktionskurve*

Sie können unser Programm natürlich noch verbessern. So wäre die Eingabe des unteren X-Wertes eventuell sinnvoll. Auch muß Z nicht unbedingt bei 0 beginnen. Es wäre dann möglich, auch kleinste Ausschnitte einer Funktionskurve darzustellen.

15.13 Beispielprogramm 3D-FUNKTION

Eine weitere Möglichkeit, Funktionen grafisch darzustellen, zeigen wir Ihnen jetzt. Hierbei wird die Funktion dreidimensional dargestellt. Zur X- und Y-Achse kommt also noch die Z-Achse hinzu. Die Z-Achse gibt die »Tiefe« unserer Darstellung an.

Das folgende Programm basiert auf einer Veröffentlichung in der Zeitschrift 64'er, Ausgabe 4/85. Wir haben das Programm für den Plus/4 umgeschrieben. Es ist aller-

dings nicht so komfortabel wie das vorherige Programm. Dafür ist es aber sehr kurz und kann daher schnell eingetippt werden.

```
10 REM 3D-FUNKTION
20 XA=50:YA=20:ZA=20
30 COLOR0,1
40 COLOR1,2
50 GRAPHIC1,1
60 FORXH=0TO10:FORYH=0TO10STEP.05
70 X=XH:Y=YH:GOSUB90:X=YH:Y=XH:GOSUB90:NEXTYH,XH
80 GETKEYA$:GRAPHIC0:END
90 Z=5+5*(SIN(X/10)*SIN(X/5)-SIN(Y/5))
100 D=XA-X:IFD=0THEN140
110 YS=15.7*(Y-X*(YA-Y)/D)+120:ZS=(-20)*(Z-X*(ZA-Z)/D)+150
120 IFYS<0ORYS>319ORZS<0ORZS>199THEN140
130 DRAW,YS,ZS
140 RETURN
```

Bei diesem Programm steht die Funktion in Zeile 90. Allerdings muß hier darauf geachtet werden, daß Z Werte von etwa -5 bis 5 erhält. Da die Berechnung ziemlich lange dauert, kann zum Testen einer Funktion in Zeile 60 ein größerer Wert hinter der STEP-Anweisung angegeben werden.

```
60 FORXH=0TO10:FORYH=0TO10STEP.5
```

Die Werte in Zeile 20 geben die gewünschte Perspektive an. Auch dazu ein Beispiel:

```
20 XA=50:YA=30:ZA=40
```

Hier sind drei Beispiele für die Zeile 90:

```
90 Z=5+2*(SIN(X/5)-SIN(Y/2))
90 Z=5+5*(SIN(3.14/10*X)*SIN(6.28/10*Y))^11
90 Z=5-2*(SIN(3.14/10*X)*SIN(3.14/10*Y))^8
```

Wir zeigen Ihnen noch zwei fertige Bilder.

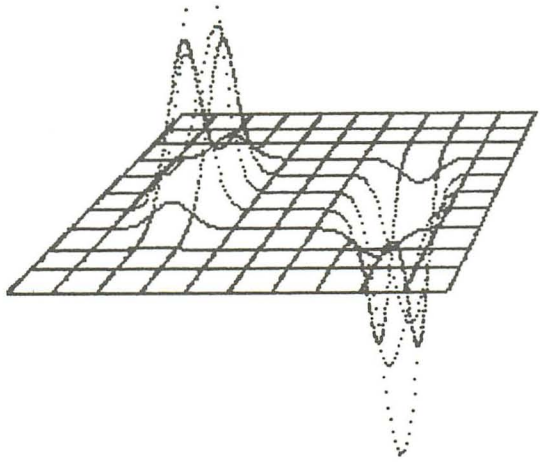


Bild 15.5: *Beispiel*

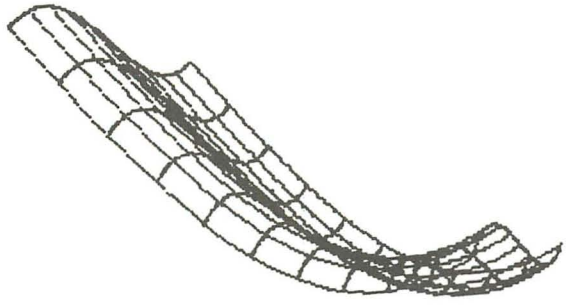


Bild 15.6: *Beispiel*

16 Disketten-Programmierung

Sind Sie glücklicher Besitzer eines Disketten-Laufwerkes? Wenn ja, dann freuen Sie sich, daß Sie die langen Wartezeiten beim Laden und Speichern mit dem Kassettenrecorder nicht mehr haben. Lesen Sie aber trotzdem dieses Kapitel, damit Sie auch richtig mit Ihrem Gerät umgehen. Sollten Sie dagegen kein Disketten-Laufwerk (auch kurz »Floppy« genannt) besitzen, empfehlen wir Ihnen gerade dieses Kapitel. Vielleicht ist ja auf Ihrem Konto einmal ein Betrag »über«, den Sie zur Anschaffung eines Laufwerks verwenden können. Sie würden das sicher nicht bereuen!

Über die Bedienung und die Programmierung der Floppy könnte man ein ganzes Buch schreiben. Deshalb ist dieses Kapitel auch etwas umfangreicher. Leider können wir nicht verhindern, daß jetzt eine große Menge Spezialbegriffe auf Sie zukommt. Wir werden Ihnen auf den nun folgenden Seiten alles Wissenswerte über dieses interessante und nützliche Gerät näherbringen.

16.1 Was sind Disketten?

Die Bezeichnung »Floppy-Disk« kommt aus dem Englischen und heißt übersetzt etwa »schlappe Scheibe«. Es handelt sich dabei um eine runde Kunststoffolie, die mit einem magnetischen Material beschichtet ist. Der Aufbau ist also ähnlich einem Tonband, nur ist hier die Folie dicker. Die Kunststoffscheibe ist in einer etwas festeren Kunststoffhülle untergebracht, um sie vor Verschmutzung und mechanischen Schäden zu schützen. Außerdem gibt diese Hülle unserer »schlappen« Scheibe eine gewisse Stabilität.

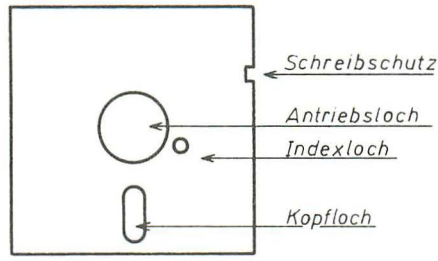


Bild 16.1: *Aufbau einer Diskette*

An der Seite der Diskettenhülle befindet sich eine Kerbe. Wenn diese Kerbe zugeklebt ist, kann die Diskette nur noch gelesen werden, ein Abspeichern von Daten ist dann nicht mehr möglich. Es handelt sich dabei also um einen Schreibschutz. In der Mitte befindet sich das Antriebsloch. Die Diskette muß in der Hülle gedreht werden, damit die Daten gelesen werden können. Dafür ist im Laufwerk ein Motor vorhanden. Durch dieses Antriebsloch wird die Diskette im Laufwerk zentriert und angetrieben. Beim Kauf von Disketten sollten Sie darauf achten, daß die Diskette im Antriebsloch einen Verstärkungsring aufweist. Die mechanische Beanspruchung beim Zentrieren und Antreiben ist recht hoch, daher ist so ein Verstärkungsring eine sinnvolle Sache. Das Indexloch ist nur für bestimmte Disketten-Laufwerke gedacht. Unser Commodore-1541- oder 1551-Laufwerk benötigt dieses Indexloch nicht. Mit einem Indexloch läßt sich feststellen, wo die Diskette »anfängt«. Unten auf der Diskette sehen Sie das sogenannte Kopfloch. Durch dieses Fenster in der Kunststoffhülle wird die Diskette gelesen und beschrieben.

Ganz wichtig:

Fassen Sie niemals auf die magnetische Beschichtung der Diskette! Der Schreib-Lesekopf würde dadurch verschmutzen, und die einwandfreie Funktion des Laufwerkes ist nicht mehr gewährleistet. Bewahren Sie Ihre Disketten auch immer in den dafür vorgesehenen Hüllen an einem möglichst staubfreien Ort auf. Das Laufwerk wird dann nicht so sehr durch Staub beansprucht. Im übrigen teilen wir die Auffassung anderer Leute nicht, das 1541-Disketten-Laufwerk wäre unzuverlässig und störungsanfällig. Wenn man das Laufwerk etwas pflegt und nicht allzusehr verstauben läßt, wird es jahrelang fehlerfrei funktionieren. Die Autoren haben jedenfalls keine schlechten Erfahrungen mit dem Laufwerk gemacht.

Wenn Sie Fragen zur Pflege Ihres Laufwerkes haben, z. B. Reinigen des Gerätes und des Schreib-Lesekopfes oder Schmieren der Führungsschienen, fragen Sie einfach einen Bekannten, der Erfahrung mit dem Laufwerk hat. Oder lesen Sie die Literatur über das 1541-Laufwerk. Bedenken Sie aber, daß das Gehäuse Ihres Laufwerkes während der Garantiezeit nicht geöffnet werden darf.

Das Floppy-Laufwerk

Zum Lesen bzw. Beschreiben der Disketten benötigen Sie natürlich das Disketten-Laufwerk. In ihm ist der Antriebsmotor für die Disketten untergebracht. Außerdem enthält das Laufwerk den Schreib-Lesekopf, der auf Schienen beweglich angebracht ist und durch einen Schrittschaltmotor bewegt wird. Im 1541- und 1551-Laufwerk ist außerdem noch eine umfangreiche Elektronik untergebracht. Es handelt sich dabei um einen kompletten Computer, der die Motoren und die Verarbeitung der Daten steuert.

Der Computer im Laufwerk hat, genau wie der Plus/4, ein ROM. In diesem Speicher ist das sogenannte DOS (Disk Operating System) gespeichert, ein Programm, das das Laufwerk erst zum Leben erweckt.

Das 1551-Laufwerk

Speziell für den Plus/4, den C16 und den C116 wurde die Floppy 1551 entwickelt. Diese hat gegenüber dem 1541-Laufwerk einige Vorteile. Sie ist nämlich preiswerter und, der größte Vorteil, sie ist etwa viermal schneller als die 1541. Das 1551-Laufwerk wird am Expansion-Port angeschlossen. Die Datenübertragung erfolgt parallel.

Beide Laufwerke, 1551 und 1541 verfügen über den gleichen Befehlssatz. (Die 1551 besitzt jedoch zusätzlich 2 interessante und nützliche Befehle mehr.) Disketten, die auf dem einen Laufwerk beschrieben wurden, können also ohne weiteres auf dem anderen Laufwerk benutzt werden.

16.2 Disketten-Laufwerk kontra Kassettenrecorder

Kassettenrecorder haben einen großen Vorteil gegenüber Disketten-Laufwerken, sie sind nämlich sehr viel billiger. Das war's auch schon; mehr Vorteile gibt es nicht, dafür aber eine ganze Reihe Nachteile.

Der größte Nachteil ergibt sich aus dem Prinzip der Datenspeicherung auf Kassetten. Hier werden die Daten sequentiell, d.h. nacheinander, aufgezeichnet. Sie werden sicher einsehen, daß es sehr umständlich ist, Teile dieser Daten zu lesen. Es muß entweder das Band zufällig auf der gewünschten Stelle stehen, oder das Band muß vorgespult werden bzw. alle anderen Daten müssen überlesen werden. Das dauert natürlich eine Weile. Bei den Disketten sieht das ganz anders aus: Die Daten werden zwar auch gewissermaßen sequentiell gespeichert, aber der Zugriff auf ganz bestimmte Teile der Daten ist viel einfacher. Der Schreib-Lesekopf geht einfach an die gewünschte Stelle, und die Daten werden gelesen. So etwas ist erstens durch die Bauart der Disketten möglich, denn der gesamte zur Verfügung stehende Speicherbereich liegt »offen« auf

einer Scheibe. Bei Kassetten sind die Daten auf einem langen Band, und man muß erst zur gewünschten Stelle vor- oder zurückspulen. Der zweite Grund des schnelleren Zugriffs auf die Daten ist folgender: Auf jeder Diskette ist ein Inhaltsverzeichnis abgespeichert. Dadurch »weiß« unser Disketten-Laufwerk immer, wo die Daten zu finden sind.

Ein weiterer Vorteil der Disketten gegenüber Kassetten ist die höhere Datenübertragungsgeschwindigkeit bei den Disketten. Wenn Sie ein 10 Kbyte langes Programm auf eine Kassette speichern, vergehen über 3 Minuten. Die Speicherung von 10 Kbyte auf Diskette dauert dagegen nur etwa 10 bis 30 Sekunden. Die Verarbeitungsgeschwindigkeit der Diskette ist etwa 10 bis 40 mal schneller als bei Kassetten.

Und noch ein Vorteil der Disketten:

Mit dem Disketten-Laufwerk ist es möglich, sogenannte »relative Dateien« zu erstellen. Bei diesen Dateien sind die einzelnen Datensätze durchnumeriert. Denken Sie zum Vergleich einmal an die Variablen. Sie können jederzeit die Variable A(55) ansehen. So ist es auch bei den relativen Dateien. Wenn Sie also einen ganz bestimmten Datensatz lesen möchten, wird einfach die entsprechende Nummer an das Disketten-Laufwerk übergeben, das Laufwerk positioniert nun den Schreib-Lesekopf auf die Stelle, wo der Datensatz abgespeichert ist und liest ihn aus. So etwas ist bei dem Kassettenrecorder unmöglich.

Wenn Sie ernsthaft mit Ihrem Computer arbeiten möchten, kommen Sie nicht umhin, ein Disketten-Laufwerk zu kaufen. Für den Anfänger dagegen reicht ein Kassettenrecorder aus.

16.3 Filetypen

Sie werden in diesem Kapitel immer wieder mit dem Begriff »File« konfrontiert. Damit Sie auch wissen, worum es sich dabei handelt, hier eine kurze Erklärung: Eine beliebige Anzahl zusammengehöriger Daten wird File genannt. File heißt übersetzt Datei. Wenn Sie also ein Programm in Ihrem Plus/4 haben und dieses Programm auf einen Datenträger, Kassette oder Diskette abspeichern, so handelt es sich dabei um ein File, in diesem speziellen Fall um ein Programmfile. Im Inhaltsverzeichnis Ihrer Diskette wird dieses File mit den Buchstaben PRG gekennzeichnet. Maschinenprogramme sind ebenfalls Programmfiles. Es gibt aber noch andere Filetypen. Sollen z. B. von einem Programm aus Daten abgespeichert oder geladen werden, so bieten sich dafür die sogenannten »sequentiellen Files« an. Auf dem Disketten-Laufwerk können Sie außerdem auch die »relativen Files« anlegen.

Ein sequentielles File (sequentiell = nacheinander) kann nur komplett geladen oder gespeichert werden. Zusätzlich ist es möglich, bei bestehenden Files am File-Ende

noch weitere Daten anzuhängen. Sequentielle Files werden im Disketten-Inhaltsverzeichnis mit den Buchstaben SEQ gekennzeichnet. Diese Files eignen sich z. B. für Adreßlisten.

Bei relativen Files ist es möglich, einzelne Daten dieser Datei von der Diskette einzulesen, bzw. einzelne Daten auf der Diskette zu ändern. Diese Files werden mit den Buchstaben REL gekennzeichnet. Der Zugriff auf einzelne Daten ist bei diesen Dateien wesentlich schneller, da hier nicht, wie bei sequentiellen Dateien, der ganze Datensatz bis zu dem Punkt, den man laden möchte, überlesen werden muß. Die relativen Dateien eignen sich ganz ausgezeichnet für Tabellen. Die Länge der relativen Datei ist nur durch die Diskettenkapazität begrenzt. Die Datei kann also einen Umfang von ca. 160 Kbyte haben, und auf jede Stelle dieser Datei kann in sehr kurzer Zeit zugegriffen werden.

Jedes File muß, wenn es auf Diskette gespeichert oder gelesen wird, »geöffnet« werden. Das Disketten-Laufwerk muß also durch einen bestimmten Befehl für die Datenübertragung vorbereitet werden. Dieser Befehl heißt OPEN.

16.4 Der Befehl OPEN

Dieser Befehl dient dem Öffnen (Vorbereiten) der Daten und Befehlskanälen Ihres Disketten-Laufwerks oder anderer Ein- und Ausgabegeräte.

Syntax:

OPEN f,g,k

f Filenummer, beliebige Nummer von 1 bis 127. Nach dem OPEN-Befehl müssen im Computer alle Befehle, die mit dem File zusammenhängen, mit dieser Filenummer gekennzeichnet werden.

g Gerätenummer. Die Datei kann nicht nur auf Kassettenrecorder oder Disketten-Laufwerk geöffnet werden, sondern auch zum Beispiel auf dem Bildschirm. Um das zu unterscheiden, gibt es diese Gerätenummer.

k Sekundäradresse. Im Zusammenhang mit dem Disketten-Laufwerk wird hier von der Kanalnummer gesprochen.

16.4.1 Die Gerätenummer

0 = Tastatur
1 = Kassettenrecorder
2 = RS-232-Schnittstelle
3 = Bildschirm
4 = Drucker
8 = Diskettenlaufwerk

Sie sehen, es ist also z. B. möglich, eine Ausgabe der Daten statt auf Disketten auf dem Drucker erfolgen zu lassen. Die Tastatur und der Bildschirm werden vom Betriebssystem schon automatisch als Ein- bzw. Ausgabegerät eingesetzt.

Trotzdem können Sie ohne weiteres von einem Programm aus mit dem OPEN-Befehl Daten zum Bildschirm bringen. Wenn es Ihnen auch unverständlich erscheint, denn die Ausgabe auf dem Bildschirm läßt sich doch viel einfacher mit dem PRINT-Befehl bewerkstelligen. Wenn von einem Programm die Ausgabe wahlweise auf dem Drucker oder auf dem Bildschirm erfolgen soll, so kann für diese Ausgabe ein und dieselbe Ausgaberoutine benutzt werden, indem im OPEN-Befehl einmal 3 und einmal 4 als Gerätenummer eingesetzt wird.

16.4.2 Die Sekundäradresse

Die Sekundäradresse wird z. B. bei dem Kassettenrecorder gebraucht.

OPEN 1,1,0,"Filenamen"	Eine Datei zum Lesen öffnen.
OPEN 1,1,1,"Filenamen"	Eine Datei zum Schreiben öffnen.
OPEN 1,1,2,"Filenamen"	Eine Datei zum Schreiben öffnen, am Ende dieser Datei wird eine End-of-Tape-Meldung geschrieben.

End of Tape bedeutet, daß auf dieser Kassette keine weiteren Files folgen. Der Computer »merkt« daran also, daß die Kassette zu Ende ist. Bei dem Diskettenlaufwerk ist die Sekundäradresse (0 bis 15) gleich der sogenannten Kanalnummer. Die Sekundäradressen 0 und 1 werden vom Betriebssystem Ihres Plus/4 für LOAD und SAVE verwendet. Die Sekundäradresse 15 ist für den Befehlskanal des Diskettenlaufwerks reserviert. Sie können also die Sekundäradressen von 2 bis 14 verwenden.

Einige Beispiele:

OPEN1,8,2,"TELEFONNUMMERN,S,W"	Öffnet eine sequentielle Datei, gekennzeichnet durch das »S«, zum Schreiben, gekennzeichnet durch das »W« (W = write, schreiben).
OPEN1,8,10,"LISTE,S,R"	Die sequentielle Datei mit dem Namen »LISTE« wird zum Lesen (R = read, lesen) geöffnet.
OPEN1,8,2,"TELEFONNUMMERN,S,A"	Die sequentielle Datei »TELEFONNUMMERN« wird um weitere Daten verlängert (A = append, anfügen).

Wir haben nun gesehen, wie eine Datei geöffnet wird. Nach dem Arbeiten mit der Datei muß diese wieder geschlossen werden. Dazu dient der CLOSE-Befehl.

16.5 Der Befehl CLOSE

Jede Datei muß wieder geschlossen werden. Das ist besonders wichtig, wenn Sie mit dem Diskettenlaufwerk arbeiten. Nicht ordnungsgemäß geschlossene Files werden im Disketten-Inhaltsverzeichnis mit einem Stern gekennzeichnet. Hinter CLOSE muß, wie bei allen Befehlen, die mit Dateien arbeiten, die Filenummer stehen.

16.6 PRINT#, GET#, INPUT#, CMD

So, wir haben unsere Datei nun geöffnet. Wie bekommen wir denn nun unsere Daten dort hinein? Genauso, wie wir Daten auf dem Bildschirm mit PRINT ausgeben können, können Daten auch mit dem PRINT-Befehl in die Datei übergeben werden. Der PRINT-Befehl muß dazu die Filenummer enthalten:

PRINT#f,Daten

Der Befehl ist nicht identisch mit dem normalen PRINT-Befehl. Man darf diesen Befehl auch nicht mit dem Fragezeichen abkürzen. Das Doppelkreuz ist das Kennzeichen für Files. Daher nennt man diesen Befehl einfach »Print-File«.

Dazu ein Beispiel:

```
10 OPEN1,8,2,"ADRESSEN,S,W"
20 INPUT"ADRESSE";A$
30 PRINT#1,A$
40 INPUT"WEITER (J/N)";JN$
50 IFJN$="J"THENGOTO20
60 CLOSE1
```

Mit diesem Programm können Sie Adressen in eine sequentielle Datei speichern. Jetzt werden wir die gespeicherten Daten wieder lesen:

```
10 OPEN1,8,2,"ADRESSEN,S,R"
20 INPUT#1,A$
30 PRINTA$
40 IFST=64THENCLOSE1:END
50 GOTO20
```

Die Systemvariable ST nimmt, wenn alle Daten gelesen sind, den Wert 64 an. Somit kann auf einfache Weise überprüft werden, ob das Programm beendet werden kann.

Die Daten werden also mit INPUT# gelesen. Auch hier wieder die Kennzeichnung mit dem Doppelkreuz. Statt INPUT# kann man auch den GET#-Befehl verwenden, Sie erinnern sich, mit GET können Sie einzelne Zeichen eingeben. So ist das auch bei GET#. Allerdings müssen Sie daran denken, daß die Daten ja eventuell auch aus mehreren Zeichen bestehen können. Wenn also GET# verwendet wird, muß erkannt werden, wann der Datensatz zu Ende ist. Das ist auch ganz einfach, denn beim Abspeichern der Daten mit PRINT# wird nach jedem Datensatz ein Carriage-Return-Zeichen, CHR$(13), abgespeichert.

```
10 OPEN1,8,15,"ADRESSEN,S,R"
20 GET#1,X$
30 IFX$=CHR$(13)THENPRINTA$:A$="":GOTO20
40 A$=A$+X$
50 IFST=64THENCLOSE1:END
60 GOTO20
```

Nach jedem gelesenen Zeichen wird einfach überprüft, ob es sich um das Zeichen für RETURN handelt. Wenn ja, werden alle Zeichen ausgegeben, die Variable wird wieder gelöscht, und das nächste Zeichen wird mit GET# gelesen. Wenn die Systemvariable ST den Wert 64 enthält, sind alle Daten ausgelesen. Dann wird die Datei wieder geschlossen und das Programm beendet.

Der Befehl CMD

Mit dem CMD-Befehl werden Ausgaben, die sonst auf den Bildschirm gehen, auf das mit OPEN bestimmte Gerät umgeleitet.

Schreibweise für CMD:

CMD f

f = Filenummer, die auch bei OPEN und CLOSE verwendet wird.

Beispiel:

```
OPEN1,4
CMD1
LIST
PRINT#1
CLOSE1
```

Diese Befehlsfolge sendet das Listing eines BASIC-Programms auf den Drucker. Mit PRINT# wird CMD wieder ausgeschaltet. Der CMD-Befehl erleichtert z. B. die Ausgabe von Text auf den Drucker. Es ist auch möglich, durch einfaches Ändern der Gerätenummer beim OPEN-Befehl, dieselbe Ausgaberoutine für verschiedene Ausgabegeräte zu nutzen.

```
OPEN2,8,2,"PROGRAMM,S,W"
CMD2
LIST
PRINT#2
CLOSE2
```

Diese Befehlsfolge schreibt ein Programmlisting als sequentielles File auf Diskette. Als sequentielles File kann es somit z. B. von einem Textverarbeitungsprogramm geladen werden.

16.7 Jokerzeichen

Zwei besondere Zeichen erleichtern das Arbeiten mit der Floppy. Es sind das Fragezeichen und der Stern. Diese Zeichen können im Filenamen für beliebige andere Zeichen eingesetzt werden. Das Fragezeichen ersetzt ein beliebiges Zeichen und der Stern ersetzt mehrere beliebige Zeichen. Das ist z. B. eine Erleichterung, wenn mehrere Files mit ähnlichem Namen gelöscht werden sollen.

Einige Beispiele:

FILE? kann z. B. heißen: **FILE1** oder **FILE2** oder **FILEA** usw.

??????SPIEL kann heißen: **KARTENSPIEL** oder **EXTRA-SPIEL**

Wird das Fragezeichen beim Laden eines Programms verwendet, so wird das erste Programm geladen, das mit dem vorgegebenen »Muster« übereinstimmt:

DLOAD"ADRESSEN?"

lädt das erste Programm, das neun Stellen lang ist und mit den Buchstaben »ADRESSEN« beginnt.

SPIEL* kann stehen für: **SPIELPROGRAMM** oder **SPIELE** oder **SPIEL EINS** usw.

DLOAD"*"

lädt das erste Programm von der Diskette mit beliebigem Programmnamen.

DLOAD"SPIEL*"

lädt das erste Programm von Diskette, dessen Namen mit »SPIEL« beginnt.

Stern und Fragezeichen können auch kombiniert werden:

DLOAD"??????SPIEL*"

lädt das erste Programm von Diskette, dessen Name mit sechs beliebigen Buchstaben beginnt, dann die Buchstaben »SPIEL« enthält, und dann beliebige Zeichen bis zur maximalen Länge (16 Zeichen) enthält. Die Jokerzeichen sind besonders gut einzusetzen beim Löschen von Files und beim Auflisten des Directorys (Inhaltsverzeichnis).

Der Klammeraffe (@)

Dieses Zeichen ermöglicht es, Files zu überschreiben. Nehmen wir an, Sie hätten das Programm mit dem Namen »PLOTTEN« verbessert oder geändert, und Sie möchten es nun unter dem gleichen Namen wieder abspeichern. Dann müssen Sie folgendes eingeben:

DSAVE"@PLOTTEN"

Durch den Klammeraffen vor dem Namen wird das alte Programm mit dem neuen überschrieben. Wenn Sie sequentielle Files überschreiben möchten, so brauchen Sie ebenfalls dieses Zeichen.

Weiterführende Literatur

Wir haben über die sehr komfortable relative Datei berichtet. Leider ist die Programmierung einer solchen Datei nicht so einfach. Auf jeden Fall würde eine Beschreibung dieses Dateityps den Rahmen dieses Buches sprengen. Wir verweisen daher auf die Bücher, die sich mit der Floppy 1541 beschäftigen. Ein ausgezeichnetes Buch dazu finden Sie im Markt&Technik Verlag. Ebenso verhält es sich mit den Direktzugriffsbefehlen. Es ist unmöglich, in einem Plus/4-Handbuch das komplette Diskettenlaufwerk ausführlich zu beschreiben. Wir beschränken uns daher hier auf das Wesentliche.

16.8 Die Diskettenbefehle

Der Plus/4 erleichtert das Arbeiten mit dem Diskettenlaufwerk durch eine Reihe von BASIC-Befehlen:

HEADER	DIRECTORY
DSAVE	DLOAD
BACKUP	COLLECT
COPY	SCRATCH
VERIFY	

Zusätzlich zu den speziellen Diskettenbefehlen brauchen wir noch weitere Befehle zum Umgang mit dem Diskettenlaufwerk:

OPEN	PRINT#
GET#	INPUT#
CMD	CLOSE

Das DOS im Diskettenlaufwerk beinhaltet eigene Systembefehle:

NEW	SCRATCH
RENAME	COPY
INITIALIZE	VALIDATE

Diese Systembefehle sind nicht zu verwechseln mit den BASIC-Befehlen, wenngleich sie teilweise dasselbe bewirken. Sie müssen die Systembefehle direkt an das Laufwerk senden. Zur Übermittlung der Systembefehle brauchen wir den BASIC-Befehl OPEN.

Schreibweise:

OPEN f,g,15,"Befehl:Filenamen..."

f	beliebige (1 bis 127) Filenummer, mit der nach dem OPEN-Befehl immer das Gerät 8 (Diskettenlaufwerk) angesprochen wird.
g	Gerätenummer des Diskettenlaufwerks. Diese Nummer ist fest im Laufwerk eingebaut und kann nur durch einen Umbau geändert werden. Das ist eventuell nötig, wenn Sie zwei Laufwerke angeschlossen haben. Bei einem normalen 1541- oder 1551-Laufwerk ist diese Nummer eine »8«.
15	Der sogenannte Befehls-Kanal des Diskettenlaufwerks.
"Befehl:Filename..."	Der gewünschte Systembefehl kann voll ausgeschrieben werden oder auch abgekürzt werden, indem nur der erste Buchstaben eingegeben wird. Nach dem Befehl muß ein Doppelpunkt stehen, wenn ein oder mehrere Filenamen folgen. Hinter dem Doppelpunkt wird die dazugehörige Information, z. B. der File-Name, angegeben.

Wir werden bei den einzelnen Befehlen die Schreibweise des OPEN-Befehls jeweils mit angeben.

Befehl HEADER

Wenn Sie eine neue Diskette kaufen, muß diese Diskette für unser Laufwerk vorbereitet werden. Es muß ein bestimmtes Grundmuster, ein »Format«, auf der Diskette abgespeichert werden. Man spricht hierbei auch vom Formatieren. Die so behandelten Disketten nennt man »softsektorierte« Disketten.

Schreibweise des Befehls:

HEADER "Diskettenname",Dx,Iy(,(ON)Ug)

Diskettenname	Maximal 16 Zeichen (Buchstaben und Ziffern) langer Name für die Diskette.
x	Laufwerknummer. D0, wenn Sie nur ein Laufwerk angeschlossen haben. Sind zwei Laufwerke angeschlossen, steht D0 für das erste, und D1 für das zweite Laufwerk.
y	Identifizierungs-Kennzeichen der Diskette. ID besteht aus zwei beliebigen Zeichen, die zusätzlich zum Diskettennamen auf der Diskette gespeichert werden.
ON g	Das Diskettenlaufwerk hat normalerweise die Geräteadresse 8. Wenn das Laufwerk eine andere Geräteadresse besitzt, kann an dieser Stelle die geänderte Adresse angegeben werden. Wird hier nichts angegeben, setzt der Computer automatisch die 8 ein. Statt ON g kann auch nur g im Befehl geschrieben werden.

HEADER formatiert also die Diskette. Dabei wird die Diskette aufgeteilt in Spuren, und die Spuren werden aufgeteilt in Sektoren. Die Diskette hat 35 Spuren. Die Spuren sind von außen nach innen von 1 bis 35 durchnumeriert. Bei den Spuren handelt es sich also um die 35 möglichen Positionen des Schreib-Lesekopfes. Die Kreisringe sind im Innern der Diskette kleiner, also ist auch der Umfang der Spur kleiner. Dementsprechend ist die Anzahl der Sektoren (Abschnitte auf der Spur) innen geringer als außen. Die Spuren 1 bis 17 haben jeweils 21 Sektoren, die Spuren 18 bis 24 haben 19 Sektoren, die Spuren 25 bis 30 haben 18 Sektoren und die Spuren 31 bis 35 haben 17 Sektoren.

Unsere Diskette wird in 683 Sektoren eingeteilt. Man bezeichnet die Sektoren auch als »Blöcke«. Für das Inhaltsverzeichnis werden einige Blöcke reserviert, ebenso für das sogenannte BAM, das ist ein Belegungsplan der Blöcke. Übrig bleiben für Ihre Programme 644 Blöcke. Jeder Block kann 256 Byte enthalten. 2 Byte davon sind reserviert für die Information, wo der nächste Block dieses Files zu finden ist. Es bleiben also 644 Blöcke mit jeweils 254 Byte, das sind zusammen über 163000 Byte.

Mit dem HEADER-Befehl wird aber nicht nur die Sektoreinteilung vorgenommen, sondern es wird auch das Inhaltsverzeichnis, das Directory, der Diskette angelegt. Außerdem wird der momentane Inhalt der Diskette gelöscht! Das ist eine sehr wichtige Eigenschaft dieses Befehls. Vergewissern Sie sich vor der Anwendung dieses Befehls genau, ob Sie auch wirklich die richtige Diskette im Laufwerk haben. Wenn Sie eine beschriebene Diskette neu formatieren, ist der Inhalt verloren. Aus diesem Grund wird der Befehl HEADER auch nicht sofort ausgeführt, sondern es erscheint auf dem Bildschirm:

ARE YOU SURE?

Wenn Sie diese Frage mit Y (für YES) beantworten, dann wird der Befehl ausgeführt.

Beispiele für HEADER:

HEADER"SPIELE",I12,D0

Formatiert eine Diskette und gibt ihr den Namen »SPIELE« und die ID »12«

HEADER"PROGRAMME",D0,U9

Formatiert eine Diskette auf dem Laufwerk 0 mit Geräteadresse 9. Die Diskette bekommt den Namen »PROGRAMME«. Da hier die ID-Angabe fehlt, bleibt die bisherige ID erhalten. Es wird also nur der Name geändert. Mit neuen, unformatierten Disketten können Sie das nicht machen, bei neuen Disketten müssen Sie immer eine ID eingeben.

HEADER ist ein BASIC-Befehl. Sie können aber auch einen System-Befehl zum Formatieren verwenden. Dann müssen Sie folgendes eingeben:

```
OPEN1,8,15,"N:SPIELE,12"
CLOSE1
```

Das bewirkt das gleiche wie unser erstes Beispiel mit HEADER.

```
OPEN1,9,15,"N:PROGRAMME"
CLOSE1
```

Dies ist eine Nachbildung des zweiten HEADER-Befehls. Beachten Sie aber, daß anschließend noch der CLOSE-Befehl eingegeben werden muß. Noch einige Anmerkungen zur Identifizierung der Disketten.

Die ID

Wir haben gesagt, daß die ID aus zwei beliebigen Zeichen bestehen kann. Sie sollten jedoch jeder Diskette eine andere ID geben. Das hat folgenden Grund: Das Diskettenlaufwerk hat auch ein eingebautes RAM. In dieses RAM wird der Blockbelegungsplan, also die BAM eingelesen. Hier wird die BAM auch berichtigt und auf den jeweils neuesten Stand gebracht und anschließend wieder auf die Diskette abgespeichert. Es gibt aber Diskettenbefehle, die die BAM nicht sofort wieder abspeichern. Wenn Sie nun zwischenzeitlich eine andere Diskette mit genau der gleichen BAM in

das Laufwerk einlegen, wird das Laufwerk mit der falschen BAM arbeiten. Das Laufwerk vergleicht nämlich die ID der Diskette mit der ID der gespeicherten BAM. Sind beide gleich, arbeitet das Disketten-Laufwerk falsch.

Der Befehl DIRECTORY

Mit diesem Befehl können Sie sich das Inhaltsverzeichnis der Diskette ansehen. Sie brauchen nur DIRECTORY einzugeben (oder drücken Sie die Taste F3), ohne Gerätenummer oder sonstige zusätzliche Informationen.

DIRECTORY (D0)(,U8)(,"Filename")

D0 und U8 kann weggelassen werden, wenn Sie nur ein Diskettenlaufwerk angeschlossen haben. Den Filenamen brauchen Sie nicht anzugeben, wenn das gesamte Inhaltsverzeichnis der Diskette ausgegeben werden soll. Ausgegeben wird der Name der Diskette, die ID und ein Kennzeichen der 1541-/1551-Laufwerke, nämlich 2A. Dann werden die Anzahl der von jedem Programm belegten Blöcke (die Sektoren), die Filenamen und die Filetypen ausgegeben.

Es gibt noch eine zweite Möglichkeit, das Directory der Diskette einzulesen. Bei dieser zweiten Möglichkeit wird, anders als bei dem Befehl DIRECTORY, das Inhaltsverzeichnis in den Plus/4-Speicher eingeladen. Das Directory wird also wie ein Programm geladen. Dazu müssen Sie eingeben:

```
LOAD"$",8
LIST
```

(Die 8 ist die Gerätenummer für das Diskettenlaufwerk.) Mit DLOAD»$« können Sie das Directory nicht einladen. Beachten Sie bitte, daß ein eventuell vorhandenes Programm im Speicher Ihres Plus/4 durch das Directory überschrieben und somit gelöscht wird.

Das Directory ist also unter dem Namen »$« auf der Diskette abgespeichert. Das Inhaltsverzeichnis wird natürlich bei jeder Änderung auf der Diskette automatisch vom DOS berichtigt.

Der Befehl DSAVE

Mit DSAVE können Sie Ihre Programme abspeichern.

Syntax:

DSAVE "Filename"(,D1)(,U9)

D1 und U9 können Sie wieder weglassen, wenn Sie nur ein Laufwerk angeschlossen haben. Der Filenamen, also der Programmnamen kann bis zu 16 Zeichen lang sein. Mit DSAVE können Sie nur BASIC-Programme abspeichern. Maschinenprogramme müssen Sie mit dem Maschinensprachemonitor abspeichern, oder über Umwege, indem Sie ein Programmfile eröffnen und die Programmdaten Byte für Byte zum Disketten-Laufwerk übertragen.

Der Befehl DLOAD

Syntax:

DLOAD "SPIELE"(,D1)(,U11)

Mit diesem Befehl wird das Programm mit dem Namen »SPIELE« vom Laufwerk 1, mit der Geräteadresse 11 in den Computer eingeladen. Wenn Sie nur ein Laufwerk angeschlossen haben, reicht der Befehl:

DLOAD "SPIELE"

Statt DLOAD können Sie auch folgenden Befehl eingeben:

LOAD "SPIELE",8

Wenn Sie ein Maschinenprogramm laden möchten, so geben Sie bitte ein:

LOAD "SPIELE",8,1

Nach dem Laden von Maschinenprogrammen sollten Sie den Befehl NEW eingeben. Mit NEW werden wichtige Speicherzellen in der Zero-Page wieder auf »normale« Werte gebracht. Wenn Sie dagegen NEW nicht eingeben, könnten Sie Fehlermeldungen bekommen, mit denen Sie nichts anzufangen wissen und die auch nicht berechtigt sind. Z.B. erscheint die Meldung »OUT OF MEMORY ERROR«. Das hat folgenden Grund: Nach dem Laden von Programmen wird die Endadresse dieser Programme auch gleichzeitig als Endadresse für den BASIC-Speicher eingesetzt. Wenn Sie nun z. B. eine Grafik von der Diskette einlesen, wird eine zu hohe Endadresse für das

BASIC (wenn der Grafikbildschirm durch GRAPHIC reserviert wurde) eingesetzt. Sie können den Befehl LOAD oder DLOAD auch in einem Programm verwenden. Dann wird das Programm geladen und automatisch gestartet.

Wenn Sie mit LOAD"Name",8,1 Maschinenprogramme laden, werden diese nicht automatisch gestartet, aber das BASIC-Programm startet erneut. Der Neustart läßt sich nicht verhindern. Das BASIC-Programm würde also wieder auf den Befehl LOAD"Name",8,1 stoßen und das Maschinenprogramm noch einmal laden. Man kann diesen Vorgang verhindern. Dazu ein Beispiel:

```
10 X=X+1
20 IFX=1THENLOAD"TASTATUR",8,1
30 ...
```

Auf den ersten Blick erscheint alles normal. Beim Start des BASIC-Programms werden ja alle Variablen auf Null zurückgesetzt. In unserem Beispiel wird dann zu X der Wert 1 addiert. In Zeile 20 wird nun, wenn X = 1 ist, das Maschinenprogramm nachgeladen. Nun haben wir gesagt, daß nach dem Einladen des Programms ein neuer Start des BASIC-Programms erfolgt. Dann müßte doch eigentlich auch die Variable X wieder auf Null gesetzt werden, mit 1 addiert und durch Zeile 20 erneut das Maschinenprogramm geladen werden.

Nun, zum Glück ist das nicht so, denn die Inhalte von Variablen werden in diesem Fall nicht gelöscht. Also hat X nach dem Einladen des Maschinenprogramms und dem Neustart des BASIC-Programms immer noch den Wert 1. Es wird nun wieder in Zeile 10 zum Inhalt von X eine 1 addiert. Nun hat X den Wert 2, und die Bedingung in Zeile 20 ist nicht erfüllt. Also wird unser Programm in Zeile 30 fortfahren.

Der Befehl BACKUP

Mit diesem Befehl lassen sich komplette Disketten kopieren. Das funktioniert allerdings nur bei Doppellaufwerken. Wenn Sie also nur ein einfaches 1541- oder 1551-Laufwerk besitzen, können Sie diesen Befehl gleich wieder vergessen.

Schreibweise für BACKUP

BACKUP Dx TO Dy (,(ON)Ug)

x Laufwerknummer des Laufwerks, in dem sich die Originaldiskette befindet

y Nummer des zweiten Laufwerks

g Gerätenummer, wenn eine andere Nummer als 8 gewünscht wird. Sie können auch ON Ug eingeben.

Die neue Diskette braucht übrigens nicht formatiert zu sein. Das erledigt BACKUP gleich mit. Wenn die neue Diskette dagegen schon formatiert ist, werden das Inhaltsverzeichnis und alle eventuell vorhandenen Daten auf dieser Disketten überschrieben! Seien Sie daher vorsichtig, wenn Sie diesen Befehl anwenden möchten. Vergewissern Sie sich immer, daß Sie auch die richtige Diskette im Laufwerk haben.

Der Befehl COLLECT

Nach längerem Gebrauch einer Diskette kann es passieren, daß die Summe der belegten und freien Blöcke im Directory nicht mehr die mögliche Gesamtzahl von 664 ergibt. Möglich ist dies z. B. durch nicht ordnungsgemäß geschlossene Files. Der Befehl COLLECT gibt diese belegten, aber nicht im Inhaltsverzeichnis angegebenen Blöcke wieder frei. Die Diskette wird also »bereinigt«. Sie werden nach Ausführung dieses Befehls im Directory wieder auf die Summe von 664 Blöcken kommen.

Syntax:

COLLECT (Dx) (,(ON)Ug)

x Laufwerknummer, entweder 0 oder 1. Wenn Sie nur ein Laufwerk angeschlossen haben, brauchen Sie hier keine Angabe zu machen.

g Gerätenummer, normal 8. Ug oder ON Ug brauchen Sie nur anzugeben, wenn Ihr Laufwerk eine andere Nummer besitzt.

Der Befehl COPY

Mit COPY lassen sich einzelne Files kopieren. Dies ist natürlich besonders sinnvoll, wenn die Files von einer Diskette auf eine andere kopiert werden, was allerdings nur mit Doppellaufwerken geht. Wenn Sie dagegen nur ein Einzellaufwerk besitzen, können Sie die Files nur mit geändertem Namen innerhalb einer Diskette kopieren.

Schreibweise von COPY:

COPY(Dx,)"Quelle"TO(Dy,)"Ziel"(,(ON)Ug)

x Laufwerknummer, des Laufwerks, in dem sich das zu kopierende File auf einer Diskette befindet.

y Laufwerknummer des zweiten Laufwerks.

g Gerätenummer, wenn diese nicht 8 ist.

Dazu einige Beispiele:

COPY D0,"NAMEN"TOD1,"NAMEN"

Kopiert das File »NAMEN« vom Laufwerk 0 auf das Laufwerk 1 und gibt dem File wieder den Namen »NAMEN«, Gerätenummer 8.

COPY "GRAFIK" TO "MALPROGRAMM"

Kopiert das File »GRAFIK« in eine Datei mit dem Namen »MALPROGRAMM«.

COPY D0 TO D1

Kopiert alle Files von Laufwerk 0 auf die Diskette in Laufwerk 1.

Der Befehl SCRATCH

Mit SCRATCH lassen sich einzelne Files auf einer Diskette löschen. Die belegten Blöcke werden wieder freigegeben.

Schreibweise für SCRATCH:

SCRATCH "Name"(,Dx)(,(ON)Ug)

"Name" Der Name des zu löschenden Files.

x Die Laufwerknummer des Laufwerks. Wenn Sie nur ein 1541-/1551-Laufwerk angeschlossen haben, können Sie diese Angabe weglassen.

g Geräteadresse, normalerweise 8. Wenn Sie z. B. nur ein 1541-/1551-Laufwerk angeschlossen haben, so hat dieses Laufwerk in der Regel die Geräteadresse 8. Dann können Sie Ug oder ON Ug weglassen.

Mit SCRATCH ist es möglich, innerhalb weniger Sekunden die Arbeit von Stunden oder Tagen zunichte zu machen. Sie müssen daher sehr vorsichtig mit diesem Befehl umgehen. Der Computer fragt aus diesem Grund auch nach der Eingabe dieses Befehls:

ARE YOU SURE?

Wenn Sie die Frage mit Y (für Yes) beantworten, wird der Befehl ausgeführt. Überlegen Sie sich aber genau, was Sie machen, bzw. mit diesem Befehl bewirken können.

Ein File, das mit diesem Befehl gelöscht wird, wird eigentlich garnicht »gelöscht«. Es wird nur im Directory der Diskette ein Byte geändert und im BAM der Diskette werden die durch das File belegten Blöcke wieder als unbelegt gekennzeichnet. Durch das geänderte Zeichen im Directory wird das File beim Auflisten des Inhaltsverzeichnisses nicht mehr aufgelistet. Wenn ein File aber einmal mit SCRATCH »gelöscht« wurde, kann es nur noch mit einem entsprechenden Programm, (z. B. Disk Monitor) gerettet werden.

Der Befehl VERIFY

Übersetzt heißt Verify »Prüfen«. Mit Verify kann ein Programm auf einer Diskette Byte für Byte mit dem Inhalt des Computerspeichers verglichen werden. Dadurch kann man feststellen, ob ein Programm auch wirklich fehlerfrei abgespeichert wurde. Wenn bei dieser Überprüfung ein Unterschied zwischen Rechnerspeicher und Diskette auftritt, wird die Fehlermeldung VERIFYING ERROR ausgegeben.

Schreibweise für VERIFY:

VERIFY "Name",g

g Gerätenummer Ihres Laufwerks, also normalerweise 8.

Wenn das Überprüfen fehlerfrei läuft, dann meldet der Computer:

VERIFYING OK

Sie sollten VERIFY verwenden, auch wenn das Überprüfen recht viel Zeit in Anspruch nimmt. Das Überprüfen dauert genauso lange wie das Einladen eines Programms. Wenn Ihr Laufwerk genauso gut funktioniert wie unsere Laufwerke, dann werden Sie wohl nie eine Fehlermeldung bekommen, wir haben das jedenfalls auch noch nicht erlebt. Wenn Ihnen diese Prozedur zu lange dauert, beschränken Sie sie wenigstens auf wichtige, lange Programme.

Sollte einmal eine Fehlermeldung erscheinen, dann machen Sie bitte folgendes: Speichern Sie Ihr Programm noch einmal ab, möglichst unter einem anderen Filenamen. Führen Sie noch einmal VERIFY durch. Wenn das Programm fehlerfrei ist, dann löschen Sie die erste Version dieses Programms auf Diskette.

16.9 Die Disketten-Systembefehle

Der System-Befehl NEW

Dieser Befehl bewirkt das gleiche, wie der BASIC-Befehl HEADER. Mit NEW werden die Disketten also formatiert.

Syntax:

```
OPEN f,g,15,"NEW:Name,id"
CLOSE 1
```

oder

```
OPEN f,g,15
PRINT#f,"NEW:Name,id"
CLOSE f
```

f — logische Filenummer, mit der nach dem OPEN mit den entsprechenden Befehlen (PRINT#f,GET#f usw.) immer das Gerät g angesprochen wird. Die Filenummer ist eine beliebige Nummer von 1 bis 127.

g — Gerätenummer, bei der 1541-/1551-Floppy normalerweise die 8.

id — Die Identifizierungsnummer der Diskette.

Wenn Sie die ID angeben, wird die Diskette völlig neu formatiert. Lassen Sie dagegen die ID weg, dann wird nur das Inhaltsverzeichnis der Diskette gelöscht (geht nicht bei neuen, unformatierten Disketten). Der Befehl NEW kann abgekürzt werden, indem nur ein N geschrieben wird.

Der System-Befehl SCRATCH

Dieser Befehl bewirkt das gleiche wie der BASIC-Befehl SCRATCH. Hiermit werden einzelne Files gelöscht.

Syntax:

OPEN f,g,15,"S:Name1,Name2..."
CLOSE f

Für f und g gilt das gleiche wie bei NEW beschrieben.

S Die Abkürzung für SCRATCH. Natürlich können Sie SCRATCH auch voll ausschreiben (aber wozu?).

Name1 Der Filenamen des Files, das gelöscht werden soll.

Wenn mehrere Files gelöscht werden sollen, dann schreiben Sie die Namen, durch Kommata getrennt, zusätzlich dazu. Beachten Sie bitte, daß ein Befehlsstring maximal 40 Zeichen lang sein darf. Eventuell müßten Sie einen zweiten SCRATCH-Befehl senden. Übrigens wird nicht das File selbst, sondern nur das Inhaltsverzeichnis der Diskette, also das Directory und die BAM, mit SCRATCH geändert.

Der System-Befehl RENAME

Mit RENAME können Sie den Namen eines Files auf Diskette ändern.

Syntax:

OPEN f,g,15,"R:neuer Name=alter Name"
CLOSE f

Statt R darf RENAME selbstverständlich voll ausgeschrieben werden. Der neue Filenamen muß zuerst angegeben werden, dann folgt, durch ein Gleichheitszeichen getrennt, der alte Filenamen. Statt der oben angegebenen Schreibweise können Sie auch folgendes eingeben:

OPEN f,g,15
PRINT#f,"R:neuer Name=alter Name"
CLOSE f

Der RENAME-Befehl wird nicht durch BASIC-Befehle unterstützt, Sie müssen daher immer die hier angegebene Schreibweise anwenden.

Der System-Befehl COPY

Diesen Befehl haben wir schon bei den Diskettenbefehlen des BASIC kennengelernt. Mit COPY lassen sich einzelne Files kopieren.

Syntax:

```
OPEN f,g,15,"C:neuer Name=alter Name"
CLOSE f
```

Dadurch wird auf einer Diskette ein File mit dem ersten Namen (neuer Name) angelegt, das identisch ist mit dem zweiten angegebenen File (alter Name).

Sie können mit diesem Befehl aber auch mehrere Files zu einem neuen File zusammenfassen. Das eignet sich zum Beispiel für sequentielle Files, also irgendwelche Daten, die zusammengefaßt werden sollen. Dann müssen Sie folgendes eingeben:

```
OPEN f,g,15,"C:neuer Name=alter Name 1, alter Name 2 ..."
CLOSE f
```

oder:

```
OPEN f,g,15
PRINT#f,"C:neuer Name=alter Name"
CLOSE f
```

Der System-Befehl INITIALIZE

Mit diesem Befehl veranlassen Sie das Disketten-Laufwerk, die BAM einer Diskette in den RAM-Speicher des Laufwerks zu laden. Lesen Sie bitte dazu den Abschnitt über die ID (Identifizierungsnummer der Disketten).

Syntax:

```
OPEN f,g,15,"INITIALIZE"
CLOSE f
```

oder:

```
OPEN f,g,15
PRINT#f,"INITIALIZE" (oder einfach abgekürzt "I")
CLOSE f
```

Der System-Befehl VALIDATE

Dieser Befehl bewirkt das gleiche, wie der BASIC-Befehl COLLECT. Es werden alle in der BAM als belegt gekennzeichneten Blöcke, die nicht zu ordnungsgemäß geschlossenen Files gehören, bzw. die als belegt gekennzeichneten Blöcke, die zu gar keinem File gehören, wieder freigegeben. Sie brauchen diesen Befehl nur anzuwenden, wenn die Gesamtzahl der im Directory angegebenen Blöcke nicht 664 ergibt. Überhaupt empfiehlt es sich natürlich, den BASIC-Befehl COLLECT zu verwenden. Wenn Ihnen schon ein starkes BASIC geboten wird, sollten Sie es auch anwenden. Vollständigkeitshalber hier aber doch die Schreibweise dieses Befehls, da Sie eventuell in fremden Programmen einmal auf VALIDATE stoßen könnten.

Syntax:

OPEN f,g,15,"V" (oder vollausgeschrieben "VALIDATE").
CLOSE f

oder

OPEN f,g,15
PRINT#f,"V"
CLOSE f

17 Direkter Speicherzugriff mit PEEK, POKE, WAIT

Mit diesen Befehlen haben Sie die Möglichkeit, den Speicherinhalt Ihres Plus/4 anzusehen oder zu ändern. Dabei dient der Befehl PEEK zum Auslesen des Speicherinhalts.

Syntax:

PEEK(Adresse)

Adresse Dezimaler Wert der Speicheradresse (0 bis 65535).

Ein Beispiel:

PRINT PEEK(34)

ergibt den Wert 130. Die angesprochene Speicherzelle ist ein Teil der sogenannten Zero Page (Seite Null), das ist ein Teil des RAM, das bei dem Plus/4 für wichtige Daten reserviert ist. Jetzt ändern wir den Inhalt einer Speicherzelle. Dazu brauchen wir den Befehl POKE.

Der Befehl POKE

POKE Adresse,Wert

Adresse Dezimaler Wert der Speicheradresse.

Wert Den Wert, dezimal geschrieben, den unsere angesprochene Speicherzelle annehmen soll.

Wir ändern nun den Inhalt einer Speicherzelle im BASIC-RAM.

POKE 7000,215

Wenn wir nun mit »PRINT PEEK(7000)« den momentanen Inhalt dieser Speicherzelle ansehen, erhalten wir den Wert 215. Es hat also geklappt. Probieren Sie noch folgendes aus (ein kleiner Tip, speichern Sie vorher das Programm ab, das Sie eventuell im Moment im Speicher haben):

POKE 46,0
DIM A(30)

Was passiert? Nichts, der Computer ist »abgestürzt«. Versuchen Sie ihn mit der RESET-Taste wieder zum Leben zu erwecken. Aber das funktioniert auch nicht. Ihnen bleibt nichts weiter übrig, als den Computer wieder neu zu starten. Dieses Beispiel soll Ihnen zeigen, was Sie mit dem POKE-Befehl »anrichten« können. Es ist eine Kleinigkeit, den Computer lahmzulegen. Seien Sie daher sehr vorsichtig mit diesem Befehl!

Geben Sie bitte folgendes Miniprogramm ein und starten Sie es mit RUN (Dieses Beispiel funktioniert nur, wenn keine Grafik aufgerufen wurde, da sonst der BASIC-Anfang an einer anderen Adresse liegt, und der POKE-Befehl somit keine Wirkung auf das Programm hat).

```
10 PRINT1000*5
```

(Geben Sie das Programm bitte genauso ein, wie es hier steht!). Natürlich erhalten Sie nach dem Programmstart den Wert:

5000

Jetzt geben Sie bitte folgende Zeile ein:

POKE4106,173

Starten Sie das Programm noch einmal, und Sie werden sich wundern, daß das Ergebnis nun plötzlich 200 lautet. Sie können sich das Programm ansehen, es lautet nicht mehr 5000*5, sondern 5000/5 (5000 dividiert durch 5). Unser POKE-Befehl hat also das Programm geändert. Das ist eine sehr ungewöhnliche Art, ein Programm zu ändern, aber denken Sie doch einmal daran, daß sich das Programm selbst ändern könnte. Vielleicht ein erster Schritt von künstlicher Intelligenz? Den POKE-Befehl können wir sehr gut gebrauchen, um z. B. im TED-Chip wichtige Register zu ändern. Ebenso ist es möglich, Daten statt in Variablen zu speichern, direkt in den Speicher zu »poken«. Variablen werden bei jedem Programmstart mit RUN gelöscht, der Speicher bleibt dagegen unverändert, vorausgesetzt Sie brauchen eine Speicherzelle, die nicht vom Betriebssystem benötigt wird. Für Ihre Daten können Sie z. B. die Speicheradressen 1630 bis 1771 benutzen, weil dieser Speicherbereich bei Ihrem Plus/4 frei ist.

PEEK und POKE können auch kombiniert werden. Das ist notwendig, wenn Sie in einer Speicherzelle nur ganz bestimmte Bits verändern möchten. Dann müssen Sie die Speicherzelle erst lesen, den Wert ändern und wieder in die Speicherzelle zurückschreiben. Schalten Sie den Bildschirm aus:

```
POKE65286,PEEK(65286)AND239
```

Der Computer ist aber noch in Betrieb. Mit unserer Befehlskombination haben wir den Inhalt der Speicherzelle 65286 mit dem Wert 239 UND-verknüpft und somit das Bit 4 auf 0 gesetzt. Dieses Bit zeigt an, ob der Bildschirm eingeschaltet ist (1), oder ob der Bildschirm ausgeschaltet ist (0).

Geben Sie jetzt »blind« ein:

```
POKE65286,PEEK(65286)OR16
```

Jetzt ist der Bildschirm wieder eingeschaltet, wir haben nur das Bit 4 wieder auf »1« gesetzt. Übrigens wird die Verarbeitungsgeschwindigkeit Ihres Plus/4 durch das Ausschalten des Bildschirms um etwa 30 Prozent schneller. Sollten Sie einmal ein Programm haben, in dem es auf schnelle Verarbeitung ankommt und Sie keine Bildschirmausgabe während der Rechenarbeit benötigen, können Sie unsere Befehlskombinationen gut anwenden.

Ein weiterer Befehl, mit dem Sie einen direkten Zugriff auf den Speicher haben, ist der WAIT-Befehl. Diesen Befehl haben wir uns schon im Kapitel 6 angesehen, daher hier nur kurz die

Syntax:

WAIT Adresse,Wert1(,Wert2)

Mit WAIT ist es möglich, ein Programm solange anzuhalten, bis eine bestimmte Speicheradresse einen bestimmten Wert annimmt. Lesen Sie bitte zum Befehl WAIT auch das Kapitel 6. Wir haben diesen Befehl in das Kapitel »Vergleichsbefehle« gebracht, weil ja bei WAIT ein Vergleich vorgenommen wird, ähnlich wie bei IF...THEN.

ACHTUNG! POKE und WAIT sind mit äußerster Vorsicht zu verwenden! Die Programme können sich »aufhängen«. Sie können Ihren Computer dann nur noch durch Drücken der RESET-Taste oder durch das Ausschalten und Wiedereinschalten zum Leben erwecken. Geben Sie einmal

WAIT134,0

ein! Jetzt hilft nur noch die RESET-Taste.

18 READ, DATA, RESTORE

Sollen vielen Variablen Werte zugewiesen werden, so sind diese drei Befehle eine große Hilfe. Fangen wir mit einen Beispiel an:

```
 10 FOR I=1 TO 10
 20 READ T(1)
 30 NEXT I
 60 FOR I=1 TO 10
 70 PRINT T(1)
 80 NEXT
100 DATA 1,2,3,4,5,6,7
110 DATA 8
120 DATA 9,10
```

Lassen wir dieses Programm ablaufen, dann werden die Zahlen 1 bis 10 auf dem Bildschirm ausgegeben.

18.1 Der Befehl READ

Syntax:

READ A

oder

READ A$, B, C, D$,....

Trifft der Rechner auf READ, so wird der Programmspeicher von oben nach unten nach DATA abgesucht. In unserem Beispiel steht DATA zuerst in der Zeile 100. Nun wird das erste Element nach DATA gelesen und der Variablen, die READ folgt, zu-

geordnet. Der Rechner merkt sich nun die Stelle, an der dieses Element stand. Beim nächsten READ wird die Suche nach einem weiteren Element von dieser Stelle fortgesetzt. Man nennt die Speicherzellen, in denen eine Adresse steht, auch Zeiger. READ setzt also den DATA-Zeiger immer um ein Element weiter. Wird kein Element mehr gefunden und existiert auch keine weitere DATA-Zeile, dann erscheint die Fehlermeldung »OUT OF DATA«.

READ dürfen auch mehrere Variablen folgen. Die Anzahl und Art der Elemente die nun zugeordnet werden, entspricht der Anzahl der Variablen. Erwartet READ einen numerischen Wert und ist das nächste Element ein String, so wird eine Fehlermeldung ausgegeben und das Programm abgebrochen. Sie müssen also aufpassen, daß immer das richtige Element übergeben wird.

18.2 DATA

DATA wird solange vom Rechner ignoriert, bis mit READ ein DATA eingelesen werden soll. Im Klartext: DATA-Zeilen dürfen an jeder Stelle des Programms stehen. Sie werden bei der Abarbeitung eines Programms übersprungen. Die einzelnen Elemente, die DATA folgen, müssen durch Kommata getrennt werden. Strings müssen zwischen Anführungszeichen stehen, wenn sie ein Komma oder ein Leerzeichen enthalten. Das Komma würde sonst fälschlicherweise als Trennzeichen zwischen zwei Elementen interpretiert werden. Das Leerzeichen würde überlesen werden.

18.3 Der Befehl RESTORE

Syntax:

RESTORE

RESTORE Zeilennummer oder Variable

Sollen DATA-Elemente mehrmals gelesen werden, kann mit RESTORE der DATA-Zeiger wieder im Programm aufwärts bewegt werden. Wird nur RESTORE eingegeben, wird der Zeiger wieder auf das erste DATA-Element des Programms gesetzt.

Mit »RESTORE Zeilennummer« kann der Zeiger auch auf eine bestimmte Zeilennummer gesetzt werden. Mit dieser Kombination läßt sich der Zeiger auch abwärts in einem Programm bewegen. Es ist darauf zu achten, daß die Zeilennummer wirklich existiert. Die Zeilennummer kann auch in einer Variablen enthalten sein, die RESTORE folgt.

Diese Befehle werden oft auch zum Poken von Programmen, die in Maschienensprache programmiert wurden, eingesetzt. Das hat den Vorteil, daß auch Programmierer ohne Maschinensprachkenntnisse Nutzen aus diesen sehr schnellen Programmen ziehen können. Dieses Poken wird ebenfalls über Schleifen ausgeführt. Um Fehlerausgaben und damit Programmabbrüche zu vermeiden, sollte das letzte Element besonders gekennzeichnet sein, um dann die Schleife zu beenden. Dies kann zum Beispiel -1 sein.

Wie man solche Schleifen für das Einlesen von DATAs programmiert, erfahren Sie im Kapitel über die Schleifenprogrammierung. Ein Beispiel ist auch im Kapitel über die Ausgabe von Zeichen vorhanden.

19 Fehlerbehandlung

Spätestens bei der Fehlersuche wird der Programmierstil offenbar. Wer ein übersichtliches Programm mit wenigen Programmverzweigungen und vielen REM-Zeilen geschrieben hat, wird nun für seine geringe Mehrarbeit entschädigt. Dieser Programmierung wird sicher auch eine sorgfältige Planung vorangegangen sein, die später die Fehlersuche ebenfalls erleichtert. Es wird wohl selten vorkommen, daß ein längeres Programm sofort fehlerfrei läuft. Die meisten Fehlermeldungen werden dann erst einmal »SYNTAX ERROR« sein. Mit der HELP-Taste kann die Zeile, in der ein Fehler auftrat, auf dem Bildschirm gelistet werden. Leider werden dann die Zeichen blinkend dargestellt und dieser Umstand macht die Fehlersuche sicher nicht leichter. Bei längeren Zeilen sollte noch ein »LIST Zeilennummer« folgen, um seinen Augen nicht zuviel zuzumuten.

Das BASIC des Plus/4 hält viele Fehlermeldungen für seinen Anwender bereit. Mit diesen Meldungen werden alle Eingabefehler und auch ein Teil der logischen Fehler erfaßt. Grobe Schnitzer in einem Programm lassen sich so sehr schnell erkennen und beseitigen. Eine Übersicht der Fehlermeldungen und deren Bedeutung finden Sie im Anhang A und im Anhang B. Es gibt aber auch Fehler, die ein Rechner nicht erkennen kann. So kann er am Ende des Programms nicht feststellen, ob das gelieferte Ergebnis Ihren Erwartungen entspricht. Dieser Art von Fehler ist oft eine unzureichende Planung des Programmes vorangegangen. Wie man ihnen auf die Spur kommt, wird jetzt erklärt.

Bringen Sie das Programm mit LIST auf den Bildschirm, und unterbrechen Sie das Listen mit CTRL/S. Gehen Sie nun Zeile für Zeile das Programm oder den als fehlerhaft vermuteten Programmteil durch. Versuchen Sie dabei dem Programmablauf zu folgen. Haben Sie den Fehler danach noch nicht entdeckt, so wiederholen Sie das

Ganze oder machen am besten mit dem nächsten Punkt weiter. Sollten Sie glücklicher Besitzer eines Druckers sein, dann lassen Sie sich am besten erst einmal das Programm ausdrucken.

Allen anderen empfehlen wir, sich wenigstens die wichtigsten Programmteile, Verzweigungen und Variablen zu notieren. Gehen Sie nun mit Hilfe Ihrer hoffentlich erstellten Planungsunterlagen den Programmablauf mit verschiedenen Eingaben durch. Überprüfen Sie, ob Sie in den verschiedenen Programmteilen zum gewünschten Ergebnis kommen. Viele Fehler werden jetzt schon gefunden.

Sollte sich immer noch ein Fehler im Programm befinden, geht die Suche weiter. Geben Sie nun hinter wichtigen Teilen den Befehl STOP ein. Das Programm wird dann an dieser Stelle unterbrochen, Sie können sich jetzt die Werte der wichtigsten Variablen ausgeben lassen. Überprüfen Sie nun, ob deren Inhalte richtig sind. Die Werte können Sie sich selbstverständlich auch durch das Programm ausgeben lassen. Stimmt der Variableninhalt, dann können Sie mit CONT das Programm fortsetzen. Sind Sie am Ende des Programms angelangt und können sich den Fehler immer noch nicht erklären, dann wird wohl die Logik nicht stimmen (eventuell Sprungfehler). Um diese zu finden, gibt es einen besonderen Befehl.

19.1 Die Befehle TRON und TROFF

Syntax:

TRON

Wird dieser Befehl eingesetzt, so zeigt Ihnen der Rechner immer die Zeilennummer an, die er gerade abarbeitet. Stehen mehrere Anweisungen, durch Doppelpunkte getrennt, in einer Zeile, so wird diese Zeilennummer der Anzahl entsprechend oft ausgegeben. Die Ausgabe der Zeilennummern wird so lange fortgesetzt, bis der Rechner auf TROFF trifft.

Syntax:

TROFF

TROFF schaltet also die Ausgabe der aktuellen Zeilennummer wieder ab.

Sie können beide Befehle an jeder Stelle eines Programms einsetzen oder auch im Direktmodus eingeben. Meist wird man aber der Ausgabe nicht so schnell mit den Augen folgen können, aber man kann das Programm ja mit der COMMODORE-Taste verlangsamen oder mit CTRL/S auch anhalten.

19.2 Fehlerbehandlung in einem Programm

Sie können in einem Programm auch auf die verschiedensten Fehler reagieren. Tritt ein Fehler auf, kann er in einer speziell programmierten Fehlerroutine eventuell sogar unschädlich gemacht werden.

19.2.1 Die Befehle TRAP und RESUME

Syntax:

TRAP Zeilennummer

Dieser Befehl sollte dort stehen, wo mit Fehlern zu rechnen ist. Diesem Befehl muß eine Zeilennummer folgen. Sie entspricht der Startzeile der Fehlerroutine. Bei TRAP wird keine Fehlermeldung ausgegeben, sondern zur angegebenen Adresse verzweigt. Der TRAP-Befehl arbeitet ähnlich GOSUB. Es gibt aber keine verschachtelten Sprünge, d.h., die Fehlerroutine muß selbst fehlerfrei sein. Der Rücksprungbefehl ebenfalls. Er heißt hier RESUME.

Syntax:

RESUME

oder

RESUME NEXT

oder

RESUME Zeilennummer

RESUME bewirkt den Rücksprung zu dem den Fehler verursachenden Befehl. Es wird nun versucht, diesen erneut abzuarbeiten. Schlägt dieser Versuch wieder fehl, so wird erneut zum Fehlerprogramm verzweigt, und das Spiel beginnt von vorn.

»RESUME NEXT« heißt für den Rechner, das Programm mit dem nächsten der Fehlerquelle folgenden Befehl fortzusetzen. Die Fehlerquelle wird ignoriert.

»RESUME Zeilennummer« gibt die Programmzeile an, bei der nach der Fehlerbehandlung das Programm weiterarbeiten soll.

Fehlervariablen

Um auf den aufgetretenen Fehler reagieren zu können, besitzt der Rechner geschützte Variablen.

ER	Allen Fehlermeldungen ist eine Fehlernummer zugeordnet. Die Nummer steht in dieser Variablen und kann ausgelesen werden.
EL	In dieser Variablen ist die Zeilennummer enthalten, in der der Fehler auftrat. Sie kann ebenfalls nur ausgelesen werden.
ERR$(ER)	In dieser Stringvariablen ist die durch ER definierte Fehlermeldung enthalten. Die Fehlermeldung kann nur ausgelesen werden.

Ein Beispiel:

```
 10 TRAP 100
 20 PRINT "GEBEN SIE EINE ZAHL VON 1 BIS 9 EIN"
 30 GETKEY A
 40 PRINT 100/A
 50 GOTO 20
100 IF ER = 0 THEN PRINT " FALSCHE EINGABE " :RESUME 20
110 PRINT ERR$ (ER) "FEHLER IN ZEILE"  EL
120 END
```

In Zeile 10 wird festgelegt, daß das Programm bei einem auftretenden Fehler zur Zeile 100 verzweigen soll. Wird als Zahl die Ziffer 0 eingegeben, erfolgte normalerweise die Ausgabe der Fehlermeldung »DIVISION BY ZERO ERROR IN 40«. Das Programm verzweigt nun aber zur Zeile 100. Die Fehlermeldung hat die Fehlernummer 20, darum erfolgt nun die Ausgabe unseres Textes, dann erfolgt der Rücksprung in Zeile 20. Das Programm wurde also nicht abgebrochen.

Erzeugen wir einmal einen Syntaxfehler und ändern dazu PRINT in Zeile 20 in RINT. Dann starten wir das Programm. Ein Syntaxfehler hat eine andere Fehlernummer, darum erfolgt in Zeile 110 nun die Ausgabe »SYNTAX FEHLER IN ZEILE «. In Zeile 120 wird das Programm beendet. Mit TRAP... RESUME läßt sich sogar die STOP-Taste ausschalten.

Verlängern wir einmal das Programm:

```
105 IF ER = 30 THEN RESUME
```

Wird jetzt die STOP-Taste gedrückt, so verzweigt das Programm zur Zeile 100. Die Bedingung in Zeile 105 wird erfüllt, also erfolgt mit RESUME der Rücksprung an die Stelle des Programms, an der es unterbrochen wurde. So, nun haben Sie einen Dauerläufer, den Sie nicht mehr unterbrechen können; oder doch?

Drücken Sie die RUN/STOP-Taste, und halten Sie sie gedrückt. Nun müssen Sie noch gleichzeitig die RESET-Taste drücken. Lassen Sie jetzt die RESET-Taste los und dann erst die andere Taste. Der Bildschirm hat sich verändert. Sie befinden sich im Maschinensprachmonitor. Verlassen können Sie ihn mit X und dann RETURN. Das Programm ist immer noch vorhanden, geben Sie einmal LIST ein. Versuchen Sie selbständig auf die Fehlernummer 22, sie entspricht der Fehlermeldung »TYPE MISMATCH«, zu reagieren. Diese Fehlermeldung erscheint immer dann, wenn statt eines numerischen Wertes ein Zeichen oder statt eines Zeichens ein numerischer Wert eingegeben wurde.

Mit all diesen beschriebenen Praktiken und Befehlen werden Sie sicher jeden Fehler in einem Programm entdecken können. Sollte es nicht sofort klappen, verzweifeln Sie nicht. Gehen Sie lieber eine Tasse Kaffee trinken oder lesen Sie die Tageszeitung durch. Mit ein wenig Abstand geht es hinterher noch einmal so gut.

20 Maschinensprache mit dem Plus/4

Im Plus/4 ist ein Monitorprogramm enthalten, das wir hier beschreiben möchten. Zunächst aber müssen wir uns einmal ansehen, was Maschinensprache überhaupt ist. Im Kapitel über den Aufbau des Computers haben wir gesehen, daß der Computer nur mit Einsen und Nullen arbeiten kann. Andere Schaltzustände kann der Datenbus auch nicht annehmen. Ebenso verhält es sich mit den Befehlen, die der Mikroprozessor bekommt. Es handelt sich dabei nur um verschiedene Zustände auf dem Datenbus. Da der Datenbus 8 Bit breit ist, kann der Mikroprozessor theoretisch maximal 2^8 = 256 verschiedene Befehle unterscheiden, nämlich genau soviele, wie auf dem Datenbus Zahlen darstellbar sind. Tatsächlich hat der 6502-Mikroprozessor aber nur 151 verschiedene Befehle (Operationscodes). Es bleiben 105 Schaltzustände übrig, die den Mikroprozessor, wenn er auf einen solchen »undefinierten« Operationscode stößt, entweder »abstürzen« läßt oder ihn zu unkontrolliertem Verhalten bewegt. Es muß also darauf geachtet werden, daß der Mikroprozessor die Befehlscodes auch wirklich in seinem Befehlssatz enthält.

Bei den 151 Befehlen handelt es sich um Transportbefehle, arithmetische Befehle, logische Befehle, Vergleichsbefehle, Verzweigungsbefehle, Sprungbefehle, Schiebebefehle, Statusregisterbefehle und noch einige andere Befehle. Diese Befehle müssen dem Mikroprozessor also durch den Datenbus mitgeteilt werden. Dazu würde es genügen, am Datenbus acht Schalter anzuschließen und die jeweils gewünschten Schaltzustände einzustellen. Das wäre natürlich eine enorm umständliche Angelegenheit. Diese »Schaltarbeit« wird uns vom Festwertspeicher abgenommen. Durch die fest eingebauten Programme ist es nämlich möglich, dem Mikroprozessor die Befehle anhand von Zahlen zu übermitteln. Das Programm, das uns dies ermöglicht, nennt sich Monitorprogramm. Hierbei brauchen die acht Schaltzustände des Datenbusses nicht durch Einsen und Nullen eingegeben zu werden, sondern durch Hexadezimalzahlen.

Die Maschinensprache besteht also aus einer Reihe von Hexadezimalzahlen, die die Befehle und Daten darstellen. Die Maschinensprache besteht aus relativ einfachen und leicht verständlichen Befehlen, die aber als ganzes Programm sehr unübersichtlich sind. Es wird ja auch nicht, wie in BASIC, mit klar verständlichem Text, sondern mit abstrakten Zahlen gearbeitet. BASIC ist eine, den Anwender unterstützende Programmiersprache, die Maschinensprache dagegen unterstützt die Fähigkeiten des Mikroprozessors. Die Maschinensprache hat daher auch einen entscheidenden Vorteil gegenüber BASIC, sie ist nämlich schneller.

Sehen wir uns dazu ein kleines Beispielprogramm an:

Es soll von 255 bis 0 zurückgezählt werden, und die aktuelle Zahl soll in der Speicherstelle $0003 angezeigt werden. Dazu brauchen wir folgendes Maschinenprogramm:

A9 FF 85 03 C6 03 D0 FC 00

Als Ersatz für diese abstrakten Zahlen gibt es die sogenannte Assemblerschreibweise. Dabei werden die Befehle nicht als Zahl, wie sie der Mikroprozessor versteht, geschrieben, sondern als Klartext. Das ist etwa vergleichbar mit den BASIC-Programmen, bei denen die BASIC-Befehle im Computerspeicher auch nicht als Klartext, sondern als Befehlszeichen, sogenannte BASIC-Token, abgespeichert sind.

Unser Beispielprogramm sieht als Assemblertext so aus:

```
LDA #$FF
STA $03
DEC $03
BNE $FC
BRK
```

Das sieht doch schon etwas besser aus! Unser Programm ist nur neun Bytes lang und braucht zur Ausführung genau 1802 Taktzyklen, das sind etwa 1,8 Millisekunden. Der Mikroprozessor bekommt an seinen »Takteingang« Rechteckimpulse. Jeder Impuls ist ein Taktzyklus, der den Mikroprozessor einen Schritt weiterarbeiten läßt. Die Maschinensprachebefehle brauchen zur Ausführung zwei bis sieben Takte »Zeit«. Da wir in unserem Programm eine Programmschleife, ähnlich FOR...NEXT in BASIC, haben, kommen wir so auf 1802 Takte.

Unser Beispielprogramm würde in BASIC folgendermaßen aussehen:

```
10 FORX=255TO0STEP-1
20 POKE3,X
30 NEXTX
```

Dieser Programmtext verbraucht auch nicht allzuviel Speicherplatz. Aber bedenken Sie, daß dieses BASIC-Programm allein die gestellte Aufgabe noch nicht erfüllt. Dazu wird der BASIC-Interpreter benötigt, der einige Kbytes Speicher belegt. In diesem Interpreter müssen umfangreiche Routinen abgearbeitet werden, um unser kurzes Beispielprogramm zum Laufen zu bekommen. Das ist auch der Grund, daß das BASIC-Programm über eine Sekunde braucht, um in der Speicherstelle 3 Werte von 255 bis 0 darzustellen. Im Vergleich zum Maschinenprogramm ist das etwa die 500-fache Zeit!

Dieses einfache Beispiel zeigt Ihnen, daß die Maschinensprache zwar unübersichtlicher aussieht, dafür aber enorme Geschwindigkeitsvorteile gegenüber BASIC hat. Zur Eingabe der Maschinensprache brauchen wir den Maschinensprachemonitor, im Plus/4 TEDMON genannt.

20.1 Der TEDMON

Dieses Programm ermöglicht es, Maschinenspracheprogramme einzugeben, sie abzuspeichern oder Programme zu laden und sie zu starten. TEDMON starten Sie mit dem Befehl:

MONITOR

Sie können diesen Befehl auch abkürzen, indem Sie nur M und Shift/O eingeben. TEDMON bietet eine Reihe von Befehlen für die Maschinenspracheprogrammierung:

A (oder .)	=	Assemblieren
C	=	Vergleichen
D	=	Dissassemblieren
F	=	Füllen
G	=	Starten
H	=	Suchen
L	=	Laden
M	=	Speicherinhalt listen
R	=	Register anzeigen
S	=	Speichern
T	=	Verschieben
V	=	Programm vergleichen (Kassette/Diskette mit Computerspeicher)
X	=	Programm beenden

TEDMON-Befehl A

Mit diesen Befehl können Sie Maschinenspracheprogramme in der Assemblerschreibweise eingeben.

Schreibweise für A:

A Adresse Befehl (Operant)

Adresse	Speicheradresse (hexadezimal), an der der Befehl gespeichert werden soll.
Befehl	Assemblerschreibweise des Befehls.
Operant	Es gibt drei Arten von Maschinensprachebefehlen: 1-Byte-, 2-Byte- und 3-Byte-Befehle. Bei 1-Byte-Befehlen darf kein Operant eingegeben werden. Bei 2-Byte-Befehlen muß ein Operant eingegeben werden und bei 3-Byte-Befehlen muß eine 2-Byte-Adresse angegeben werden.

Beispiele:

A 3000 JMP $3000

Das Beispiel ergibt eine unendliche Programmschleife, da mit dem Befehl JMP (Jump - springe) hier zur Adresse $3000 gesprungen wird. JMP ist ein 3-Byte-Befehl.

A 3000 LDA #$FF

LDA ist ein 2-Byte-Befehl. Hier wird ein prozessorinternes Register (Accu) mit dem Wert $FF geladen.

A 3000 BRK

BRK ist ein 1-Byte-Befehl, der für einen »Software-Interrupt« sorgt, eine Programmunterbrechung. Der Befehl A kann auch durch einen Punkt (.) ersetzt werden. Nach der Eingabe eines Assemblerbefehls wird der Befehl in der hexadezimalen Schreibweise ausgegeben, und die nächste Adresse wird angegeben.

TEDMON-Befehl C

Mit C können zwei Speicherbereiche Byte für Byte verglichen werden.

Schreibweise für C:

C 1.Adresse 2.Adresse 3.Adresse

1.Adresse	Anfangsadresse des zu vergleichenden Speicherbereichs.
2.Adresse	Endadresse dieses Speicherbereichs.
3.Adresse	Anfangsadresse des Bereiches, der mit dem ersten Bereich verglichen werden soll.

TEDMON-Befehl D

Mit diesem Befehl können Sie genau das Gegenteil vom Befehl A bewirken. Es wird also von einem, schon gespeicherten Programm der Assemblertext erstellt. Man nennt diesen Vorgang »Disassemblieren«.

Schreibweise:

D Adresse (Endadresse)

Adresse	Anfangsadresse
Endadresse	Endadresse des Bereichs, der disassembliert werden soll.

Beispiel:

D 9000

Es wird ab Adresse $9000 disassembliert. Dabei werden maximal 21 Zeilen auf dem Bildschirm ausgegeben. Sie können die Disassemblierung fortsetzen, indem Sie nur D (ohne Adresse) eingeben. Wenn Sie dagegen die Endadresse mit eingeben, wird der gesamte Bereich disassembliert.

TEDMON-Befehl F

Mit diesen Befehl lassen sich auf einfachste Weise ganze Speicherbereiche mit einem bestimmten Wert füllen.

Schreibweise:

F 1.Adresse 2.Adresse Wert

1.Adresse	Anfangsadresse des zu füllenden Speicherbereichs.
2.Adresse	Endadresse des Bereichs.
Wert	Der hexadezimale Wert, mit dem der Bereich belegt werden soll.

Beispiel:

F 2000 3000 F1

Hierdurch werden alle Adressen zwischen $2000 bis einschließlich $3000 mit dem Wert $F1 belegt.

TEDMON-Befehl G

Mit diesem Befehl können Sie Ihre Maschinenspracheprogramme starten. Dazu geben Sie bitte ein:

G Adresse

Natürlich können Sie mit G auch Programme starten, die im ROM gespeichert sind.

TEDMON-Befehl H

Dies ist ein sehr nützlicher Befehl, den Sie dazu brauchen können, in einem Speicherbereich nach bestimmten Daten zu suchen.

Schreibweise für H:

H Anfangsadresse Endadresse Daten

Anfangs- und Endadresse	Der zu untersuchende Speicherbereich.
Daten	Hier können Sie die Daten einsetzen, die in dem angegebenen Speicherbereich gesucht werden können. Das können entweder einzelne Bytes sein, oder auch mehrere Bytes (bis zu 27), jeweils durch ein Leerzeichen getrennt.

Es kann aber auch ein String eingesetzt werden. Vor dem String, der maximal 32 Zeichen lang sein darf, muß ein Accent-Zeichen stehen.

Beispiel:

H 8000 FFFF 20 DC A0

Dieses Beispiel sucht im Speicherbereich $8000 bis $FFFF die Bytes $20 $DC $A0.

Der Computer wird die Anfangsadressen ausgeben, an denen das erste Byte steht:

A05C A066 A06C A072

H 8000 FFFF 'CBM

Jetzt sucht der Computer nach der Zeichenfolge CBM. Er findet diese in:

8007 FC56

TEDMON-Befehl L

Mit diesem Befehl werden Programme von Kassette oder Diskette geladen.

Schreibweise:

L ("(d:)Filename"(,g))

d	Laufwerknummer, entweder 0 oder 1. Wenn nur ein Disketten-Laufwerk angeschlossen ist, kann die Angabe der Nummer entfallen.
g	Gerätenummer. 1 für den Kassettenrecorder und 8 für das Disketten-Laufwerk. Wenn vom Kassettenrecorder geladen werden soll, kann die Angabe weggelassen werden.

Soll das nächste auf einer Kassette befindliche, Programm geladen werden, braucht nur der Befehl L eingegeben zu werden.

Beispiel:

L "GRAFIK1",8

Lädt das File »GRAFIK1« von der Diskette.

L "SPIEL"

Lädt das File »SPIEL« von der Kassette.

TEDMON-Befehl M

Mit diesem Befehl werden beliebige Speicherbereiche auf dem Bildschirm angezeigt. Dabei wird die Adresse des ersten Byte ausgegeben. Dann folgen acht Byte und dann die entsprechenden CHR$-Codes der acht Byte (in reverser Schrift).

Schreibweise:

M (Anfangsadresse) (Endadresse)

Anfangsadresse	Wird nur diese Adresse eingegeben, werden 96 Byte ausgegeben.
Endadresse	Werden beide Adressen eingegeben, so erfolgt die komplette Ausgabe von der Anfangs- bis zur Endadresse.

Die Ausgabe kann mit der COMMODORE-Taste verlangsamt werden, oder mit der RUN/STOP-Taste abgebrochen werden. Wenn anschließend nur der M-Befehl eingegeben wird, wird die Ausgabe fortgesetzt. Untersuchen Sie den Speicherbereich ab $8007. Wie wir bei unserem Beispiel für den H-Befehl gesehen haben, soll ab Speicherstelle $8007 die Zeichenfolge CBM stehen. Um das zu überprüfen, geben Sie bitte ein:

M 8007

Es werden nun 12 Zeilen mit je acht Byte ausgegeben. Und tatsächlich sind die ersten drei Byte die gesuchten Zeichen CBM. Wenn die Ausgabe fortgesetzt werden soll, brauchen Sie nur die Taste M und die RETURN-Taste zu drücken. Die Ausgabe von weiteren 12 Zeilen beginnt unmittelbar.

M 8000 8100

gibt den ganzen Bereich von $8000 bis $8100 aus.

Ändern von Speicherinhalten

Mit dem Größer-als-Zeichen »>« können Sie bis zu acht Speicherstellen ändern. Durch den M-Befehl wird das Größer-als-Zeichen schon automatisch ausgegeben. Hier brauchen Sie dann nur noch mit den Cursortasten an die gewünschte Stelle gehen und die Änderungen einzutippen. Nach Druck der RETURN-Taste wird der Speicherinhalt geändert.

Schreibweise:

> Adresse (ein bis acht Byte)

Der TEDMON-Befehl R

Mit diesem Befehl lassen sich die Prozessor-Registerinhalte anzeigen. Dazu brauchen Sie nur R eingeben. Ausgegeben werden der Programmzähler PC, das Statutsregister SR, der Accu AC, das X-Register XR, das Y-Register YR und der Stapelzeiger SP.

Ändern der Prozessorregister

Die mit R angezeigten Register des Mikroprozessors können auch geändert werden. Dazu dient das Semikolon (;). Das Semikolon wird mit R schon automatisch ausgegeben. Sie brauchen nur noch mit den Cursor-Tasten an die gewünschte Stelle gehen und Ihre Änderung einzugeben.

Der TEDMON-Befehl S

Mit diesem Befehl können Speicherinhalte auf Kassette oder Diskette abgespeichert werden.

Schreibweise:

S "Filename",g,Anfangsadresse,Endadresse+1

Beispiel:

S "ROUTINE",1,2000,21FF

Durch diesen Befehl wird der Speicherbereich von $2000 bis $21FE auf Kassette abgespeichert.

S "DATEN",8,1700,2400

speichert den Bereich von $1700 bis $23FF unter dem Namen »DATEN« auf Diskette. Die Speicherung erfolgt als Programmfile. Denken Sie immer daran, zur Endadresse eine 1 zu addieren, sonst fehlt das letzte Byte.

Der TEDMON-Befehl T

Mit diesem Befehl kann der Inhalt eines Speicherbereiches in einen anderen Speicherbereich kopiert werden.

Schreibweise:

T 1.Adresse 2.Adresse 3.Adresse

1.Adresse	Anfangsadresse des zu kopierenden Bereichs.
2.Adresse	Endadresse des zu kopierenden Bereichs.
3.Adresse	Anfangsadresse des Speicherbereichs, in den kopiert werden soll.

Beispiel:

T 3000 3100 1800

Hierdurch wird der Bereich von $3000 bis $3100 in den Bereich $1800 bis $1900 kopiert.

Der TEDMON-Befehl V

V ist die Abkürzung für VERIFY. Diesen Befehl kennen Sie ja schon. Mit V wird der Inhalt eines Files auf Kassette oder Diskette mit dem Inhalt des entsprechenden Speicherbereichs im Plus/4 verglichen. Wird dabei ein Unterschied festgestellt, wird die Meldung ERROR ausgegeben.

Schreibweise:

V "Filenamen",g

Filenamen	Name des zu prüfenden Programms
g	Gerätenummer, 1 für Kassette und 8 für Diskette.

Rückkehr aus TEDMON

Mit dem Befehl X können Sie wieder in das BASIC Ihres Plus/4 zurückkehren. Ein eventuell vorhandenes BASIC-Programm wurde nicht verändert, sofern Sie nicht selbst den BASIC-Speicherinhalt geändert haben.

20.2 Der BASIC-Befehl SYS

Mit SYS können Sie Maschinenspracheprogramme von BASIC aus starten. Das funktioniert sowohl im Direktmodus, wie auch im Programmodus. Wir haben dadurch also die Möglichkeit Maschinenspracheroutinen auch in BASIC zu nutzen.

Schreibweise für SYS:

SYS Adresse

Als Adresse ist die Startadresse des Maschinenprogramms in dezimaler Schreibweise anzugeben. Die Angabe der Adresse muß natürlich genau stimmen, da der Rechner sich sonst eventuell aufhängen könnte. Mit SYS können natürlich auch Programme des BASIC- oder Betriebssystem-ROM gestartet werden. Nicht starten lassen sich da

gegen Programme, die sich im Bereich von $8000 bis $FFFF im RAM befinden. Mehr dazu im Kapitel 21 (Bankswitching).

Geben Sie einmal SYS 52651 ein. Auf dem Bildschirm erscheinen vier Namen. Dabei handelt es sich um die vier Autoren, die das BASIC entwickelt haben. Wir können dem Maschinenprogramm keine Werte übergeben. Es ist also nicht möglich, eventuell Variablen zu übergeben. Doch es gibt noch eine zweite Möglichkeit, Maschinenprogramme von BASIC aus zu nutzen: Der Befehl USR.

20.3 Der BASIC-Befehl USR

Mit diesem Befehl ist es ebenfalls möglich Maschinenspracheprogramme zu starten. Diesen Programmen können auch Werte übergeben werden, damit weitere Berechnungen durchgeführt werden können.

Schreibweise:

USR (Variable)

Sie sehen, hier wird keine Startadresse eingegeben. Die Adresse ist aber unbedingt nötig. Deshalb muß sie schon vor diesem Befehl festgelegt werden. Die Startadresse wird hierbei in den Speicheradressen 1281 und 1282 abgespeichert. Das geschieht von BASIC aus mit dem Befehl POKE. Da eine Adresse 16 Bit hat, in die Speicherstellen aber nur je acht Bit gespeichert werden können, muß die Adresse aufgeteilt werden.

Das geschieht folgendermaßen: Die Startadresse wird durch 256 dividiert und das Ergebnis wird abgerundet. In BASIC kann die Berechnung mit der Formel:

H = INT(Adresse/256)

durchgeführt werden. Das Ergebnis ist das sogenannte höherwertige Byte. Dieser Wert muß in die Speicherzelle 1282 abgespeichert werden.

Nun brauchen wir noch das niederwertige Byte. Das errechnen wir folgendermaßen:

L = Adresse - 256*H

Dieser Wert wird in Speichezelle 1281 abgespeichert. Nun kann das Maschinenprogramm mit dem Befehl USR gestartet werden. Die Variable, die hierbei übergeben wird, wird im sogenannten Fließkomma-Akku aufgenommen und kann von dort aus bearbeitet werden. Wenn das Maschinenprogramm mit einem RTS-Befehl wieder verlassen wird, wird die Variable mit dem Wert geladen, der im Fließkomma-Akku steht.

Der Befehl USR wird selten verwendet, man benutzt lieber den Befehl SYS. Auch mit SYS können Werte vom BASIC in das Maschinenprogramm übernommen werden. Allerdings ist das nur mit kleinen Tricks möglich. Der Vorteil liegt aber darin, daß auch mehrere Werte übernommen werden können. Wir möchten hier aber nicht zu weit in diese Materie einsteigen. Wenn Sie Interesse an diesen Themen haben, lesen Sie bitte die nächsten Kapitel oder spezielle Literatur über die Maschinensprache des 6502.

20.4 Einführung in die Maschinensprache

Bisher haben Sie in unserem Buch nur die Programmierung in BASIC kennengelernt. Somit besitzen Sie nun Kenntnisse über eine höhere Programmiersprache. In diesem Kapitel haben wir Ihnen den Maschinensprachemonitor vorgestellt. Doch was nützt Ihnen das Monitorprogramm, wenn Sie keine Kenntnisse über die Maschinensprache haben. Nun, das wird sich jetzt bald ändern, denn wir werden Ihnen eine kurze Einführung in die Maschinensprache geben.

Bei der Maschinensprache handelt es sich um die Befehle, die der Microprozessor Ihres Plus/4 direkt verstehen kann. Also nicht »PRINT« usw., sondern »Load Accu« oder »Jump« und dergleichen. Dazu möchte ich Sie noch einmal an das Kapitel 2 (Grundsätzlicher Aufbau eines Computers) erinnern: Dort haben wir festgestellt, daß der Computer BASIC-Befehle nur über ein Übersetzungsprogramm (BASIC-Interpreter) verstehen kann. Dabei werden die BASIC-Befehle in eine Reihe von Maschinensprachebefehle umgewandelt.

Mit dem Monitorprogramm können Sie die Maschinensprachebefehle direkt eingeben. Zur Einführung in die Maschinensprache muß angemerkt werden, daß mit diesem Thema leicht ein ganzes Buch gefüllt werden kann, wir werden jetzt also das ganze Thema im komprimierter Form bringen. Das bedeutet aber keineswegs, daß unsere Einführung unvollständig ist. Wir werden Ihnen zunächst den Aufbau des 7501-Mikroprozessors zeigen, beschreiben anschließend die Befehls- und Adressierungsarten und bringen zum Schluß des Kapitels einige Beispielprogramme. Doch zunächst sehen wir uns einmal den Aufbau und die internen Register des 7501-Mikroprozessors an.

20.5 Aufbau des 7501

Im Plus/4 wird ein 7501-Microprozessor verwendet, ein softwarekompatibler Typ zum bekannten 6502. Der Prozessor ist in einem 40poligen Gehäuse untergebracht.

Die Besonderheit des 7501 besteht darin, daß dieser Prozessor über sieben programmierbare Ein- und Ausgabeleitungen verfügt. Diese werden im Plus/4 für die serielle Schnittstelle und für die Kassettenrecorder-Schnittstelle verwendet. Die sieben I/O-Leitungen werden mit P0, P1, P2, P3, P4, P5 und P7 bezeichnet. Die Leitung P2 ist über einen Inverter auch am User-Port angeschlossen und trägt dort die Bezeichnung ATN.

Die Leitungen P0 bis P7 können über die Adressen $0000 und $0001 programmiert werden. Die Adresse $0000 dient dabei als Datenrichtungsregister und $0001 als Datenregister. Eine '1' im Datenrichtungsregister schaltet die entsprechende I/O-Leitung als Ausgang. Dort erscheint dann, je nach Zustand des entsprechenden Bits im Datenregister, entweder Low oder High. Wenn im Datenrichtungsregister ein Bit den Wert '0' hat, wird die entsprechende I/O-Leitung als Eingang geschaltet. Im Register $0001 kann dann der Zustand dieser Leitung abgefragt werden.

Kommen wir zur Beschreibung der CPU-Register. Sie enthält sechs interne Register, die wir uns nun anschauen werden.

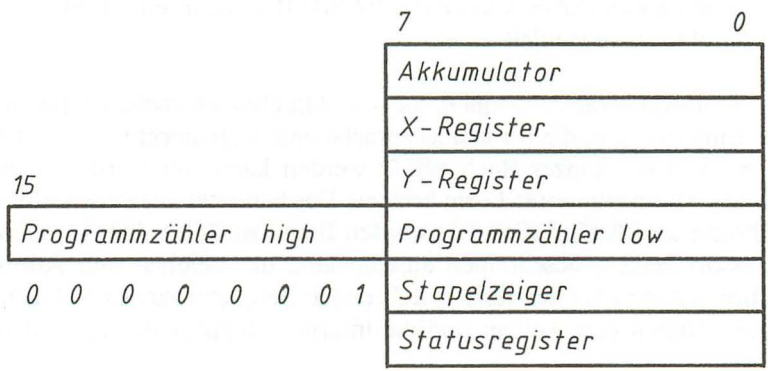


Bild 20.1: *Register des 7501*

Der Akkumulator ist ein 8-Bit-Register. Dieser Akkumulator ist das zentrale Rechenregister der CPU, also das wichtigste Register. Wir werden in diesem Kapitel noch des öfteren auf den Akkumulator und seine Programmierung zurückkommen. Im Akku erscheinen z.B die Ergebnisse logischer Operationen, Additionen und Subtraktionen.

Die X- und Y-Register sind ebenfalls 8-Bit-Register. Mit diesen Registern lassen sich z. B. indizierte Adressierungen durchführen. Wir kommen später noch darauf zurück.

Der Programmzähler besteht aus zwei 8-Bit-Registern. Hier wird stets die Adresse angezeigt, an der die CPU gerade eine Operation ausführt. Genaugenommen zeigt der Programmzähler bei der Abarbeitung eines Befehls schon auf den nächsten Befehl, da die CPU die Befehls- und Datenbytes zunächst einliest um den Befehl dann zu interpretieren und auszuführen. Der Programmzähler zeigt dann schon auf das nächste Befehlsbyte. Sie brauchen sich um den Programmzähler, außer bei den Verzweigungsbefehlen, keine Gedanken zu machen.

Das nächste Register, wieder ein 8-Bit-Register, ist der sogenannte Stapelzeiger, auch Stackpointer genannt. Mit diesem Register hat es eine besondere Bewandnis: Die CPU ist hierdurch in der Lage, Daten in einen besonderen Speicherbereich zu speichern und wieder herauszulesen. Dieser besondere Speicherbereich, Stapelspeicher genannt, liegt bei $0100 bis $01FF. Mit dem 8-Bit-Register lassen sich dabei die Adressen von $00 bis $FF darstellen. Die übrigen acht Bits (%00000001) werden dann automatisch von der CPU dazugegeben (siehe Bild 20.1). Sie wissen ja, eine Adresse wird mit 16 Bit dargestellt.

Der Stapelzeiger zählt vom Wert $FF auf 0 zurück, d.h., beim Einschalten des Computers hat er den Wert $FF. Die ersten Daten, die im Stapelspeicher abgespeichert werden sollen, werden somit an Adresse $01FF abgespeichert. Beim Lesen der Daten aus dem Stapelspeicher werden die zuletzt hineingeschriebenen Daten zuerst herausgelesen. Stellen Sie sich einen Stapel Bretter vor. Das zuletzt auf dem Stapel abgelegte Brett muß auch zuerst wieder heruntergenommen werden. Natürlich können Sie auch versuchen, das unterste Brett herauszuziehen, das ist aber schon mit ein wenig Aufwand verbunden. Bei unserem Stapelspeicher ist es ebenfalls schwierig, »ältere« Daten zu lesen. Man müßte dazu den Stapelzeiger mit einem speziellen Befehl ändern. Dabei ist aber Vorsicht geboten, denn allzu leicht passiert es dabei, daß der Stapelspeicher »überläuft«, und Sie entweder Fehlermeldungen bekommen, oder die Programme sich aufhängen.

Kommen wir nun zum nächsten Register, dem Statusregister. In diesem Register wird der aktuelle Zustand der CPU angezeigt. Jedes einzelne Bit des Statusregisters hat dabei eine besondere Bedeutung, die wir jetzt erläutern möchten. Wir bezeichnen die Bits im Statusregister auch als »Flags« (Flaggen). Die Flags können durch spezielle Befehle gelöscht (0) oder gesetzt (1) werden. Ansonsten werden die Flags durch fast alle Maschinensprachbefehle beeinflußt.

Bit 0 des Statusregisters ist das Carry-Flag. Dieses Flag wird gesetzt (es wird 1), wenn bei der Ausführung eines Befehls ein Übertrag aus Bit 7 des entsprechenden Registers erfolgt, bzw. wenn bei einem Vergleichsbefehl die angesprochene Speicherzelle gleich oder größer als das Prozessorregister ist.

Bit 1 (Zero-Flag) wird 1, wenn das Ergebnis einer Rechenoperation Null ist, bzw. wenn die Operanden eines Vergleichsbefehls gleich sind.

Bit 2 (Interrupt-Flag) zeigt an, ob ein IRQ (Interruptrequest) zugelassen wird oder nicht. Ist das Bit 2 = 0, ist der IRQ, also eine Programmunterbrechung zugelassen.

Bit 3 (Dezimal-Flag) zeigt an, ob Addition und Subtraktion dezimal oder hexadezimal ausgeführt wird. Bit 3 = 0 = Dezimal.

Bit 4 (Break-Flag) zeigt an, ob eine Programmunterbrechung mit einem BRK-Befehl erfolgte.

Bit 5 hat immer den Wert 1.

Bit 6 (Überlauf- oder Overflow-Flag) wird 1, wenn bei einer Addition oder bei einer Subtraktion im Akku ein Übertrag von Bit 6 nach Bit 7 erfolgt. Außerdem beeinflußt der Bit-Befehl dieses Flag.

Bit 7 (Negativ-Flag) wird 1, wenn ein Ergebnis negativ ist.

20.6 Die Zero-Page

Bevor wir uns die Befehle und Adressierungsarten ansehen, noch ein paar Worte zur Zero-Page (Seite 0).

Mit Zero-Page bezeichnet man den Speicherbereich von $0000 bis $00FF. Dieser Bereich befindet sich unmittelbar vor dem Stapelspeicher. Die Speicherzellen der Zero-Page können mit speziellen Befehlen direkt angesprochen werden. Der Vorteil liegt darin, daß dabei nur das Low-Byte, 8 Bit, angegeben werden muß (ähnlich wie beim Stapelspeicher). Die restlichen 8 Bit (%00000000) werden von der CPU automatisch angegeben. Durch diese Möglichkeit spart man Speicherplatz in den Maschinenprogrammen. Außerdem wird durch die Zero-Page-Befehle Rechenzeit gespart. Die Zero-Page wird bei den 6502-Rechnern, auch beim Plus/4, als Zwischenspeicher genutzt. Darüberhinaus gibt es einige Befehle, die mit speziellen Adressierungsarten nur mit der Zero-Page zusammenarbeiten.

20.7 Befehlsarten des 7501

Den umfangreichen Befehlssatz des 7501-Mikroprozessors kann man in Gruppen unterteilen: Transportbefehle, Statusregisterbefehle, arithmetische Befehle, logische Befehle, Verzweigungsbefehle, Sprungbefehle, Vergleichsbefehle, Schiebebefehle und sonstige Befehle. Die nun folgende Schreibweise der Befehle ist die sogenannte Assemblerschreibweise. Dabei werden die Befehle nicht als Hexadezimalcode, sondern anhand von jeweils drei Buchstaben angegeben. Diese sind die Abkürzung für die eigentlichen Befehle. Bei der folgenden Auflistung geben wir Ihnen die Bedeutung der Befehle und die beeinflußten Flags des Statusregisters in den Klammern an.

In der Anlage M finden Sie eine komplette Befehlsauflistung mit den entsprechenden Befehlscodes der verschiedenen Adressierungsarten und ebenfalls mit der Angabe der beeinflußten Flags.

Transportbefehle

LDA (Lade den Akkumulator, N, Z)
STA (Transportiere Akkumulatorinhalt)
LDX (Lade X-Register, N, Z)
STX (Transportiere X-Registerinhalt)
LDY (Lade Y-Register, N, Z)
STY (Transportiere Y-Registerinhalt)
TAX (Lade X-Register mit Akkumulatorinhalt, N, Z)
TAY (Lade Y-Register mit Akkumulatorinhalt, N, Z)
TXA (Lade Akku mit X-Registerinhalt, N, Z)
TYA (Lade Akku mit Y-Registerinhalt, N, Z)
TXS (Speichere X-Register im Stapelzeiger)
TSX (Lade X-Register mit Stapelzeiger, N, Z)
PLA (Lade Akku mit Stapelspeicherinhalt, N, Z)
PHA (Speichere Akku in Stapelspeicher)
PLP (Lade Statusregister mit Stapelspeicherinhalt)
PHP (Speichere Statusregister in Stapelspeicher)

Statusregisterbefehle

CLC (Lösche Carry-Flag, C=0)
SEC (Setze Carry-Flag, C=1)
CLD (Lösche Dezimal-Flag, D=0)
SED (Setze Dezimal-Flag, D=1)
CLI (Lösche Interrupt-Flag, I=0)
SEI (Setze Interrupt-Flag, I=1)
CLV (Lösche Überlauf-Flag, V=0)

Arithmetische Befehle

ADC (Addiere zum Akku plus Carry-Flag, N, Z, C, V)
SBC (Subtrahiere vom Akku, Ergebnis minus Carry-Flag, N, Z, C, V)
INC (Addiere 1 zum Speicherinhalt, N, Z)
DEC (Subtrahiere 1 vom Speicherinhalt, N, Z)
INX (Addiere 1 zum X-Registerinhalt, N, Z)
DEX (Subtrahiere 1 vom X-Registerinhalt, N, Z)
INY (Addiere 1 zum Y-Registerinhalt, N, Z)
DEY (Subtrahiere 1 vom Y-Registerinhalt, N, Z)

Logische Befehle

AND (UND-Funktion mit Akkuinhalt, N, Z)
ORA (ODER-Funktion mit Akkuinhalt, N, Z)
EOR (EXCLUSIV-ODER-Verknüpfung mit Akkuinhalt, N, Z)

Verzweigungsbefehle

BCC (Verzweige, wenn C-Flag = 0)
BCS (Verzweige, wenn C-Flag = 1)
BNE (Verzweige, wenn Z-Flag = 0)
BEQ (Verzweige, wenn Z-Flag = 1)
BPL (Verzweige, wenn N-Flag = 0)
BMI (Verzweige, wenn N-Flag = 1)
BVC (Verzweige, wenn V-Flag = 0)
BVS (Verzweige, wenn V-Flag = 1)

Sprungbefehle

JMP (Springe an andere Speicherstelle)
JSR (Springe in Unterprogramm)
RTS (Kehre aus Unterprogramm zurück)
RTI (Kehre aus Interruptprogramm zurück)

Vergleichsbefehle

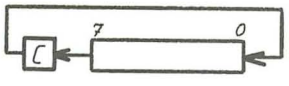
CMP (Vergleiche mit Akku, N, Z, C)
CPX (Vergleiche mit X-Register, N, Z, C)
CPY (Vergleiche mit Y-Register, N, Z, C)
BIT (UND-Verknüpfung mit Akku, ohne Akku zu ändern, Z, N-Flag bekommt den Wert des Bit 7 des Ergebnis, V-Flag bekommt den Wert des Bit 6 des Ergebnisses)

Bei diesen Befehlen werden nur die Flags im Statusregister geändert.

Schiebebefehle

ASL (Schiebe Akku- oder Speicherinhalt eine Stelle nach links, Bit 0 wird 0, Bit 7 kommt in das Carry-Flag, N, Z, C)

LSR (Schiebe Akku- oder Speicherinhalt eine Stelle nach rechts, Bit 7 wird 0, Bit 0 kommt in das Carry-Flag, N, Z, C)



ROL (Schiebe Akku- oder Speicherinhalt eine Stelle nach links, Bit 0 bekommt den Inhalt des Carry-Flags, Bit 7 kommt in das Carry-Flag, N, Z, C)

ROR (Schiebe Akku- oder Speicherinhalt eine Stelle nach rechts, Bit 7 bekommt Inhalt des Carry-Flags, Bit 0 kommt in das Carry-Flag, N, Z, C)

Sonstige Befehle

NOP (No Operation, Leerbefehl, bewirkt nichts)
BRK (Sogenannter Software-Interrupt, setzt I-Flag und bewirkt einen IRQ. BRK beachtet nicht ein eventuell gesetztes I-Flag.)

20.8 Adressierungsarten

Je nach Adressierungsart besteht ein Befehl aus einem, zwei oder drei Byte. Ein-Byte-Befehle bestehen natürlich nur aus dem Befehlsbyte selbst, zwei-Byte-Befehle bestehen aus dem Befehlsbyte und einem Adreßbyte (Zero-Page) oder einem Operand. Drei-Byte-Befehle schließlich bestehen aus dem Befehlsbyte und zwei Adreßbytes (niederwertiges Adreßbyte, höherwertiges Adreßbyte - Low-Byte, High-Byte).

Nun kann man nicht alle Befehle in allen Adressierungsarten verwenden. Eine Übersicht über die möglichen Befehle und deren Adressierungsarten gibt die Tabelle in Anhang M.

Wir zeigen Ihnen jetzt die verschiedenen Adressierungsarten. Dabei wenden wir, wie schon beim Kapitel 20.7 (Befehlsarten), die Assemblerschreibweise an. Die Befehle werden also durch drei Buchstaben bezeichnet. Die Beispiele in diesem Kapitel können Sie sich übrigens mit Ihrem Plus/4 ansehen. Sie müssen dazu das Monitorprogramm starten und den Befehl »D«, gefolgt von der im Beispiel angegebenen Adresse eingeben. Auf dem Bildschirm erscheinen dann einige Zeilen Assemblertext. Die erste Zeile ist die hier beschriebene Befehlszeile. Die Assemblerschreibweise richtet sich nach der Adressierungsart.

Die unmittelbare Adressierung

Sie belegt zwei Byte. Das zweite Byte ist immer ein Datenbyte.

Beispiel:

. A02D A9 80 LDA #$80

Der Akkumulator wird mit dem Wert $80 geladen.

Mögliche Befehle: ADC AND CMP CPX CPY EOR LDA LDX LDY ORA SBC

Die absolute Adressierung

Diese Adressierungsart belegt drei Bytes, wobei dem Befehlsbyte eine zwei-Byte-Adresse folgt.

Beispiel:

. A035 20 66 A0 JSR $A066

Das Programm verzweigt in eine Unterroutine, die bei der Adresse $A066 beginnt.

Mögliche Befehle: ADC AND ASL BIT CMP CPX CPY DEC EOR INC JMP JSR LDA LDX LDY LSR ORA ROL ROR SBC STA STX STY

Die Zero-Page-Adressierung

Sie belegt zwei Bytes. Das zweite Byte gibt die Adresse der Zero-Page an.

Beispiel:

. A02F 85 61 STA $61

Der Inhalt des Akkus wird in Speicherzelle $0061 abgelegt.

Mögliche Befehle: ADC AND ASL BIT CMP CPX CPY DEC EOR INC LDA LDX LDY LSR ORA ROL ROR SBC STA STX STY

Akkumulatorbezogene Adressierung

Diese belegt nur ein Byte, also das Befehlsbyte selbst. Diese Befehle beeinflussen nur den Akkumulator.

Beispiel:

. A0AE 4A LSR

Durch diesen Befehl wird der Inhalt des Akkus um eine Stelle nach rechts geschoben. Dabei bekommt das Bit 7 den Wert 0 und das Carry-Flag erhält den Inhalt des Bit 0.

Mögliche Befehle: ASL LSR ROL ROR

Implizierte Adressierung

Sie belegt ebenfalls nur ein Byte. Dazu gehören zum Beispiel Transportbefehle innerhalb der CPU und die Stackbefehle.

Beispiel:

. A0B1 A8 TAY

Dieser Befehl kopiert den Inhalt des Akkus in das Y-Register.

Mögliche Befehle: BRK CLC CLD CLI CLV DEX DEY INX INY NOP PHA PHP PLA PLP RTI RTS SEC SED SEI TAY TSX TXA TXS TYA

Indiziert indirekte Adressierung (Vorindiziert)

Nun wird's interessant. Diese Adressierungsart belegt zwei Byte, wobei das zweite Byte eine Adresse in der Zero-Page ist. Nun passiert folgendes: Der Inhalt des X-Registers wird mit der angegebenen Zero-Page-Adresse addiert. Das ergibt eine Adresse, deren Inhalt das Low-Byte der Zieladresse ist, also das niedrige Adreßbyte. Das High-Byte befindet sich eine Adresse weiter. Diese Zieladresse wird nun angesprochen. Für diese etwas komplizierte Adressierungsart gibt es im BASIC- und Betriebssystem des Plus/4 kein Beispiel. Denken wir uns also selbst eines aus:

81 A0 STA ($A0,X)

Der Akkuinhalt wird in der Speicherzelle gespeichert, deren Adresse in der Speicherzelle steht, deren Adresse sich aus der Summe der Adresse $00A0 und dem X-Register ergibt, alles klar?

Mögliche Befehle: ADC AND CMP EOR LDA ORA SBC STA

Indirekt indizierte Adressierung (Nachindiziert)

Dies sind ebenfalls 2-Byte-Befehle. Dabei wird zum Inhalt der angegebenen Zero-Page-Adresse das Y-Register addiert. Das Ergebnis ist die Zieladresse (Low-Byte). Das High-Byte muß in der im Befehl angegebenen Adresse + 1 stehen. Wenn sich bei der Addition ein Übertrag ergibt, wird zum High-Byte eine 1 addiert.

Beispiel:

. A0E2 B1 22 LDA ($22),Y

Hier wird zunächst der Inhalt der Adresse $0022 mit dem Inhalt des Y-Registers addiert. Das ergibt das Low-Byte der Zieladresse. Der Inhalt der Adresse $0023 plus ein eventuell vorhandener Übertrag ergibt das High-Byte der Zieladresse.

Mögliche Befehle: ADC AND CMP EOR LDA ORA SBC STA

Indizierte Adressierung Zero-Page-X-Register

Ebenfalls eine 2-Byte-Adressierung. Dabei wird zur angegebenen Zero-Page-Adresse der Inhalt des X-Registers addiert. Das High-Byte der Zieladresse ist immer 0.

Beispiel:

.A1C9 95 29 STA $29,X

Der Akku wird in der Speicherzelle abgespeichert, deren Adresse sich aus $0029 und dem X-Register ergibt.

Mögliche Befehle: ADC AND ASL CMP DEC EOR INC LDA LDY LSR ORA ROL ROR SBC STA STY

Absolut-X und Absolut-Y indizierte Adressierung

Hierbei handelt es sich wieder um 3-Byte-Befehle. Hier wird zur angegebenen Adresse das X- oder Y-Register addiert. Das Ergebnis ist die Zieladresse.

Beispiele:

. 9033 9D 00 02 STA $0200,X

Der Akku wird in der Speicherzelle abgespeichert, deren Adresse sich aus $0200 und dem X-Register ergibt.

. 937F D9 53 84 CMP $8453,Y

Der Inhalt der Adresse, die sich aus $8453 und dem Y-Register ergibt, wird mit dem Akku verglichen (beeinflußt N-, Z- und C-Flag).

Mögliche Befehle für Absolut-X: ADC AND ASL CMP DEC EOR INC LDA LDY LSR ORA ROL ROR SBC STA

Mögliche Befehle für Absolut-Y: ADC AND CMP EOR LDA LDX ORA ABC STA

Relative Adressierung

Kommen wir zu den Verzweigungsbefehlen. Dabei wird die relative Adressierung angewandt. Es handelt sich hierbei um 2-Byte-Befehle, mit denen der Programmzähler beeinflußt wird. Grundsätzlich wird bei den Verzweigungsbefehlen aufgrund bestimmter Zustände des Statusregisters verzweigt. Es wird, wenn die Bedingung (Sie können das mit IF...THEN... in BASIC vergleichen) erfüllt ist, die im Befehl angegebene Adresse zum Programmzähler addiert. Bei dieser Adresse handelt es sich um ein sogenanntes 2er-Komplement. D.h., die relative Adresse von 0 bis $7F ergibt Vorwärtsverzweigungen von 0 bis 127 Schritten. Ab $80 bis $FF ergeben sich Rückwärtsverzweigungen, wobei $FF eine Verzweigung um 1 Schritt zurück, und $80 eine Verzweigung um 128 Schritte zurück ergibt.

Das Ganze hat einen einfachen Grund: Um überhaupt Rückwärtsverzweigungen zu ermöglichen, muß eine negative relative Adresse angegeben werden. Unser relatives Adreßbyte besteht also aus 7 Adreßbits und einem Vorzeichenbit. Ein solches vorzeichenbehaftetes Byte wird 2er-Komplement genannt. Wir werden jetzt einmal eine solche Zahl bilden.

Wir wollen das 2er-Komplement der Zahl 100 bilden. Dazu muß zunächst jedes Bit dieser Zahl invertiert und zum Ergebnis eine 1 addiert werden.

```
Unsere Zahl     = 100 $64 %01100100
Negiert         = 155 $9B %10011011
+ 1             =   1 $01 %00000001
Ergibt          = 156 $9C %10011100
```

Bei einem Verzweigungsbefehl, bei dem eine Rückwärtsverzweigung um 100 Schritte erfolgen soll, müssen wir also die relative Adresse $9C angeben.

Beispiele:

. 9004 D0 03 BNE $9009

Wenn das Zero-Flag 0 ist, springt das Programm um drei Schritte nach vorn. Bei der Bearbeitung dieses Befehls steht der Programmzähler schon bei $9006!

. 9021 30 D7 BMI $8FFA

Das Programm springt um 41 Schritte zurück, wenn das Negativ-Flag 1 ist. Beachten Sie bitte, der Programmzähler steht bei $9023!

Mögliche Befehle: BCC BCS BEQ BMI BNE BPL BVC BVS

Indirekte Adressierung

Hier gibt es nur einen Befehl, einen Sprungbefehl. Dabei wird zur Adresse gesprungen, deren Low-Byte in der angegebenen Adresse steht. Das High-Byte steht in der angegebenen Adresse + 1.

Beispiel:

.8683 6C 00 03 JMP ($0300)

In der Adresse 0300 steht das Low-Byte und in der Adresse $0301 das High-Byte der Zieladresse.

Möglicher Befehl: JMP

Zero-Page-Y-Register indizierte Adressierung

Dabei handelt es sich um 2-Byte-Befehle. Zum angegebenen Adreßbyte wird das Y-Register addiert. Das Ergebnis ist das Low-Byte der Zieladresse. Das High-Byte ist immer 0.

Beispiel:

B6 A0 LDX $A0,Y

Das X-Register wird mit dem Inhalt der Speicherzelle geladen, deren Adresse sich aus der Adresse $A0 und dem Y-Register ergibt. Diese Speicherzelle liegt immer in der Zero-Page, da das High-Byte immer 0 ist.

Mögliche Befehle: LDX STX

20.9 Programmieren in Maschinensprache

Sie haben nun die Befehle des 7501-(6502-)Mikroprozessors kennengelernt und müßten eigentlich in der Lage sein, einfache Programme selbst zu schreiben.

Achtung: Bei der Eingabe der Maschinenprogramme sollte man sehr vorsichtig vorgehen. Allzuleicht passiert es, daß sich der Computer »aufhängt«, d.h., daß die CPU unkontrollierte Dinge verrichtet. So können z. B. sehr schnell wichtige Speicherzellen im Systemspeicher falsche Werte bekommen. Dann hilft oft nur noch das Drücken der RESET-Taste. In den meisten Fällen gibt es aber noch einen Trick: Drücken Sie die RUN/STOP-Taste, und halten Sie sie gedrückt. Dann drücken Sie die RESET-Taste und lassen sie wieder los. Wenn dann die gewohnte Monitoreinschaltmeldung erscheint, lassen Sie die RUN/STOP-Taste wieder los. Sie befinden sich jetzt wieder im Monitorprogramm. Leider gibt es aber auch Situationen, in denen das vorher Gesagte nicht funktioniert.

Am besten ist es, wenn Sie jedes Programm abspeichern, bevor Sie es starten. Bei der Maschinensprache gibt es nun einmal keine Fehlermeldungen wie bei der BASIC-Programmierung.

Beginnen wir mit einem einfachen Programmbeispiel: Es sollen zwei Werte addiert werden. Der erste Wert steht im Akku, der zweite im X-Register. Unsere Werte sollen die Zahlen $A0 und $6E sein. Das Ergebnis soll in der Speicherzelle $00D0 abgespeichert werden.

```
. 1000 A9 A0 LDA #$A0 ;Akku mit Wert $A0 laden
. 1002 18 CLC         ;Carry-Flag löschen
. 1003 69 6E ADC $6E  ;Akku plus $6E plus Carry-Flag addieren
. 1005 85 D0 STA $D0  ;Ergebnis in $D0
. 1007 00 BRK         ;Zurück zum Monitorprogramm
```

Sie sehen hier eine Besonderheit bei der Assemblerschreibweise, die Kommentare. Diese Anmerkungen erleichtern das Verständnis des Programms. Sie können diese Anmerkungen allerdings nicht mit Ihrem Monitorprogramm eintippen, sie sollen Ihnen in unserem Buch nur eine kleine Hilfe sein. Solche Anmerkungen sind in professionellen Assemblerprogrammen möglich. Diese bieten übrigens noch eine Menge mehr, z. B. Variablen oder Symboladressen statt irgendwelcher Hexadezimaladressen.

Da unser Plus/4 aber nur über einen sogenannten Direktassembler verfügt, müssen Sie das Programm (und alle folgenden Beispielprogramme) folgendermaßen eingeben:

```
. 1000 LDA #$A0 <RETURN-Taste>
CLC <RETURN-Taste>
ADC #$6E <RETURN-Taste>
STA $D0 <RETURN-Taste>
BRK <RETURN-Taste>
<RETURN-Taste>
```

Wenn Ihnen das zu lang ist, können Sie das Programm auch folgendermaßen eingeben (maximal 8 Bytes pro Zeile):

```
>1000 A9 A0 18 69 6E 85 D0 00 <RETURN-Taste>
```

Und schon ist das Programm im Speicher. Starten läßt sich dann mit:

```
G 1000 <RETURN-Taste>
```

Soviel zur Eingabe der Programme. Wir werden im folgenden hauptsächlich die Assemblerschreibweise für unsere Programme anwenden. Auf solche Hinweise, wie <RETURN-Taste>, verzichten wir jetzt, denn das sollte allmählich klar sein. Kommen wir nun zu unserem Programm zurück.

Wenn wir das Programm gestartet haben, wird in der Speicherzelle $D0 das Ergebnis stehen, nämlich $0E. Das Ergebnis steht übrigens auch noch im Akku, den wir ja bei der Einschaltmeldung des Monitorprogramms sehen: In der Registeranzeige steht unter AC der Wert $0E. Aber das Ergebnis stimmt doch nicht, werden Sie sagen, $A0 und $6E ergibt doch schließlich $10E. Stimmt, da der Akku aber nur ein 8-Bit-Register ist, stehen dort auch nur die unteren acht Bit des Ergebnisses. Wenn wir das ganze Ergebnis haben wollen, müssen wir das Programm ein wenig erweitern. Das Ergebnis soll in $D0 und $D1 stehen.

```
. 1000 D8      CLD
. 1001 A9 00   LDA #$00
. 1003 85 D1   STA $D1
. 1005 18      CLC
. 1006 A9 A0   LDA #$A0
. 1008 69 6E   ADC #$6E
. 100A 90 02   BCC $100E
. 100C E6 D1   INC $D1
. 100E 85 D0   STA $D0
. 1010 00      BRK
```

Wir haben hier zunächst das Dezimal-Flag gelöscht. Das ist unbedingt notwendig, da wir hier dual rechnen möchten. Nur so wird unser Programm richtig funktionieren. Jetzt wird nämlich bei dem ADC-Befehl, je nach Ergebnis, das Carry-Flag gesetzt. Erinnern wir uns: Das Carry-Flag wird gesetzt, wenn bei der Addition ein Übertrag aus Bit 7 des Akkus auftritt. Und das ist ja hier der Fall; das Ergebnis ist größer als $FF.

Die CPU springt nun, wenn das Carry-Flag nicht gesetzt ist, in die Adresse $100E (BCC $100E). Wenn das Carry-Flag dagegen gesetzt ist, wird der Inhalt der Speicherzelle $D1 um 1 erhöht (incrementiert). In $D0 und $D1 steht nun unser richtiges Ergebnis, nämlich $010E.

Nun möchten wir ein Programm schreiben, das es ermöglicht, eine bestimmte Anzahl von Zeichen über die Tastatur einzugeben. Wenn die Anzahl erreicht wird, sollen die Zeichen auf dem Bildschirm ausgegeben werden.

Dazu müssen wir jetzt etwas Grundsätzliches anmerken: Der Plus/4 besitzt 64 Kbyte ROM. Das sind also Programme, die fest im Computer eingebaut sind. 32 Kbyte sind im BASIC- und Betriebssystem-ROM untergebracht. Hierbei handelt es sich um alle möglichen Routinen. Sie werden sicher schon merken, worauf wir hinaus möchten, nämlich auf das Verwenden dieser Routinen in eigenen Programmen. Warum sollten wir also eine Tastaturabfrageroutine entwickeln, wenn doch schon so eine Routine im ROM enthalten ist. Ebenso verhält es sich mit der Ausgabe auf dem Bildschirm oder anderen Ausgabegeräten.

Hierzu muß man nur eines wissen: Die Einsprungsadressen und Besonderheiten dieser Routinen. Wir werden Ihnen hier und in den folgenden Abschnitten die wichtigsten Betriebssystemroutinen mit ihren Einsprungsadressen zeigen. Kommen wir also zu unserem Ein- und Ausgabeprogramm.

20.9.1 Beispielprogramm »Tastaturabfrage«

Mit dem Programm wollen wir genau fünf Zeichen von der Tastatur eingeben. Diese sollen auf dem Bildschirm ausgegeben werden.

```
. 1000 A2 05    LDX #$05    ;Zähler
. 1002 20 CF FF JSR $FFCF   ;BASIN, Eingabe eines Zeichens
. 1005 95 D0    STA $D0,X   ;Zeichen abspeichern
. 1007 CA       DEX         ;Zähler minus 1
. 1008 D0 F8    BNE $1002   ;Zähler ist noch nicht auf 0
. 100A A2 05    LDX #$05    ;Zähler für Ausgabe
. 100C B5 D0    LDA $D0,X   ;Zeichen laden
. 100E 20 D2 FF JSR $FFD2   ;BSOUT, Ausgabe eines Zeichens
. 1011 CA       DEX         ;Zähler minus 1
. 1012 D0 F8    BNE $100C   ;Zähler noch nicht auf 0
```

```
. 1014 00       BRK          ;zurück zum TEDMON
```

Zwei Routinen des Betriebssystems haben wir hier kennengelernt, die Eingabe (BASIN $FFCF) und die Ausgabe (BSOUT $FFD2). Mit diesen Routinen wird das Zeichen entweder dem Akku übergeben, bzw. das Zeichen im Akku wird auf dem Bildschirm ausgegeben.

Nun wieder zu unserem Programm. Wir können mit diesem Programm auch mehr als fünf Zeichen eingeben, allerdings werden nur die ersten fünf Zeichen gespeichert. Das ergibt mitunter auf dem Bildschirm ein mächtiges Durcheinander.

Also werden wir eine andere Methode zur Tastaturabfrage brauchen. Dazu überlegen wir einmal, was in unserem Plus/4 so vorgeht. Eigentlich wird die Tastatur ja ständig abgefragt, und zwar mit dem Interruptprogramm. Dabei wird der aktuelle Tastencode in der Speicherzelle $C6 gespeichert. Die Anzahl der gedrückten Tasten (seit dem letzten RETURN), die im sogenannten Tastaturpuffer ($0527 bis $0530) gespeichert werden, steht in der Speicherzelle $EF. Diese Zero-Page-Adresse brauchen wir nur abzufragen. Wenn sie den Wert '5' enthält, lesen wir einfach die ersten fünf Zeichen aus dem Tastaturpuffer aus. Das Programm dazu sieht dann folgendermaßen aus:

```
. 1000 A2 05    LDX #$05          ;schon fünf Tasten gedrückt?
. 1002 E4 EF    CPX $EF
. 1004 D0 FC    BNE $1002         ;nein
. 1006 A2 00    LDX #$00          ;ja, Ausgabe der Zeichen
. 1008 BD 27 05 LDA $0527,X
. 100B 20 D2 FF JSR $FFD2         ;BSOUT
. 100E E8       INX
. 100F E0 05    CPX #$05          ;fünf Zeichen ausgegeben?
. 1011 D0 F5    BNE $1008         ;nein
. 1013 00       BRK
```

20.9.2 Beispielprogramm »Grafikmuster«

Der Bildschirm unserer hochauflösenden Grafik soll mit einem Muster hinterlegt werden. Dazu schreiben wir ein Programm, das wir von BASIC aus mit einem SYS-Befehl starten. Damit wir auch unterschiedliche Muster erzeugen können, geben wir hinter dem SYS-Befehl noch eine beliebige Nummer an. Hier zunächst das Programm. Das Programm belegt den Speicherbereich ab $065E. Dieser Bereich (bis $06EB) steht Ihnen für Maschinenspracheprogramme zur Verfügung, ohne daß BASIC-Speicher verlorengeht. In unseren Programmbeispielen brauchen wir auch Speicherzellen der Zero-Page. Wir können die Speicherzellen von $D0 bis $E8 verwenden.

```
. 065E 20 DE 9D JSR $9DDE   ;Übernimmt Wert bis 65535 aus BASIC
. 0661 A0 00    LDY #$00
. 0663 A2 20    LDX #$20
. 0665 84 D0    STY $D0     ;Startadresse Grafikspeicher Low
. 0667 86 D1    STX $D1     ;      "              "        High
. 0669 B1 D0    LDA ($D0),Y ;Grafikspeicher mit dem Wert in $14
. 066B 45 14    EOR $14     ;EXCLUSIV-ODER verknüpfen
. 066D 91 D0    STA ($D0),Y ;und wieder abspeichern
. 066F A5 14    LDA $14     ;Inhalt von $14
. 0671 49 FF    EOR #$FF    ;negieren
. 0673 85 14    STA $14
. 0675 E6 D0    INC $D0     ;Inhalt von $D0 plus 1
. 0677 D0 F0    BNE $0669   ;Verzweigen, wenn ungleich Null
. 0679 E6 D1    INC $D1     ;Inhalt von $D1 plus 1
. 067B A6 D1    LDX $D1     ;und mit dem Wert $40
. 067D E0 40    CPX #$40    ;vergleichen. Verzweige nach
. 067F D0 E8    BNE $0669   ;$0669, wenn Wert kleiner als $40
. 0681 60       RTS         ;Zurück zum BASIC
```

Wir verwenden auch hier wieder eine interessante Betriebssystemroutine, nämlich GETADR ($9DDE). Diese Routine holt einen Wert bis 65535 aus dem BASIC-Text und speichert ihn in $14 (Low-Byte) und $15 (High-Byte) ab. Wir verwenden in unserem Programm nur das Low-Byte. Dieses Byte speichern wir im Grafikspeicher ab. Dabei verwenden wir eine EXCLUSIV-ODER-Verknüpfung, um den eventuell vorhandenen Inhalt des Grafikspeichers nicht zu zerstören. Die Adresse des Grafikspeichers steht in $D0 und $D1. Diese Adresse wird bis zum Wert $3FFF hochgezählt. Danach kehrt das Programm wieder zurück ins BASIC. Um das Muster etwas interessanter zu machen, verknüpfen wir unseren Wert in $14 mit dem Wert #$FF mit einer EXCLUSIV-ODER-Funktion. Dadurch werden die einzelnen Bits negiert. Das Programm starten Sie von BASIC mit:

SYS 1630,X

X = beliebiger Wert von 0 bis 255

Beispiel:

```
10 GRAPHIC1,1
20 CIRCLE,100,100,100
30 FOR X = 0 TO 255
40 SYS 1630,X
50 GETKEY A$
60 NEXT
```

Wenn Sie den Grafikbildschirm zweimal hintereinander mit demselben Wert verknüpfen, erhalten Sie den Ursprungsbildschirm. Sie können mit dem Maschinenprogramm natürlich noch etwas experimentieren. Z.B. muß der Wert in $14 nicht jedesmal mit #$FF verknüpft werden. Wenn Sie das nicht möchten, setzen Sie einfach in

$0671 und in $0672 den Befehl NOP ein. Das ergibt dann einen gewissen plastischen Effekt. In $066B können Sie auch ORA $14 oder AND $14 einsetzen. Experimentieren Sie mal ein wenig. Dadurch können Sie die Maschinensprache am besten kennenlernen.

20.9.3 Beispielprogramm »Grafik umspeichern«

Unser Grafikbildschirm soll im RAM in den Bereich $DD00 bis $FCFF abgespeichert werden. Gleichzeitig soll der dort enthaltene Speicherinhalt in den Grafikspeicher (also $2000 bis $3FFF) transportiert werden. Wir tauschen also den Speicherinhalt dieser zwei Bereiche aus.

```
. 065E A9 40    LDA #$40     ;High-Byte Grafikspeicherende + 1
. 0660 85 D4    STA $D4
. 0662 A0 00    LDY #$00     ;Low-Byte Grafikspeicher 1 und 2
. 0664 A2 20    LDX #$20     ;High-Byte Grafikspeicher 1
. 0666 84 D0    STY $D0
. 0668 86 D1    STX $D1
. 066A A2 DD    LDX #$DD     ;High-Byte Grafikspeicher 2
. 066C 84 D2    STY $D2
. 066E 86 D3    STX $D3
. 0670 78       SEI
. 0671 8D 3F FF STA $FF3F    ;RAM einschalten
. 0674 B1 D2    LDA ($D2),Y ;Byte aus Speicher 2 lesen
. 0676 48       PHA          ;und auf Stack
. 0677 B1 D0    LDA ($D0),Y ;Byte aus Speicher 1 lesen
. 0679 91 D2    STA ($D2),Y ;und in Speicher 2 schreiben
. 067B 68       PLA          ;Wert vom Stack holen
. 067C 91 D0    STA ($D0),Y ;und in Speicher 1 schreiben
. 067E C8       INY          ;Y-Register + 1
. 067F D0 F3    BNE $0674    ;kein Übertrag
. 0681 E6 D1    INC $D1      ;bei Übertrag Anfangsadresse + 1
. 0683 E6 D3    INC $D3
. 0685 A5 D1    LDA $D1      ;Grafikspeicherende erreicht?
. 0687 C5 D4    CMP $D4
. 0689 D0 E9    BNE $0674    ;nein
. 068B 8D 3E FF STA $FF3E    ;ja, ROM einschalten
. 068E 58       CLI
. 068F 60       RTS          ;zurück in das BASIC
```

In diesem Programm wird das Interrupt-Flag gesetzt. Das ist unbedingt notwendig, da wir das Betriebssystem-ROM ausschalten müssen. Erinnern Sie sich noch einmal an die Aufteilung der Speicherbereiche des Plus/4: Unterhalb des BASIC- und Betriebssystem-ROM liegt der RAM-Speicher. Diesen können wir durch Ansprechen der Speicherzelle $FF3F aktivieren. Nur darf dann aber kein Interrupt mehr erfolgen, da das ROM, in dem die Interruptroutine liegt, ja ausgeschaltet ist. Lesen Sie dazu bitte auch das Kapitel 21.

In unserem Programm verwenden wir dieses Mal zwei Zeiger: $D0 $D1 und $D2 $D3. In $D4 steht der Maximalwert + 1 des High-Byte des Grafikspeichers. Wir laden nun zunächst einen Wert aus dem Zielbereich (ab $DD00) und speichern diesen im Stapelspeicher ab (Befehl PHA). Nun laden wir den Wert der entsprechenden Speicherzelle des Grafikspeichers und speichern diesen im Zielbereich ab. Anschließend holen wir den Wert aus dem Stapelspeicher zurück (PLA) und speichern diesen im Grafikspeicher ab. Somit werden die Speicherbereiche $2000-$3FFF und $DD00-$FCFF einfach ausgetauscht.

Bevor Sie dieses Programm starten, müssen Sie unbedingt beachten: Der Speicherbereich ab $DD00 muß für BASIC gesperrt werden. Das muß geschehen, bevor Sie Ihr BASIC-Programm eingeben. Dazu tippen Sie ein:

POKE 56,221:CLR

Sie haben jetzt noch rund 52500 Bytes für Ihre Programme zur Verfügung, können nun aber zwischen zwei Grafikbildschirmen mit SYS 1630 hin- und herschalten.

20.9.4 Beispielprogramm »OLD-Routine«

Sie haben sicher auch schon einmal den Fehler gemacht ein BASIC-Programm mit NEW zu löschen, obwohl das Programm noch gar nicht gelöscht werden sollte. Möglicherweise hatten Sie das Programm noch nicht abgespeichert. Oder Sie mußten Ihr Programm durch Drücken der RESET-Taste beenden. Wenn Sie dabei vergessen die RUN/STOP-Taste zu drücken, ist das Programm weg - gelöscht!

Moment, moment, es ist noch nicht alles verloren, denn genaugenommen ist das Programm nicht gelöscht. Es sind nur die ersten drei Bytes des BASIC-Speichers auf den Wert 0 gesetzt worden. Das Programm selbst ist noch da, Sie können es nur nicht mehr listen oder starten. Nun, holen wir das Programm doch einfach zurück. Zwei Möglichkeiten können wir Ihnen dazu zeigen.

Zunächst die einfachere Methode:

Geben Sie im Direktmodus folgendes ein:

POKE 4097,1:DELETE 1

Ihr Programm ist jetzt wieder vollständig im Speicher vorhanden. Dabei muß man aber beachten, daß der Anfang des BASIC-Speichers vor der Eingabe dieser Befehle auf dem gleichen Wert liegt, wie vor dem Drücken der RESET-Taste. Hatten Sie zum

Beispiel vorher die Grafik eingeschaltet, so müssen Sie auch einmal die Grafik einschalten und wieder ausschalten. Dann erst dürfen Sie die genannten Befehle eingeben.

Hier nun die zweite Methode ein gelöschtes Programm zurückzuholen. Dabei werden wir Ihnen auch erklären, wie so etwas funktioniert. Geben Sie nach dem RESET oder NEW bitte folgendes ein:

```
POKE PEEK(43)+PEEK(44)*256,1:SYS34840:SYS34891:CLR
```

Jetzt können Sie Ihr Programm wieder listen, abspeichern oder starten, es steht wieder unverändert im Speicher. Sollte sich dagegen das Programm bei einem Probelauf selbst zerstört haben (durch irgendwelche POKEs), haben Sie jetzt natürlich auch nur die zerstörte Version im Speicher, aber diesen Fall, der sicher jeden Programmierer verzweifeln läßt, wollen wir einmal vergessen.

Wenn Sie die oben angegebene Befehlszeile nicht jedesmal eingeben möchten, können Sie die Routine auch als Maschinenprogramm auf Kassette oder Diskette abspeichern und bei Bedarf laden und mit einem SYS-Befehl starten.

Wir nennen diese Routine »OLD-Routine«. Es gibt nämlich BASIC-Interpreter, die den Befehl OLD kennen, mit dem man »gelöschte« Programme retten kann. Wir schreiben das Programm wieder in den Speicherbereich ab $065E:

```
. 065E A0 00    LDY #$00      ;Koppeladresse simulieren
. 0660 A9 01    LDA #$01      ;und in
. 0662 91 2B    STA ($2B),Y   ;BASIC-Speicher speichern
. 0664 20 18 88 JSR $8818     ;neue Koppeladresse berechnen
. 0667 20 4B 88 JSR $884B     ;Variablenanfang neu berechnen
. 066A 4C 9A 8A JMP $8A9A     ;BASIC-Befehl CLR
```

Was passiert denn nun durch die OLD-Routine? Stellen Sie sich vor, Sie haben ein BASIC-Programm im Speicher und ändern die erste Programmzeile. In diesem Fall werden alle Koppeladressen neu berechnet. Genau das machen wir hier auch. Wir gaukeln unserem Computer eine Änderung im BASIC-Programm vor und mit der Routine ab $8818 werden nun alle Koppeladressen, auch Linkadressen genannt, neu berechnet. Das geht solange, bis im Speicher zweimal der Wert #$00 erscheint. Das ist nämlich das Zeichen für das Programmende. Nun werden Sie auch begreifen, wieso wir in unserer Routine eine Koppeladresse »simulieren«. Dabei wird nichts weiter gemacht, als dem Computer gesagt, hier ist das Programm noch nicht zu Ende. Erinnern Sie sich bitte, durch NEW werden die ersten drei Bytes im Speicher auf #$00 gesetzt, und das ist ja das Zeichen für Programmende.

Nach der Berechnung der Koppeladressen muß noch der Zeiger für den Variablenanfang neu berechnet werden. Anschließend wird die CLR-Routine aufgerufen, die uns alle übrigen Zeiger wieder normalisiert.

Speichern Sie das Maschinenprogramm nun mit dem Monitor unter dem Namen »OLD SYS 1630« ab. Sie können es jetzt jederzeit bei Bedarf wieder laden mit:

LOAD "OLD SYS 1630",8,1 (von Diskette)
LOAD "OLD SYS 1630",1,1 (von Kassette)

und starten mit:

SYS 1630

Wenn vor dem Löschen Ihres Programms die hochauflösende Grafik eingeschaltet war und Sie die RESET-Taste gedrückt haben, sollten Sie vor dem Aufrufen unserer OLD-Routine die Grafik ein- und wieder ausschalten:

GRAPHIC1:GRAPHIC0

Dadurch wird der Anfang des BASIC-Speichers wieder auf den richtigen Wert gesetzt (bei der hochauflösenden Grafik $4000 statt $1000, wenn die Grafik nicht eingeschaltet war).

Soviel zur Einführung in die Maschinensprache. Sie werden sicher schon jetzt festgestellt haben, daß die Maschinensprache des 6502 (7501) nicht so kompliziert ist, wie sie auf dem ersten Blick erscheinen mag.

Im nächsten Kapitel beschäftigen wir uns weiter mit der Maschinensprache, nur werden wir dann noch etwas tiefer in die Materie einsteigen. Wir nennen das Kapitel daher »Fortgeschrittene Maschinenspracheprogrammierung auf dem Plus/4«. Dort beschreiben wir die Interruptprogrammierung, das sogenannte »Bankswitching« und zeigen Ihnen anschließend weitere Programmbeispiele und wichtige Betriebssystemroutinen. Wenn Sie selbst in der Maschinensprache programmieren möchten, benutzen Sie bitte unbedingt die Befehlsübersicht in der Anlage.

21 Fortgeschrittene Maschinensprachprogrammierung auf dem Plus/4

In diesem Kapitel werden wir noch tiefer in den Plus/4 eindringen. Ziel ist es, Ihnen weitere Möglichkeiten dieses Computersystems aufzuzeigen. So werden Sie unter anderem auch die Speicherverwaltung des Plus/4 kennenlernen. Für spätere Programme ist dieses Thema ein absolutes Muß. Wir haben uns auch in diesem Kapitel bemüht die erforderlichen Grundlagen so verständlich wie möglich zu erklären. Also, fangen wir gleich an.

21.1 Interruptprogrammierung

Eine der wohl faszinierendsten Programmierarten stellt wohl die Interruptprogrammierung dar. In diesem Kapitel werden Sie erfahren, was Interrupt eigentlich heißt, wie er erzeugt wird, was er bewirkt, welche Interruptmöglichkeiten es beim Plus/4 gibt, und wie man diese last not least programmiert. Dies hört sich für manch einen sicher schlimm an, das ist es aber eigentlich gar nicht.

21.1.1 Was ist ein Interrupt?

Interrupt kommt aus dem Englischen und heißt übersetzt Unterbrechung. Unterbrochen wird das laufende Programm Ihres Plus/4. Wenn Sie Ihren Rechner einschalten, blinkt der Cursor, und der Plus/4 wartet auf Arbeit. Das Warten ist aber kein Stillstand, denn es wird durch eine Warteschleife des Betriebssystems hervorgerufen. Der Prozessor arbeitet also, obwohl Sie dies nicht so ohne weiteres feststellen können.

Wenn Sie nun eine Taste drücken, erscheint das entsprechende Zeichen auf dem Bildschirm. Wer aber glaubt, daß diese Arbeiten von der Warteschleife ausgeführt werden, der befindet sich auf dem viel zitierten Holzweg.

Die Warteschleife wird zigmal in der Sekunde unterbrochen. Der Prozessor nimmt während dieser Unterbrechung die Arbeit an einer anderen Stelle des Betriebssystems auf. Dort wird dann unter anderem die Tastatur abgefragt. Ist zu diesem Zeitpunkt eine Taste gedrückt, so wird der entsprechende Tastencode ermittelt. Dies ist aber nur eine der vielen Aufgaben, die der Prozessor während des Interrupts zu erfüllen hat. Das Gleiche passiert, wenn Sie den Rechner ein Programm abarbeiten lassen. Ihm ist es dabei egal, ob es sich um ein BASIC- oder um ein Maschinenprogramm handelt. Der Interrupt erfolgt pünktlich und, was noch wichtiger ist, ständig.

Dieser Umstand macht die Interruptprogrammierung für uns sehr interessant. Gelingt es, eigene Programme während der Interruptphase ablaufen zu lassen, so können wir viele Dinge einfacher programmieren. Durch die hohe Geschwindigkeit der Maschinensprache entsteht dann der Effekt, daß scheinbar zwei Programme gleichzeitig laufen. Am Ende dieses Kapitels werden wir dies in einem Beispielprogramm, welches einen Tastaturpiep verursacht, demonstrieren.

Mit diesen Erklärungen sind vielleicht die ersten Fragen beantwortet, aber die nächste Frage liegt Ihnen sicher schon auf der Zunge. Woran erkennt denn nun der Prozessor, daß er ein anderes Programm abarbeiten soll?

21.1.2 Die Interrupterzeugung

Nehmen Sie es uns bitte nicht übel, aber wir müssen Sie zunächst mit zwei weiteren Fremdworten bekannt machen. Es sind Hardwareinterrupt und Softwareinterrupt. Es folgt zunächst die Erklärung des Hardwareinterrupts.

Zur Beschreibung dieser Interruptart benötigen wir Angaben über die Hardware.

Der Plus/4 besteht nicht nur aus dem Prozessor und den RAM- und ROM-Bausteinen, sondern er enthält noch weitere Chips. Einer dieser Chips ist für die einwandfreie Funktion des Rechners besonders wichtig. Sein Name ist TED. Diese Abkürzung steht für TExt Display. Der TED ist für Aufgaben wie der Speicherverwaltung, dem Bildschirmaufbau, der Tastatur- und Joystickabfrage und auch für die Auslösung eines Hardwareinterrupts verantwortlich. Er verfügt über einige RAM-Speicheradressen, die genau wie »normale« RAM-Adressen angesprochen werden können, das heißt, es können sowohl Werte in ihnen gespeichert als auch ausgelesen werden. Diese Adressen liegen im Bereich von $FF00 bis $FF3F.

Ein Chip wie der TED, der so viele Aufgaben zu erfüllen hat, weist eine ganze Anzahl von Anschlüssen auf. Einer, Pin acht, heißt IRQ (Interrupt request, auf deutsch Programmunterbrechung). Werden während des Betriebs unseres Plus/4 gewisse Bedingungen erfüllt, so macht sich dies an einer Spannungsänderung am Pin acht bemerkbar. Welche Bedingungen hierzu erforderlich sind, wird später erklärt.

Unser Prozessor verfügt ebenfalls über einen »IRQ Pin«, es ist der Pin drei. Beide Pins sind miteinander verbunden, und so wird auch dem Prozessor die Spannungsänderung, hervorgerufen vom TED, mitgeteilt. Diese Spannungsänderung nennt man auch Interruptanforderung. Es gibt auch noch einen weiteren Chip der Interruptanforderungen stellen kann, der Ein-/Ausgabebaustein. Wir werden in diesem Kapitel aber nur Interrupts behandeln, die den TED betreffen und gehen daher nicht näher auf diesen Chip ein.

Um auf eine Interruptanforderung richtig zu reagieren, besitzt der Prozessor ein eigenes, in ihm eingebautes Programm. Soll ein Interrupt erfolgen, werden von ihm folgende Schritte unternommen: Der Befehl, der gerade in Arbeit ist, wird noch ausgeführt. Danach wird das Interrupt-Flag des Statusregisters überprüft. Ist es gesetzt, das heißt hat es den Wert eins, dann wird die Anforderung auf eine Progammunterbrechung ignoriert, und der nächste Befehl wird abgearbeitet. In den meisten Fällen wird aber das I-Flag nicht gesetzt sein, mit anderen Worten, es wird eine Programmunterbrechung zugelassen. In diesem Fall wird zunächst der Programmzähler (in Englisch heißt er Programmcounter oder kurz Pc) auf den Stapel gebracht. Dies geschieht deshalb, weil der Prozessor nach Beendigung des Interruptprogramms mit der Abarbeitung des Programms vor der Unterbrechung fortfahren soll. Damit das ursprüngliche Programm unter den gleichen Bedingungen wie vor dem Interrupt weiterarbeitet, muß auch das Statusregister auf den Stapel gerettet werden. Auch diese Arbeit wird, wie das Setzen der I-Flagge, automatisch ausgeführt. Jetzt passiert etwas Besonderes:

Der Programmzähler (Pc) wird mit dem Inhalt der Adressen $FFFE und $FFFF geladen. Es sind die Adressen, die der Prozessor gerade noch adressieren kann. Da man den Plus/4 auf verschiedene Banks umschalten kann, gehen wir beim Folgenden davon aus, daß das Betriebssystem eingeschaltet ist. Diese Adressen (beim Plus/4 haben sie den Inhalt $B3 und $FC) enthalten einen Zeiger, der auf die zu bearbeitende Interruptroutine zeigt. Beim Plus/4 wird also zur Adresse $FCB3 verzweigt und dort die Abarbeitung des Interruptprogramms begonnen. Da dieser Interrupt durch eine Leitung signalisiert wird, nennt man ihn Hardwareinterrupt.

Kommen wir zum Softwareinterrupt:

Die Auslösung dieser Interruptart erfolgt durch den BRK-Befehl des Prozessors. Wenn er diesen Befehl abarbeiten soll, läuft das schon oben Beschriebene fast ganz genauso ab. Es gibt aber zwei Eigenarten. Da der BRK-Befehl meist zum Austesten von Maschinenprogrammen benötigt wird und dabei oft als Ersatz eines Zwei-Byte Befehls dient, wird der Programmzähler als Pc+2 auf den Stapel gebracht. Nach der Rückkehr vom Interrupt wird mit dem Befehl fortgefahren, der in der zweiten, dem BRK-Befehl folgenden, Adresse zu finden ist. Die Verzweigung erfolgt auch bei BRK zur Adresse, die in den Speicherstellen $FFFE und $FFFF steht. Als Unterscheidung zum Hardwareinterrupt wird neben dem IRQ-Flag auch noch die BREAK-Flagge im Statusregister gesetzt.

Unser Prozessor unterscheidet sich, was die Interrupts angeht, von den 6502 Typen. Diese besitzen noch eine NMI-Leitung. NMI heißt: nicht maskierbarer Interrupt und bedeutet nichts anderes, als daß man diesen Interrupt nicht sperren kann. Er wird also immer ausgeführt. Diese Möglichkeit besitzt der 7501, der in unserem Plus/4 eingebaut ist.

Falls Sie nicht alles verstanden haben, so ist das auch nicht weiter schlimm. Es kam uns aber darauf an, Ihnen auch einmal etwas von den Dingen zu erklären, die in den Bereich der Hardware gehen. Lesen Sie noch einmal nach, wenn Sie selbst Interruptprogramme schreiben wollen.

21.1.3 Die Interruptprogrammierung beim Plus/4

In diesem Kapitel werden wir uns ausschließlich mit der Programmierung des Hardwareinterrupts beschäftigen. Er ist auch der flexibelste der Interruptarten. Schalten Sie vor dem Weiterlesen bitte den Maschinensprachmonitor ein. Jetzt folgt:

D FCB3 FCBB

In der Routine, die Sie jetzt auf dem Bildschirm sehen, werden die Registerinhalte auf den Stapel gerettet. Erst der AKKU, dann das X-Register und dann das Y-Register. Mit STA $FD00 wird jetzt das BASIC-ROM eingeschaltet. Mit JMP $CE00 wird in die eigentliche Interruptroutine verzweigt. Führen auch wir diese Verzweigung durch.

D CE00 CE0B

Zuerst wird der AKKU mit dem Inhalt des Statusregisters geladen, der vor der Verzweigung in die Interruptroutine aktuell war und sich jetzt auf dem Stapel befindet. Dieser Inhalt wird mit $10 AND verknüpft. Wurde der Interruptaufruf durch ein

BREAK hervorgerufen, das heißt Bit vier des Statusregisters ist gesetzt, so ergibt sich ein Ergebnis, das ungleich null ist, und es erfolgt eine Verzweigung zur Adresse $CE0B. Ist das Ergebnis aber null, so wird ab Adresse $0C08 weitergemacht. In dieser Adresse wird ein indirekter Sprung zu der Adresse ausgeführt, die in den Speicheradressen $0314 und $0315 vorhanden ist.

An dieser Stelle können wir uns mit eigenen Routinen einhaken. Sehen wir uns den Inhalt dieser Adressen an.

M 0314 0315

Die Adresse lautet: $CE0E. Da sich die Adressen $0314 und $0315 im RAM befinden, können wir sie leicht selbst ändern und auf eine eigene Routine zeigen lassen. Wie das erfolgen muß, wird jetzt beschrieben. Geben Sie dazu die nachstehenden Zeilen mit dem Monitorprogramm ein.

```
.065E SEI
 LDA #$6C
 STA $0314
 LDA #$06
 STA #$0315
 CLI
 RTS
```

Mit diesen Befehlen verbiegen wir den Interrupteinsprung auf $066C. Dazu muß zuerst das I-Flag mit SEI gesetzt werden. Erinnern Sie sich bitte, wenn dieses Flag gesetzt ist, wird kein Interrupt zugelassen. SEI ist besonders wichtig, denn wird während des »Verbiegens« eine Interruptanforderung an den Prozessor gestellt, so verzweigt das Programm an eine undefinierte Speicherstelle. Ihr Computer ist »abgestürzt«.

Nachdem der Zeiger »verbogen« wurde, wird mit CLI der Interrupt wieder zugelassen und dann mit RTS unser kleines Programm beendet. Durch diese paar Programmzeilen ist es möglich, eigene Routinen, die ab Adresse $066C stehen, in den Interrupt einzubinden. Noch aber steht nichts Sinnvolles in den Adressen, und würden wir dieses Programm starten, so würde der Plus/4 unweigerlich nur Dummheiten machen. Geben wir also ein:

```
. 066C INC $FF19
 JMP $CE0E
```

Wenn Sie diese Zeilen eingegeben haben, verlassen Sie bitte den Monitor mit »X«. Es folgt SYS 1630. Auf Ihrem Bildschirm wird es jetzt wohl etwas unruhiger geworden sein. Bei jedem Interrupt wird durch unser Programm die Farbe des Bildschirmrandes geändert. Die Änderung wird durch das Hochzählen der Speicherstelle $FF19 hervorgerufen. Sie repräsentiert im TED das Register für die Rahmenfarbe. Mit JMP

$CE0E springen wir wieder in die eigentliche Interruptroutine. So bleiben auch alle anderen Funktionen des Rechners erhalten. Mögen Sie es noch bunter? Dann ändern Sie nach einem Reset das Programm wie unten:

```
. Ø66F INC $FF15
 JMP $CEØE
```

Nach SYS 1630 flackert auch der Hintergrund. Im TED stellt das Register $FF15 die Hintergrundfarbe dar. Durch das Einfügen dieser Zeile haben wir den Interrupt sogar sichtbar machen können. Sie werden erkennen, daß das untere Fünftel des Bildschirms immer eine andere Farbe hat als die vier restlichen Fünftel. Das liegt am Rasterinterrupt des Plus/4.

Obwohl schon wieder ein, für die meisten unserer Leser, unverständliches Wort auftaucht, sollten Sie das Buch nicht entnervt aus der Hand legen. Wir werden uns bemühen, auch das Folgende leichtverständlich zu erklären. Da wir ein neues Unterkapitel beginnen, bietet sich eine Pause geradezu an. Das, was jetzt beschrieben wird, geht wieder mal, wir können es nicht ändern, ans »Eingemachte«.

21.1.4 Die Interruptregister des TED

Wie eingangs erwähnt, besitzt der TED verschiedene RAM-Register, die sowohl gelesen als auch beschrieben werden können. Wir haben schon die Register $FF15 und $FF19 kennengelernt und diese im vorangegangenen Beispiel beliebig verändert. Wir kommen jetzt zu den Registern, die unmittelbar mit der Interrupterzeugung zu tun haben.

Das erste Register, das jetzt beschrieben wird, soll den Namen Interrupt Request Register bekommen. Dieses Register besteht aus acht Bit. Bis auf die Bit 0 und 5 haben die anderen eine Funktion zu erfüllen. Eine kleine Übersicht:

Interrupt Request Register $FF09

Bit	7	6	5	4	3	2	1	Ø
	IRQ	TIM 3	----	TIM 2	TIM 1	LITP	RAST	----

Fangen wir gleich mit der Erklärung der einzelnen Bit, und deren Bedeutung an.

Bit 0 .. Dieses Bit hat keine Funktion und kann daher beliebig verändert werden.
Bit 1 .. Signalisiert den Rasterinterrupt.
Bit 2 .. Soll einen vom Lightpen ausgelösten Interrupt signalisieren, da der Plus/4 aber über keinen derartigen Eingang verfügt, ist es ohne Bedeutung.
Bit 3 .. Zeigt an, daß der Interrupt durch Unterlauf von Timer 1 ausgelöst wurde.
Bit 4 .. Wie Bit 3, nur hier Unterlauf des Timers 2.
Bit 5 .. Keine Funktion.
Bit 6 .. Unterlauf des Timers 3.
Bit 7 .. Signalisiert das Auftreten eines Interrupts.

Zum besseren Verständnis wird auch gleich das nächste Register, wir wollen es ab jetzt Interrupt Mask Register nennen, erklärt. Es verfügt ebenfalls wieder über acht Bit.

Interrupt Mask Register $FF0A

```
Bit   7      6       5      4       3      2      1      0
     ---   TIM 3    ----   TIM 2   TIM 1   LITP   RAST   Bit 9
```

Bit 0 .. Dieses Bit ist Bit 9 des Rasterregisters.
Bit 1 .. Dieses Bit ermöglicht den Rasterinterrupt.
Bit 2 .. Soll einen Interrupt durch Lightpen ermöglichen.
Bit 3 .. Ist es gesetzt, so löst Timer 1 beim Unterlauf einen Interrupt aus.
Bit 4 .. Wie Bit 3, nur hier Timers 2.
Bit 5 .. Keine Funktion.
Bit 6 .. Interrupt bei Unterlauf des Timers 3.
Bit 7 .. Signalisiert das Auftreten eines Interrupts.

Der TED verfügt über drei Timer, die jeweils zwei Byte lang sind. Diese Timer werden kontinuierlich im Systemtakt (1 MHz) heruntergezählt. Man kann ihre Register sowohl lesen als auch mit einem neuen Startwert laden. Die Adressen der Register finden Sie im Anhang bei der Registerübersicht des TED. Wird der Timer 1 zum Beispiel mit dem Wert 60000 geladen, so wird er in einer Millionstel Sekunde um eins heruntergezählt. Nach 60 Millisekunden (ms) hat er den Wert null erreicht. Beim nächsten Systemtakt bekommt er den Wert 65535, es ist ein Unterlauf entstanden. Ist zu diesem Zeitpunkt das Bit drei des Mask Registers gesetzt, so werden ebenfalls Bit drei und sieben des Interrupt Request Registers auf eins gesetzt. Dadurch erfolgt eine Interruptanforderung an den Prozessor.

Bit sieben wird immer dann eins, wenn eines oder mehrere der anderen Bit des IRQ Registers eins werden. In der Interruptroutine kann jetzt sehr einfach festgestellt werden, von welchem Baustein ein Interrupt ausgelöst wurde, und welche Bedingun-

gen eingetreten sind. Bevor die Interruptroutine beendet wird, muß noch das gesetzte Bit im IRQ Register zurückgesetzt werden. Dies erreicht man durch einen Schreibzugriff auf das betreffende Bit. Das Zurücksetzen geht zum Beispiel durch :

```
LDA $FF09
STA $FF09
```

Vom Betriebssystem wird der Interrupt je nach Erfordernis mal von Timern oder auch vom Rasterstrahl ausgelöst. Das Rasterregister besteht aus neun Bits. Bit neun ist das Bit null des Mask Registers. Da der TED auch das Fernsehbild erzeugt, weiß er welche Zeile des Bildschirms gerade vom Rasterstrahl beschrieben wird. Stimmt diese Zeile mit dem Wert, gebildet aus Register $FF0B und Bit null des Registers $FF0A, überein, so wird bei gesetztem Bit eins des Mask Registers ein Interrupt ausgelöst. Arbeiten Sie nicht gerade mit angeschlossenen Geräten, so ist immer dieser Interrupt aktiv.

Das soll jetzt reichen. Sie haben eine ganze Menge über den Interrupt und seine Programmierung erfahren. Nach soviel grauer Theorie wollen Sie jetzt sicher wieder einmal praktisch tätig werden. Das folgende Programm erzeugt bei jedem Tastendruck einen Ton. Da es nur während des Interrupts abläuft, können Sie Ihren Plus/4 wie gewohnt benutzen. Geben Sie erst das BASIC-Ladeprogramm ein. Weitere Erklärungen folgen.

```
10 FOR I= DEC("065E") TO DEC("06AA")
20 READ A$
30 POKE I,DEC(A$)
40 B= B+DEC(A$)
50 NEXT
60 IF B<>10644 THEN PRINT"FEHLER IN DATAS"
100 DATA 78,A9,73,8D,14,03,A9,06,8D
101 DATA 15,03,A5,CD,85,D0,A5,CA,85
102 DATA D1,58,60,A5,CA,C5,D1,F0,04
103 DATA 85,D1,D0,08,A5,CD,C5,D0,F0
104 DATA 19,85,D0,A5,C6,C9,40,F0,11
105 DATA A9,FF,8D,0E,FF,A9,1F,0D,11
106 DATA FF,8D,11,FF,A9,04,85,D2,C6
107 DATA D2,D0,08,A9,E0,2D,11,FF,8D
108 DATA 11,FF,4C,0E,CE
```

Na, wunde Finger? Es ist ja nun geschafft. Speichern Sie das Programm vor dem Starten bitte unbedingt ab. Danach wird es mit RUN gestartet. Bei der Meldung »FEHLER IN DATAS« haben Sie sich irgendwo vertippt. Kontrollieren Sie daher noch einmal die Prüfsumme und dann die DATA-Zeilen. Unser Programm steht jetzt als Maschinenprogramm im Speicher und wird mit SYS 1630 gestartet. Nach »READY«

sollte nun bei jedem Tastendruck, außer CTRL und SHIFT, ein Ton ertönen. Sie können nun schon fast blind auf Ihrem Plus/4 schreiben, und müssen nicht immer den Bildschirm im Auge behalten.

Da durch das Programm der SOUND-Befehl nur noch eingeschränkt nutzbar ist, erfolgt die Erklärung des Programms an Hand des Assemblerlistings. Sie werden so das Programm leicht Ihren Bedürfnissen anpassen können.

```
. 065E 78       SEI
. 065F A9 73    LDA #$73
. 0661 8D 14 03 STA $0314
. 0664 A9 06    LDA #$06
. 0666 8D 15 03 STA $0315
. 0669 A5 CD    LDA $CD
. 066B 85 D0    STA $D0
. 066D A5 CA    LDA $CA
. 066F 85 D1    STA $D1
. 0671 58       CLI
. 0672 60       RTS
```

In diesem Teil des Programms wird der Interruptvektor »verbogen«. Er zeigt danach auf unser eigentliches Programm ab Adresse $0673. Zuerst muß aber unbedingt ein Interrupt mit SEI verhindert werden. Sollte nämlich während des »Verbiegens« eine Interruptanforderung kommen, so würde der Rechner sofort »abstürzen«. Um eine Synchronisation zwischen Zeichenausgabe und Ton zu erreichen, wird dann noch die derzeitige Position des Cursors in die Speicher $D0 und $D1 gerettet. Danach wird mit CLI der Interrupt wieder freigegeben und mit RTS ins BASIC zurückgekehrt. Die Initialisierung ist abgeschlossen. Erfolgt jetzt ein Interrupt, so wird zuerst in unser Programm verzweigt.

```
. 0673 A5 CA LDA $CA
. 0675 C5 D1 CMP $D1
. 0677 F0 04 BEQ $067D
. 0679 85 D1 STA $D1
. 067B D0 08 BNE $0685
. 067D A5 CD LDA $CD
. 067F C5 D0 CMP $D0
. 0681 F0 19 BEQ $069C
. 0683 85 D0 STA $D0
```

Unser Programm fragt nicht die Tastatur ab. Diese Aufgabe wird nach wie vor vom Betriebssystem bei der weiteren Interruptbehandlung erledigt. Daher geht die Tonerzeugung immer einen Interrupt nach. Es wird daher zuerst überprüft, ob sich die Position des Cursors auf dem Bildschirm verändert hat. Ist dies der Fall, werden die geänderten Werte in den Speicherstellen $D0 oder $D1 gespeichert.

```
. 0685 A5 C6 LDA $C6
. 0687 C9 40 CMP #$40
. 0689 F0 11 BEQ $069C
```

Da sich die Cursorposition auch bei LIST, PRINT oder anderen Befehlen ändert und dadurch ein entsprechend langer Dauerton entsteht, wird in diesem Programmteil überprüft, ob die Positionsänderung durch einen Tastendruck erfolgte. Der Plus/4 merkt sich bis zum nächsten Interrupt den Code der zuletzt gedrückten Taste in der Speicherstelle $C9.

```
. 068B A9 FF    LDA #$FF
. 068D 8D 0E FF STA $FF0E
. 0690 A9 1F    LDA #$1F
. 0692 0D 11 FF ORA $FF11
. 0695 8D 11 FF STA $FF11
```

Mit diesen Programmzeilen wird der Ton erzeugt. Dazu wird der Speicherstelle $FF0E die Frequenz übergeben. In $FF11 wird die Stimme eins des TED eingeschaltet und die höchste Lautstärke eingestellt.

```
. 0698 A9 04 LDA #$04
. 069A 85 D2 STA $D2
. 069C C6 D2 DEC $D2
. 069E D0 08 BNE $06A8
```

Mit diesen Befehlen legen wir auf einfache Weise die Tondauer fest. Sie beträgt drei Interruptzyklen, das heißt der Ton wird erst nach weiteren drei Interruptanforderungen wieder ausgeschaltet.

```
. 06A0 A9 E0    LDA #$E0
. 06A2 2D 11 FF AND $FF11
. 06A5 8D 11 FF STA $FF11
. 06A8 4C 0E CE JMP $CE0E
```

Zum Abschalten wird die Lautstärke wieder auf Null gesetzt und gleichzeitig auch Bit 4 des Registers $FF11, mit dem die Stimme eins ein- oder ausgeschaltet wird, gelöscht. Da unser Plus/4 weiterhin seinen Dienst wie gewohnt verrichten soll, wird dann in die eigentliche Interruptroutine des Betriebssystems verzweigt. Diese Verzweigung erfolgt immer, ganz egal ob der Ton ein- oder ausgeschaltet ist.

So, wir sind nun am Ende der Interruptprogrammierung angekommen. Wir hoffen, daß Ihnen unser Streifzug durch diese, den meisten wohl noch unbekannte Welt des Plus/4 Spaß gemacht hat. Sie sollten auf jeden Fall noch weiter experimentieren, denn wir können Ihnen immer nur einen kleinen Teil der Möglichkeiten aufzeigen, die der Rechner bietet. Im folgenden Kapitel über das Bankswitching des Plus/4 werden Sie erneut mit diesem Thema in Berührung kommen.

21.2 Das Bankswitching

Der zunehmende Preisverfall für Speicherbauteile ließ die Forderung aufkommen, größere Speicher in die Computer einzubauen. Dabei stieß man bei den Computern mit 16-Bit-Adressbussen sehr schnell an die Grenze des Möglichen. Wir haben bereits in den Kapiteln 2 und 3 festgestellt, daß mit 16 Bit maximal 64 Kbyte Speicher adressiert werden kann. Nun besitzt der Plus/4 bereits 64 Kbyte RAM. Dazu kommen noch 64 Kbyte ROM. Nun fragt man sich natürlich, wie ist so etwas überhaupt möglich? Nun, das Zauberwort, mit dem diese Probleme gelöst werden können, heißt Bankswitching. Sehen Sie sich dazu bitte das Bild 21.1 auf der nächsten Seite an.

Grundsätzlich lassen sich mit den 16 Adreßleitungen »nur« 64 Kbyte Speicher adressieren. Aber es wäre doch möglich, einen Bereich des Speichers (z. B. des RAMs) auszuschalten und dafür ein ROM einzuschalten. Genau das wird mit dem Bankswitching erreicht. An unserem Bild erkennen wir das daran, daß die Speicher einfach übereinander liegen. So ist zum Beispiel die Speicheradresse 32768 oder $8000 einmal als RAM-Speicherzelle vorhanden. Zum anderen ist diese Adresse aber auch im BASIC-ROM, im unteren ROM der eingebauten Software und noch maximal zwei Mal in externen, also am Expansionsport angeschlossenen, Speicherbauteilen vorhanden.

Zwischen diesen Bauteilen kann beliebig umgeschaltet werden. Und das nennt man Bankswitching (Bereichsumschaltung). Der Computer »sieht« immer nur seine 64 Kbyte Speicher. Ihm ist es völlig egal aus welchen Bauteilen die 64 Kbyte bestehen und wie zwischen diesen Bauteilen hin- und hergeschaltet wird, Hauptsache, die CPU bekommt stets logische und erlaubte Befehle.

In diesem Kapitel zeigen wir Ihnen, wie Sie zwischen den Speicherbauteilen umschalten können, und welche Probleme das Bankswitching auch mit sich bringen kann.

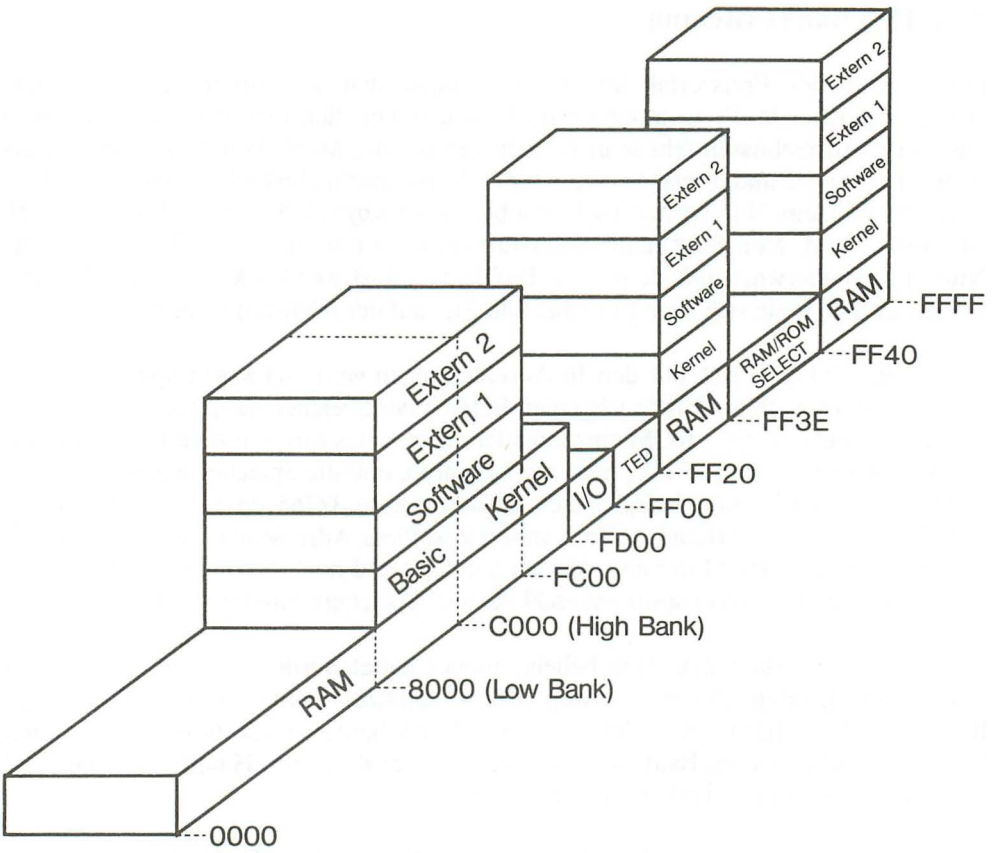


Bild 21.1: *Speicheraufbau des Plus/4*

Zum Glück muß man nicht mit irgendwelchen Schaltern zwischen den Speicherbereichen umschalten. Das geschieht vielmehr sehr elegant per Software, bzw. durch einfaches Ansprechen bestimmter Speicheradressen, die hier als »Schalter« dienen.

Zunächst einmal kann grundsätzlich zwischen RAM und ROM umgeschaltet werden. Das geschieht durch einen Schreibzugriff auf die Speicheradressen $FF3E und $FF3F.

. STA $FF3E schaltet ROM ein
. STA $FF3F schaltet RAM ein

Wenn das RAM eingeschaltet ist, belegt dieses die Adressen von $0000 bis $FCFF, $FF20 bis $FF3D und $FF40 bis $FFFF. Die übrigen Adressen werden belegt durch den TED und die I/O-Bauteile.

Achtung: Wenn das RAM eingeschaltet werden soll, muß vorher unbedingt der Interrupt ausgeschaltet werden. Das geschieht durch das Setzen des I-Flags (Maschinensprachbefehl SEI, $78).

Für die Ein- und Ausgabebausteine sind immer die Adressen $FD00 bis $FEFF reserviert. Der TED belegt immer die Adressen $FF00 bis $FF1F und die Adressen $FF3E und $FF3F. Beim Umschalten zwischen ROM und RAM bleibt der Akkuinhalt übrigens erhalten.

Wenn nun mit STA $FF3E in das ROM umgeschaltet wird, so wird das gerade aktuelle ROM eingeschaltet. Aber auch jetzt werden die Adressen $FD00 bis $FEFF, $FF00 bis $FF1F und $FF3E und $FF3F von den Ein- Ausgabebauteilen und vom TED belegt. Diese Bereiche werden also niemals vom RAM oder ROM eingenommen.

Nun haben wir gerade vom »aktuellen« ROM gesprochen. Wie Sie unserem Bild entnehmen können, besteht die Möglichkeit zwischen vier verschiedenen ROMs umzuschalten. Dabei wird noch zusätzlich die Aufteilung zwischen Lowbank und Highbank gemacht. Lowbank belegt die Adressen von $8000 bis $BFFf, und Highbank belegt die Adressen $C000 bis $FFFF.

Im Plus/4 ist eine Umschaltlogik, bestehend aus einem 74LS175-Chip, einem 74LS27 und einem 74LS139, eingebaut. Diese, auf den ersten Blick etwas kompliziert aussehende Schaltung, ermöglicht es uns, zwischen 16 verschiedenen Speicherkonfigurationen umzuschalten. So ist es z. B. möglich, im Adressbereich $8000 bis $BFFF das BASIC-ROM einzuschalten und gleichzeitig im Bereich $C000 bis $FFFF ein externes ROM oder RAM einzuschalten, das am Expansionsport angeschlossen ist. Zwischen den 16 verschiedenen Möglichkeiten kann man umschalten, indem man die Adressen $FDD0 bis $FDDF anspricht. Welche Speicheradresse welche Konfigura-

tion einschaltet, entnehmen Sie bitte der folgenden Tabelle (in der Tabelle erscheint der Begriff »Kernel«, es handelt sich dabei um das Betriebssystem).

Tabelle 21.1: *Speicherkonfigurationen mit den Einschaltadressen*

Befehl	Lowbank ($8000-$BFFF)	Highbank ($C000-$FFFF)
. STA $FDD0	BASIC-ROM	Kernel-ROM
. STA $FDD1	eingeb. Software Low	Kernel-ROM
. STA $FDD2	ext. Speicher Low 1	Kernel-ROM
. STA $FDD3	ext. Speicher Low 2	Kernel-ROM
. STA $FDD4	BASIC-ROM	eingeb. Software High
. STA $FDD5	eingeb. Software Low	eingeb. Software High
. STA $FDD6	ext. Speicher Low 1	eingeb. Software High
. STA $FDD7	ext. Speicher Low 2	eingeb. Software High
. STA $FDD8	BASIC-ROM	ext. Speicher High 1
. STA $FDD9	eingeb. Software Low	ext. Speicher High 1
. STA $FDDA	ext. Speicher Low 1	ext. Speicher High 1
. STA $FDDB	ext. Speicher Low 2	ext. Speicher High 1
. STA $FDDC	BASIC-ROM	ext. Speicher High 2
. STA $FDDD	eingeb. Software Low	ext. Speicher High 2
. STA $FDDE	ext. Speicher Low 1	ext. Speicher High 2
. STA $FDDF	ext. Speicher Low 2	ext. Speicher High 2

Dazu ein Beispiel:

```
. 1000 A2 05     LDX #$05     ;Nummer der Konfiguration
. 1002 9D D0 FD  STA $FDD0,X  ;Konfiguration einschalten
. 1005 86 FB     STX $FB      ;Nummer in $FB (für Interrupt)
. 1007 4C 03 80  JMP $8003    ;Start der Software
```

Mit unserem Beispiel wird die fünfte Möglichkeit aus unserer Tabelle eingeschaltet. Dabei handelt es sich im Bereich von $8000 bis $BFFF um das im Plus/4 eingebaute Software-ROM Low und im Bereich von $C000 bis $FFFF um das Software-ROM High. Somit werden also die im Plus/4 eingebauten Software-ROMs eingeschaltet. Die Startadresse der Software ist $8003.

In unserem Beispiel speichern wir in $FB die Nummer unserer gewünschten Speicherkonfiguration ab. Das hat folgenden Grund: Wie Sie wissen, wird jede 60stel Sekunde das Interruptprogramm abgearbeitet. Dabei wird zum Beispiel die Tastatur abgefragt. Nach diesem Interruptprogramm wird die Speicherkonfiguration, deren Nummer in $FB steht, eingeschaltet. Normalerweise steht in $FB der Wert $00. Der Tabelle können Sie entnehmen, daß dann das BASIC-ROM und das Betriebssystem-ROM eingeschaltet sind. Wenn wir in unserem kleinen Beispielprogramm den Wert in $FB nicht

entsprechend geändert hätten, würde nach dem nächsten Interrupt wieder die alte Speicherkonfiguration eingeschaltet werden.

Nun könnten wir auch auf die Idee kommen, einfach den Wert in $FB entsprechend zu ändern. Die Idee ist zwar nicht schlecht, aber in der Praxis funktioniert das leider nicht. Die gewünschte Speicherkonfiguration würde zwar tatsächlich eingeschaltet werden, nur wissen wir leider nicht, wann der nächste Interrupt kommt. Wenn dann schon vorher ein Sprung in das ROM erfolgt, landen wir zunächst noch im BASIC-ROM. Dann würde irgendwann der Interrupt folgen und das Software-ROM eingeschaltet werden. Ob unsere CPU dann im Software-ROM einen sinnvollen Befehl erwischt, ist reine Glückssache, auf jeden Fall wäre ein einwandfreier Start der Software nicht gewährleistet.

Zum Einschalten und Starten der ROMs gibt das Plus/4-Betriebssystem einige Hilfen. So schaltet z. B. die Routine ab $FCC9 das Modul ein, dessen Nummer im X-Register steht. Dann wird ein Sprung zu der Adresse gemacht deren Lowbyte in $02FE und Highbyte in $02FF steht. Hierzu wieder ein Beispiel:

```
. 1000 A9 03    LDA #$03
. 1002 A0 80    LDY #$80
. 1004 8D FE 02 STA $02FE ;Startadresse Low
. 1007 8C FF 02 STY $02FF ;Startadresse High
. 100A A2 05    LDX #$05  ;Modul (5), siehe Text
. 100C 86 FB    STX $FB   ;Modulnummer
. 100E 4C C9 FC JMP $FCC9 ;Modul einschalten und starten
```

Mit dieser Routine wird das Programm im Software-ROM ab Adresse $8003 gestartet.

In diesem Zusammenhang werden wir mit einem neuen Begriff konfrontiert, dem »Modul«. Im weiteren Verlauf dieses Kapitels und des Buches werden wir nur noch mit der Bezeichnung Modul arbeiten. Damit ist nichts weiter gemeint, als eine der möglichen Speicherkofigurationen. Wenn wir also aus unserer Tabelle die Möglichkeit 5 heraussuchen, so handelt es sich dabei um das Modul (5).

Nach dem Einschalten des Plus/4 ist das Modul (0) eingeschaltet (BASIC-ROM und Betriebssystem).

Wenn Sie sich noch einmal das Bild 21.1 ansehen, werden Sie eine weitere Besonderheit feststellen können: Im Bereich von $FC00 bis $FCFF ist entweder das RAM eingeschaltet oder das Betriebssystem-ROM, niemals aber ein anderes ROM. Das hat auch einen sinnvollen Grund: In diesem Bereich des Betriebssystem-ROM stehen wichtige Routinen, z. B. Modul-Reset, Modulstart und ein Teil der Interruptroutine. Diese Routinen werden immer wieder gebraucht, daher sind sie auch praktisch immer im verfügbaren Speicherbereich vorhanden. Das bedeutet aber auch, daß bei den an-

deren ROMs oder externen Speichern dieser Bereich $FC00-$FCFF unwirksam ist, er kann doch niemals erreicht werden. Das bis hierher gesagte ist schon recht kompliziert, deshalb folgt nun eine kurze Zusammenfassung:

Zwischen RAM und ROM kan mit einem Schreibzugriff auf die Adressen $FF3E und $FF3F umgeschaltet werden.

```
. STA $FF3E schaltet ROM ein
. STA $FF3F schaltet RAM ein
```

Zusätzlich kann zwischen 16 verschiedenen ROM-Konfigurationen laut Tabelle 21.1 umgeschaltet werden. Jede Einstellung wird als Modul bezeichnet. Die Module erhalten Nummern, die aus ihrer Einschaltadresse minus $FDD0 bestehen. Ein Schreibzugriff auf eine der Adressen von $FD00 bis $FD0F schaltet das gewünschte Modul ein. Zusätzlich sollte für die Interruptroutine in der Speicherzelle $FB die Modulnummer stehen.

21.3 Bankswitchingroutinen

Im Anhang H dieses Buches finden Sie die Belegung der Zero-Page und des Systemspeichers bis $0800. Dort finden Sie bei den Adressen $0494-$04DC und $05EC-$06EB die Bezeichnung »Bankswitching«. Sehen Sie sich doch diese Routinen einmal mit dem Monitor an. Geben Sie D 0494 ein. Auf dem Bildschirm erscheint:

```
. 0494 8D 9C 04 STA $049C
. 0497 78       SEI
. 0498 8D 3F FF STA $FF3F
. 049B B1 00    LDA ($00),Y
. 049D 8D 3E FF STA $FF3E
. 04A0 58       CLI
. 04A1 60       RTS
```

Hier verändert sich das Programm selbst, und zwar wird der Akkuinhalt in der Adresse $049C abgespeichert. Dadurch wird der indirekt indizierte Befehl in $049B beeinflußt. Diese Routine dient zum Lesen beliebiger Speicherzellen des RAM. In der Routine wird der Interrupt durch Setzen des I-Flags unterbunden. Das muß auch unbedingt geschehen, denn ein Interrupt während das RAM eingeschaltet ist, würde den Rechner zum Abstürzen bringen.

Wenn Sie diese Routine einmal starten und z. B. im Akku der Wert #$D0 und im Y-Register der Wert #$10 steht, wird der Akku mit dem Inhalt der RAM-Adresse geladen, deren Lowbyte sich aus der Summe des Inhalts von $00D0 und dem Y-Register ergibt. Das Highbyte steht in der Adresse $00D1 und wird um 1 erhöht, wenn bei der Addition von $00D0 und Y ein Übertrag aufgetreten ist.

Beispiel:

```
. 1000 A2 00     LDX #$00
. 1002 86 D0     STX $D0
. 1004 A2 A0     LDX #$A0
. 1006 86 D1     STX $D1
. 1004 A0 10     LDY #$10
. 1006 A9 D0     LDA #$D0
. 1008 20 94 04  JSR $0494
. 100B 00        BRK
```

Mit dieser Routine wird die RAM-Speicherzelle $A010 gelesen. Die Bankswitchingroutinen werden hauptsächlich vom Betriebssystem, vom BASIC-Interpreter und natürlich von der eingebauten Software genutzt. Die Routinen in $0494 bis $04E6 bewirken im Einzelnen:

Startadresse	Wirkung
$0494	Lesen einer RAM-Adresse. Dabei nachindizierte Adressierung mit Y-Register, Akku muß vorher Zero-Page-Adresse enthalten, in der das Lowbyte der Zieladresse steht.
$04A5	Lesen einer RAM-Adresse, deren Lowbyte in $003B und Highbyte in $003C steht. Dabei wird wieder die nachindizierte Adressierung mit dem Y-Register angewandt.
$04B0	Wie $04A5, aber Lowbyte in $0022, Highbyte in $0023.
$04BB	Wie $04A5, aber Lowbyte in $0024, Highbyte in $0025.
$04C6	Wie $04A5, aber Lowbyte in $006F, Highbyte in $0070.
$04D1	Wie $04A5, aber Lowbyte in $005F, Highbyte in $0060.
$04DC	Wie $04A5, aber Lowbyte in $0064, Highbyte in $0065.

Im Bereich von $05EC bis $06EB steht nach dem Einschalten des Plus/4 nur eine Routine. Diese beginnt in Adresse $05F5. Mit dieser Routine lassen sich Module ein-

schalten und an einer beliebigen Adresse starten. Wenn Sie diese Routine jetzt starten würden, würde das den Start der eingebauten Software zur Folge haben. Im Zero-Page-Listing steht bei den Adressen $05F0-$05F4 als Bezeichnung »Long-Jump«. Damit bezeichnet man beim Plus/4 Sprünge zwischen verschiedenen Moduln.

Sehen Sie sich die Routine ab $05F5 einmal mit dem Monitor an, Sie werden sicher gleich erkennen, was dort geschieht. Zunächst wird die Modulnummer in das X-Register geladen (hier #$05 für die eingebaute Software), dann wird die Startadresse bestimmt und in $05F0 und $05F1 abgespeichert. Anschließend wird die Routine in $FCFA gestartet, die das Modul (x) einschaltet und einen indirekten Sprung in die Adresse $05F0 durchführt.

Im Betriebssystem-ROM gibt es noch weitere Routinen, die für das Bankswitching genutzt werden können:

$FC7F — Schaltet Modul ein, dessen Nummer im X-Register steht. Die Nummer des alten Moduls muß im Akku stehen. Dann wird die Speicherzelle gelesen, deren Adresse in $00BE und $00BF steht. Dabei wird wieder die nachindizierte Adressierung mit dem Y-Register angewandt. Anschließend wird wieder das alte Modul eingeschaltet. So können also Werte aus Moduln ausgelesen werden, die zur Zeit nicht eingeschaltet sind.

$FC89 — Schaltet Modul ein, dessen Nummer im X-Register steht. Die Nummer des alten Moduls muß wieder im Akku stehen. Anschließend wird im Modul ein Programm gestartet, dessen Startadresse in $00F0 und $00F1 steht. Trifft die CPU in diesem Programm auf ein RTS ($60), wird das alte Modul wieder eingeschaltet, und es erfolgt ein Rücksprung in das Programm, das $FC89 aufgerufen hat.

$FCB3 — Routine mit der Bezeichnung »PULS«. Diese Routine wird bei jedem Interrupt aufgerufen. Sie schaltet das Modul (0) ein (BASIC- und Betriebssystem-ROM) und startet die Interruptroutine. PULS kann sehr gut für eigene Softwaremoduln benutzt werden, die am Expansionsport angeschlossen sind. Nach der Interruptroutine wird wieder das vorherige Modul aufgerufen.

$FCC9 — Einschaltroutine für Modul, dessen Nummer in $FB steht. Es wird zunächst das Modul eingeschaltet und anschließend ein indirekter Sprung in $02FE gemacht. Lowbyte der Startadresse in $02FE, Highbyte in $02FF.

21.4 Modul-Reset

Beim Einschalten des Plus/4 und beim Drücken der RESET-Taste wird der besagte Modul-Reset durchgeführt. Wenn Sie einmal auf die Idee kommen sollten, eigene EPROMs zu brennen und diese am Expansionsport des Plus/4 anzuschließen, ist es wichtig, daß Sie wissen, was in der Reset-Routine passiert.

Hierdurch ist es auch möglich, die Moduln automatisch zu starten. Wenn Sie also Ihren Plus/4 für bestimmte Aufgaben einsetzen möchten und das entsprechende Programm in einem EPROM haben, ist es mitunter sehr nützlich, wenn dieses Programm nach dem Einschalten des Rechners sofort gestartet wird.

Die Modul-Reset-Routine befindet sich im Betriebssystem-ROM ab Adresse $FC1E. Dort passiert folgendes:

Nacheinander werden die Module 15, 10, 5 und 0 eingeschaltet. Dann wird im gerade aktiven Modul in den Adressen $8007 bis $8009 nach der Kennung »CBM« gesucht. Wenn diese Kennung vorhanden ist, wird aus dem Modul der Inhalt der Adresse $8006 gelesen. Der gelesene Wert ist die Modulkennzahl. Das BASIC-ROM hat die Kennzahl 0 und die eingebaute Software die Kennzahl $0C.

Diese Kennzahl wird in einer Tabelle ($05EC bis $05EF) abgespeichert. Wenn nun diese Kennzahl den Wert 1 hat, wird das Modul gestartet, und somit kann also ein automatischer Modulstart, ein »Auto-Start«, erfolgen.

Wir zeigen Ihnen nun einen Hexdump eines Auto-Start-Moduls.

```
>8000 4C 0A 80 00 00 00 01 43
>8008 42 4D ...
```

In $8000 steht der Sprungbefehl in das eigentliche Modulprogramm, in $8006 steht der Wert 1 für den Auto-Start und in $8007-$8009 stehen die ASCII-Werte für »CBM«. In $800A beginnt das Programm.

21.5 Anschluß eines EPROMs am Expansionsport

Auf dem folgenden Bild 21.2 zeigen wir Ihnen, wie Sie ein EPROM am Expansionsport anschließen können.

In unserem Beispiel belegt das EPROM die Adressen ab $8000. Ein EPROM vom Typ 23128 oder 27128 belegt dann die Adressen $8000 bis $BFFF. Sie können aber auch ein 2764 anschließen. Bei diesem EPROM-Typ wird die Adreßleitung 13 nicht

benötigt, da das EPROM nur 8 Kbyte belegt. Sie sollten dann den Pin 26 des EPROMs an +5 Volt legen. Das 2764 belegt somit die Adressen $8000 bis $9FFF.

Natürlich können Sie den Adreßbereich des EPROMs auch ab $C000 festlegen. Dazu müßte der Pin 20 des EPROMs an -C1 High (Expansionsport Stift 6) und der Pin 22 des EPROMs müßte an -CS1 (Expansionsport Stift 9) angeschlossen werden. Einfacher ist es, wenn Sie Pin 22 der EPROMs grundsätzlich an Masse anschließen, die Schaltung funktioniert dann auch einwandfrei. Wenn Sie den gesamten Adreßbereich von $8000 bis $FFFF nutzen möchten, müssen Sie zwei EPROMs vom Typ 27128 oder 23128 anschließen. Pin 20 des ersten EPROMs wird an -C1 Low und Pin 20 des zweiten EPROMs an -C1 High angeschlossen. Statt -C1 Low und -C1 High können Sie auch -C2 Low und -C2 High benutzen. Selbstverständlich können Sie auch vier EPROMs anschließen indem Sie alle vier Select-Leitungen verwenden. Wie Sie diese EPROMs in den Adreßbereich des Rechners einblenden, und wie die darin enthaltenen Programme gestartet werden, entnehmen Sie bitte der Tabelle 21.1 und den beschriebenen Beispielen.

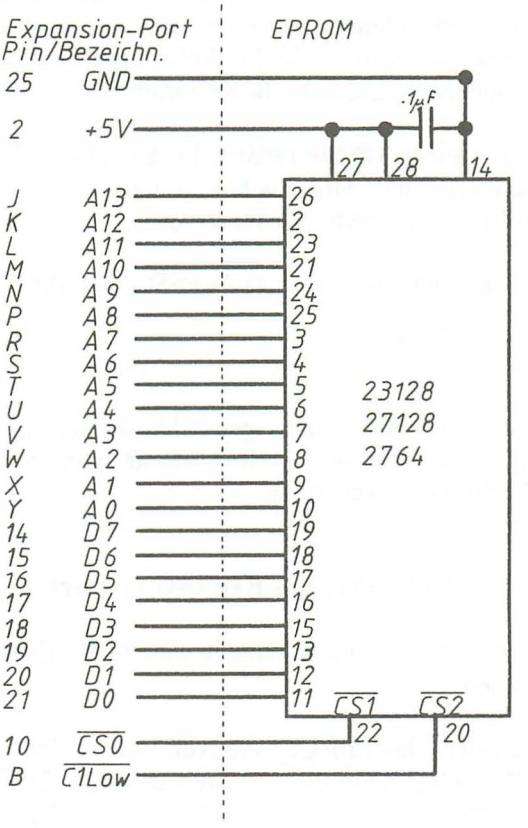


Bild 21.2: *EPROM am Expansionsport*

21.6 Das Erzeugen von DATA-Zeilen

So, nun haben Sie in den vorangegangenen Kapiteln Ihren Plus/4 schon sehr viel genauer kennenlernen können. Sicher hat der eine oder andere schon kurze Maschinensprachprogramme geschrieben. Wenn diese Routinen in ein bestehendes BASIC-Programm eingebunden werden sollen, trifft man sehr leicht auf Schwierigkeiten. Entweder muß man diese Routinen nachladen, dies ist aber da umständlich, oder es müssen mit Hilfe des Maschinensprachmonitors DATA-Zeilen erstellt werden. Diese Methode ist aber sehr zeitraubend. In diesem Teil des Kapitels soll Ihnen eine wesentlich schnellere und komfortablere Art vorgestellt werden. Wir zeigen Ihnen die Programmierung eines DATA-Zeilenerzeugers.

Die Beschreibung dieses Programms ist sehr ausführlich, damit Ihnen der Einstieg in die Maschinensprache noch etwas leichter fällt. Das Programm wurde für das einfachere Eingeben als BASIC-Ladeprogramm und für die Erklärung als Assemblerlisting abgedruckt.

Kommen wir zur Problemstellung:

Wir wollen ein Programm in Maschinensprache schreiben, mit dessen Hilfe es möglich sein soll im Speicher befindliche Maschinensprachprogramme in DATA-Zeilen darzustellen. Die erzeugten Datas sollen später mit einem BASIC-Lader wieder in den Speicher gepoked werden können. Schon jetzt werden Sie das erste Problem erkannt haben. Um eine BASIC-Zeile per Programm zu erzeugen, muß man erst einmal wissen wie eine solche Zeile aufgebaut ist. Dies wollen wir Ihnen im einzelnen erklären.

21.6.1 Der Aufbau einer BASIC-Programmzeile

Führen Sie bitte vor dem Weiterlesen einen RESET aus. Jetzt geben Sie ein:

```
10 PRINT "Plus/4"
```

Schalten Sie den Maschinensprachmonitor ein und geben ihm den Befehl:

M 1000 1010

Das Ergebnis:

```
>1000 00 0F 10 0A 00 99 20 22
>1008 50 4C 55 53 34 22 00 00
>1010 00 00
```

Die dritte Zeile wurde nicht bis zum Schluß wiedergeben, da der Rest individuell verschieden ausfallen kann.

Betrachten wir das Ergebnis etwas näher. Der erste Wert ist null. Diese Null ist immer vorhanden, denn sie zeigt dem BASIC-Interpreter, daß hier der Programmspeicher beginnt. Als nächste Werte sehen wir 0F und 10. Bei diesen beiden Werten handelt es sich um eine Adresse. Da beim 6502 Adressen im Format Low Byte, High Byte abgelegt werden, müssen wir beide Werte nur vertauschen, und erhalten so die uns verständliche Adresse 100F. Diese Adresse nennt man Koppeladresse. Sie sagt dem Interpreter, an welcher Stelle des Speichers die nächste Programmzeile beginnt.

Die folgenden beiden Byte enthalten die Werte 0A und 00. Dies ist keine neue Adresse, sondern repräsentiert unsere Zeilennummer, wiederum im Low Byte, High Byte Verfahren abgespeichert. Drehen wir auch diese Zahlen um. Das Ergebnis ist 000A. Wenn wir diesen Wert in das Dezimalsystem umrechnen, erhalten wir die Zahl 10, den Wert unserer Zeilennummer.

Das nächste Byte hat den Wert 99. Rechnen wir auch diese Zahl ins Dezimalsystem um, so erhalten wir 153. Sehen Sie einmal in der Tabelle über die BASIC-Token im Anhang nach. Unter der Zahl 153 finden Sie PRINT. Der Interpreter wandelt also beim Übernehmen einer Programmzeile jedes Befehlswort in eine Zahl um und speichert auch nur diese an der entsprechenden Stelle im Speicher ab. Es ist leicht einzusehen, daß diese Methode eine Menge Speicherplatz spart. Für PRINT wären sonst 5 Byte erforderlich, so wird der Platz auf ein Byte beschränkt. Nun aber zum nächsten Byte.

Dieses hat den Wert 20. Wir hoffen auch bei Ihnen, sonst haben Sie beim Eingeben der Programmzeile das SPACE zwischen PRINT und den Anführungszeichen vergessen. Dieser Fehler hat sich dann aber auch schon eher bemerkbar gemacht, denn Ihre Koppeladresse müßte 100E lauten. Berichtigen Sie diesen Fehler bitte gleich, damit es Ihnen leichter fällt unseren Erklärungen zu folgen. Zurück zur Hex-Zahl 20. Umgerechnet ins Dezimalsystem ergibt sie 32. Diese Zahl repräsentiert den CHR$-Code für ein Space.

Der Rest ist ziemlich einfach: 22 steht für die Anführungszeichen, 50 für das P, 4C für das L, 55 für das U, 53 für das S und 34 schließlich für die 4. Das nächste Byte enthält wieder den CHR$-Code für die Anführungszeichen. Wir wenden uns den letzten drei Byte zu. Alle enthalten den Wert 00. Wenn der Interpreter auf ein Byte mit diesem Wert trifft, so erkennt er an ihm, daß die gerade bearbeitete Programmzeile zu Ende ist. Er erinnert sich nun der Koppeladresse und sieht in ihr nach, wo das Programm weitergeht. Unsere Koppeladresse zeigt auf 100F. Dieses Byte und das nächste müßten nun die Koppeladresse für die nächste Koppeladresse enthalten. Tun sie in unserem Fall aber nicht. Bei uns haben beide ebenfalls den Wert null. An diesen bei-

den Nullen erkennt der Interpreter das Programmende, und meldet sich mit READY in den Direktmodus zurück. Um das eben Beschriebene zu untermauern, verlassen wir den Monitor und geben ein:

```
20 PRINT
```

Jetzt wieder zurück in den Monitor und eingegeben:

M 1000 1010

Die Zeilen sehen jetzt so aus:

```
>1000 00 0F 10 0A 00 99 20 22
>1008 50 4C 55 53 34 22 00 15
>1010 10 14 00 90 00 00 00
```

Auf die Darstellung des letzten Byte wurde aus den oben angeführten Gründen verzichtet. Geändert hat sich nur in der zweiten Zeile das letzte Byte und die dritte Zeile. Basteln wir uns doch einmal die nächste Koppeladresse. Siehe letztes Byte der zweiten Zeile und erstes Byte der dritten Zeile. Die Adresse lautet: 1015. An dieser Adresse stehen wiederum zwei Byte mit dem Inhalt 0, und sagen somit dem Interpreter: »Du darfst aufhören, das BASIC-Programm ist zu Ende«.

Fassen wir zusammen:

Jede Programmzeile benötigt fünf Verwaltungsbyte. Byte eins und zwei einer Programmzeile enthalten die Koppeladresse. Sie gibt an, an welcher Stelle des Programmspeichers die nächste abzuarbeitende Zeile steht. Sind beide Byte null, so heißt dies Programmende. Die nächsten beiden Byte (Byte drei und vier) enthalten die Zeilennummer der gerade zu bearbeitenden Zeile. Das letzte Byte der Zeile ist stets null. Sie zeigt dem Interpreter das Ende der Programmzeile an.

Betrachtet man diese Zusammenfassung, so wird einem sehr schnell klar, daß man, werden mehrere Programmzeilen zusammengefaßt, einiges an Speicherplatz sparen kann. Bei unserem Plus/4 wird so etwas, Dank des großen Speicherplatzes, wohl kaum nötig sein.

Als nächstes wollen wir einmal eine eigene DATA-Zeile mit Koppeladresse, BASIC-Token und allem Drum und Dran, mit Hilfe des Monitors, eingeben. Verlassen Sie den Monitor und führen Sie NEW aus. Jetzt wird wieder der Monitor eingeschaltet und dann erfolgt:

M 1001 1010

Die erstellte Programmzeile soll später zwei DATA-Elemente enthalten, und die Zeilennummer soll 256 sein.

Die ersten beiden Byte lassen wir im Moment noch außer Betracht. Rechnen wir die Zeilennummer ins HEX-Zahlensystem um, so ergibt sich daraus $0100. Byte drei und vier werden also in 00 01 geändert (bedenken Sie Low Byte, High Byte Anordnung). Das BASIC-Token DATA hat den Wert 131, das ergibt $83 und ist somit Byte fünf. Ändern Sie die Byte sechs, sieben und acht in 30 2C 30. Dies sind die HEX-Werte der CHR$-Codes Null, Komma und noch einmal Null. Die Zeile ist jetzt fertig eingegeben, es folgt also einmal 00 für das Zeilenende, und noch zweimal 00, um dem Interpreter auch das Programmende anzuzeigen.

Kommen wir nun zur Koppeladresse, die ja bekanntlich in den ersten beiden Byte einer Zeile stehen muß. Die Adresse der vorletzten Null unserer Zeile ist 100A. Die ersten Byte werden in 0A 10 geändert. Das Endergebnis sieht so aus:

```
>1001 0A 10 00 01 83 30 2C 30
>1009 00 00 00
```

Verlassen Sie bitte den Monitor und geben dann LIST ein. Das Ergebnis:

```
256 DATA0,0
```

Was Sie da sehen, ist zwar recht schön, aber so einfach ist es denn doch nicht. Geben Sie ein:

```
257 PRINT
```

Nach RETURN erfolgt der »Absturz« des Systems. Dies liegt an einigen Zeigern, die nach dem Eingeben der Zeile nicht berichtigt wurden. Was ein Zeiger ist und was er bewirkt, erfahren Sie im nächsten Abschnitt. Erwecken Sie jetzt erst einmal Ihren Plus/4 mit einem Reset wieder zum Leben.

So, nun können wir uns schon denken, wie eine durch das Programm erstellte BASIC-Zeile aussehen muß. Bis dahin, meine Freunde der Bits und Bytes und RAMs und ROMs, ist wahrlich noch ein weiter Weg. Wer bis jetzt noch nicht genug hat, und wir hoffen, daß es unter Ihnen keinen gibt, der folge uns ins nächste Unterkapitel.

21.6.2 Der Aufbau des BASIC-Programmspeichers

Im letzten Abschnitt haben wir eine kleine Schlappe erlitten, die es auszumerzen gilt. Das Mißgeschick passierte nur, weil uns noch einige Grundkenntnisse über die Speicherverwaltung fehlten. Dies soll nun nachgeholt werden.

Im BASIC-Programmspeicher wird nicht nur das Programm abgelegt, sondern auch ebenfalls alle Variablen, Variablenfelder, Strings und Stringvariablenfelder. Der Speicher wird hierbei dynamisch verwaltet, er wird immer an die gegebenen Umstände angepaßt. Machen Sie doch bitte nach NEW im Direktmodus folgendes:

PRINT FRE(X)

Das Ergebnis:

60669

Sie haben also 60669 Byte für ein BASIC-Programm zur Verfügung. Jetzt folgt:

DIM A(1000)
PRINT FRE(X)

Der noch freie Speicherbereich ist auf 55657 Byte geschrumpft. Der BASIC-Interpreter, der auch gleichzeitig für die Speicherverwaltung zuständig ist, hat einen Zeiger verändert und somit den Platz für ein 1000 Elemente umfassendes Feld reserviert. Diesen reservierten Platz benötigen wir aber nicht und geben daher:

CLR

ein. Nun steht uns der ursprüngliche Platz wieder zur Verfügung. Das Ganze sehen wir uns auch mit dem Maschinensprachmonitor an.

M 2B 3A ergibt

```
>002B 01 10 03 10 03 10 03 10
>0033 00 FD FE FC 00 FD 00 FF
```

Die Fülle der Zahlen repräsentiert eine Anzahl von Zeigern, die genau unter die Lupe genommen werden.

Die Zeiger setzen sich aus jeweils zwei Byte zusammen und zeigen, wie der Name schon sagt, auf etwas. Die ersten beiden Speicherstellen enthalten die Werte 01 10. Auch die Zeiger sind wieder im Low Byte, High Byte Verfahren abgespeichert. Drehen wir die Zahl um, ergibt das 1001. Dies hat aber nichts mit dem bekannten Mär-

chenbuch zu tun, sondern sagt uns, und natürlich auch dem Interpreter, an dieser Speicherstelle beginnt der BASIC-Speicher. Eigentlich beginnt der Speicher schon bei $1000, aber das erste Byte des Speichers muß ja stets Null sein. Aus diesem Grund zeigt der Zeiger auf $1001, dem Low Byte der ersten Koppeladresse.

Die nächsten beiden Werte ergeben, nach oben genanntem Verfahren in die richtige Reihenfolge gebracht, 1003. Warum dieser verzwickte Wert, werden sich einige unter Ihnen fragen. Es ist gar nicht so schwer. Der Programmspeicher beginnt bei 1001. Wir haben im letzten Abschnitt festgestellt, daß dort eine zwei Byte lange Koppeladresse steht. 1001 plus 0002 ergibt 1003. Dieser Zeiger weist auf den Anfang der Variablen hin. Nach der Eingabe einer Programmzeile wird er regelmäßig vom Interpreter berichtigt, und zeigt so immer auf das erste der Koppeladresse folgende Byte. Stellt man sich den Speicher als Block vor, so kann man sagen, daß dieser Zeiger von unten nach oben wandert.

Auch der Zeiger, der von den nächsten beiden Byte repräsentiert wird, hat den Wert 1003 und wandert ebenfalls nach oben. Er zeigt auf den Beginn der Felder und stellt gleichzeitig das Ende des Variablenspeichers dar.

Die nun folgenden Zeiger betrachtet man am besten gemeinsam. Der erste lautet auch wieder 1003, der zweite FD00. Der erste Wert deutet auf das Ende des Speichers für die Felder hin, der zweite auf das Ende des Stringspeichers. Wenn der Zeiger für den Anfang der Felder nach oben wandert, dann ist es nur logisch, daß der Zeiger für deren Ende das Gleiche tut. Wir haben noch kein Feld dimensioniert, also ist der Anfang auch gleich dem Ende dieses Speicherbereichs.

Was ist nun aber mit dem anderen Zeiger? Nun, der wandert nach unten. Wenn beide Zeiger, also der für das Ende der Felder und der für das Ende der Stringtexte, einmal denselben Wert enthalten, so erfolgt die Ausgabe der berühmten Fehlermeldung: OUT OF MEMORY ERROR.

Mit diesem Wissen ausgestattet, werden Sie sicher ebenfalls die Differenz zwischen der Einschaltmeldung, 60671 BYTES FREE, und einem sofort ausgeführten PRINT FRE(X), 60669, erkennen. Diese falsch anmutende Meldung kommt dadurch zustande, weil der Beginn der Variablen zwei Byte höher liegt als der Anfang des Programmspeichers.

Als nächstes müssen wir nur noch herausfinden, wo denn nun der Speicherbereich für die Strings beginnt. Dazu überschlagen wir einfach die nächsten zwei Byte und wenden uns den ihnen folgenden zu. Der Inhalt heißt 00 FD. Der Zeiger zeigt also auf FD00. Diese Adresse ist einerseits das Ende des BASIC-Speichers und andererseits der Beginn des Stringspeichers. Dieser Zeiger ist starr, das heißt, er wird vom Interpreter nicht verändert.

Zum besseren Verständnis haben wir das oben Gesagte einmal gezeichnet.

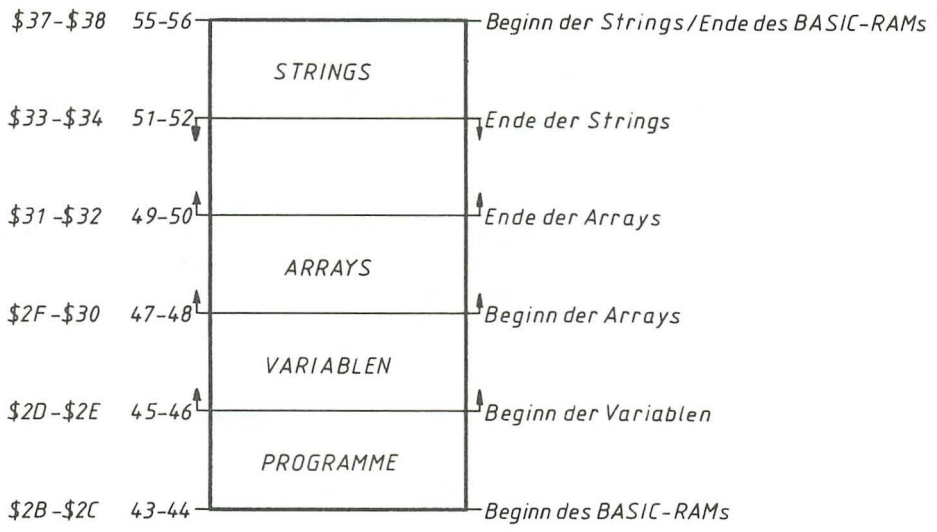


Bild 21.2: *Der BASIC-Programmspeicher*

Na, ahnen Sie schon warum uns vorhin dieses Mißgeschick passierte? Wir haben vergessen den Zeiger auf das Ende des BASIC-Speichers zu setzen und somit den Beginn der Variablen zu berichtigen. Strenggenommen müßten wir auch die Zeiger auf den Beginn und das Ende der Felder berichtigen, aber diese Arbeit lassen wir uns vom Computer abnehmen, etwas soll er ja auch mal wieder zu tun bekommen. Zuerst müssen wir aber noch einmal unsere Programmzeile über den Monitor eingeben.

```
>1001 0A 10 00 01 83 30 2C 30
>1009 00 00 00
```

Nun wird der Variablenanfang heraufgesetzt.

```
>002D 0B 10
```

Verlassen Sie nun den Monitor und geben dann LIST ein. Es hat sich nichts geändert, alles sieht O.K aus. Ist es aber noch nicht ganz. Die anderen Zeiger müssen erst mit dem BASIC-Befehl CLR auf ihre richtigen Werte gebracht werden. Also:

CLR

Nun endlich können Sie beruhigt eine weitere Zeile eingeben.

```
257 PRINT "PLUS/4"
```

Sie können das Programm jetzt auch starten und sich überzeugen, es ist geschafft! Sie hätten statt CLR auch RUN nehmen können, da dieser Befehl ebenfalls die übrigen Zeiger berichtigt.

Diese sehr langwierige Prozedur soll Ihnen später unser Maschinenprogramm abnehmen. Da das Programm aber etwas länger ist, paßt es leider nicht mehr in den freien Bereich ab $065E oder dezimal 1630. Wie man im BASIC-Programmspeicher Platz für ein Maschinenprogramm schafft, ohne daß dieses vom BASIC überschrieben werden kann, davon handelt der nächste Abschnitt.

21.6.3 Speicherplatzreservierung

Die Zeigerwirtschaft des Plus/4 hilft nicht nur dem Interpreter, sondern auch dem Programmierer.

Wie Sie bisher erfahren haben, stehen die Zeiger auf dem Anfang und auf dem Ende des BASIC-Speichers »fest«. Das heißt eigentlich nichts anderes, als daß sie vom Interpreter nicht verändert werden. Bis auf eine Ausnahme. Wenn Sie die Grafik Ihres Plus/4 einschalten, wird der Zeiger auf den Beginn des Programmspeichers nach oben bewegt und dann alle übrigen berichtigt. Schalten Sie die Grafik wieder aus, so bewegt sich der Zeiger wieder abwärts, d.h. er zeigt auf dieselbe Speicherstelle wie vor dem Grafikaufruf. Die anderen Zeiger werden ebenfalls dem aktuellen Stand angepaßt.

Schalten wir wieder einmal den Monitor ein.

M 2B 32

```
>002B 01 10 03 10 03 10 03 10
```

Jetzt wollen wir versuchen einen Speicherbereich mit der Länge von 256 Byte zu reservieren. Es ist folgendes zu tun:

```
>002B 01 11 03 11
>2000 00
```

Verlassen Sie den Monitor und geben Sie nun CLR ein. Es folgt: PRINT FRE (X). Die aktuelle Speichergröße beträgt nur noch 60413 Byte. Es fehlen also scheinbar 256 Byte. Diese Speicherstellen wurden aber lediglich reserviert und lassen sich von BASIC nur noch mit PEEK oder POKE abfragen oder ändern. Nun noch schnell die Erklärung:

Geändert haben wir lediglich den BASIC-Start, indem der Zeiger in $2B und $2C nach oben verändert wurde. Er zeigt jetzt auf 1101, dem neuen Beginn des BASIC-Speichers. Wie auch schon vorhin, mußte danach auch der Zeiger auf den Variablenanfang noch geändert werden. Der dann folgende Schritt ist ebenfalls leicht zu erklären. Wir wissen ja bereits, daß das erste Byte des Programmspeichers stets null sein muß. Dies wurde mit dem nächsten Schritt vollzogen. Um auch die übrigen Zeiger zu berichtigen war dann noch CLR erforderlich, und der Speicher war endgültig verkleinert. Diese Speichergröße bleibt jetzt bis zu einem Reset, nicht einem RUN/STOP-Reset, erhalten.

In den freien Speicherplatz könnten nun Maschinenprogramme oder irgendwelche Werte abgelegt werden. Schalten Sie die Grafik ein, so wird der Bereich von $1800 bis $1FFF von ihr belegt, und der BASIC-Speicher beginnt nun bei $2000 (Zeiger auf $2001). Soll also auch mit der Grafik gearbeitet werden, so kann bei dieser Methode nur der Bereich von $1000 bis 17FF für derartige Anwendungen genutzt werden.

Einen weiteren Nachteil hat das Höherlegen des BASIC-Programmspeichers. Haben Sie bereits ein BASIC-Programm eingegeben und soll noch ein Maschinenprogramm in den unteren Bereich nachgeladen werden, so wird der Anfang des im Speicher stehenden Programms von diesem überschrieben. Der Rechner wird auch höchstwahrscheinlich »abstürzen«. Dieser Bereich muß also immer sofort nach dem Einschalten oder Reset durch »verbiegen« der Zeiger reserviert werden.

Eine bessere Methode ist das Heruntersetzen des BASIC-Endes, und des Zeigers auf die Stringtexte. Auch dies soll einmal demonstriert werden. Nach einem Reset und Einschalten des Monitors folgt:

```
>0033 00 FC
>0037 00 FC
```

Nach der Rückkehr ins BASIC und PRINT FRE (X) können Sie feststellen, daß wir das gleiche Ergebnis erhalten. Wir mußten auch kein CLR eingeben, keine Null setzen und haben somit den Aufwand reduziert. Dies alles wurde durch das Setzen der Zeiger für den Stringanfang und das BASIC-Ende erreicht. Die in diesem Kapitel verwendeten Speicherstellen können Sie auch im Anhang G nachlesen. Vorteil dieser Methode der Speicherplatzreservierung ist, daß wir einen sehr viel größeren Bereich vor BASIC-Zugriff schützen können, und daß man auch Maschinenprogramme vom BASIC aus fast problemlos nachladen kann.

So, nun haben Sie schon einige und vor allem wichtige Grundlagen zu unserem, immer noch geplanten, DATA-Erzeuger erhalten. Da der Plus/4 aber schon seit einiger Zeit nichts mehr zu tun hat, werden wir im nächsten Abschnitt mit dem Programmieren beginnen.

21.6.4 Das Ladeprogramm zum DATA-Erzeuger

Unser Programm soll im oberen Speicherbereich ab $FB00 stehen. Da es aber auch Routinen des Betriebssystems nutzt, wurde es erforderlich eigene Bankswitchingroutinen zu schreiben, die ab $065E zu finden sind. Um den Aufwand beim Einladen des Programms zu verringern, steht in diesem Bereich auch gleichzeitig eine Laderoutine für den zweiten Teil des Programms, den eigentlichen DATA-Erzeuger. Sie brauchen also später nur das nachfolgende Ladeprogramm einzuladen und dieses zu starten. Der zweite Teil wird dann ohne weiteren Aufwand nachgeladen.

Zum Eingeben des Laders:

Sie finden das Ladeprogramm sowohl als BASIC-Lader als auch als Assemblerlisting vor. Die bequemste Art des Programmierens ist, den BASIC-Lader zu verwenden. Nach dem Eingeben sollten Sie das Programm vor dem ersten Start sicherheitshalber abspeichern. Meldet sich das Programm später mit: FEHLER IN DATAS, so stimmt die Prüfsumme nicht mit der Summe der DATAs überein. Sie müssen Ihr Programm dann noch einmal überprüfen und berichtigen, dann sollte es vor einem weiteren Versuch wiederum abgespeichert werden.

Sind Sie glücklicher Besitzer einer Diskstation, dann ändern Sie bitte die Zeile 110 so:

```
110 DATA 85,38,85,34,A9,08,85,AE
```

Besitzer eines Kassettenrecorders übernehmen bitte das Listing ohne Änderung.

Der BASIC-Lader:

```
10 FOR T=1630 TO 1744
20 READ A$: B=B+DEC(A$)
30 POKE T,DEC(A$)
40 NEXT
50 IF B <> 12585 THEN PRINT "FEHLER IN DATAS"
100 DATA A9,00,85,37,85,33,A9,FB
110 DATA 85,38,85,34,A9,01,85,AE
120 DATA A9,01,85,AD,A0,0D,84,AB
130 DATA 88,A9,5E,85,AF,A9,02,85
140 DATA B0,B9,C4,06,91,AF,88,10
150 DATA F8,A9,00,85,9A,20,D5,FF
160 DATA 90,19,60,78,8D,3F,FF,20
```

```
170 DATA 00,FB,8D,3E,FF,58,60,8D
180 DATA 3E,FF,58,20,00,00,78,8D
190 DATA 3F,FF,60,A0,17,B9,91,06
200 DATA 99,5E,06,88,10,F7,60,A9
210 DATA 01,A0,00,91,2B,20,18,88
220 DATA 20,4B,88,4C,9A,8A,44,41
230 DATA 54,41,47,45,4E,45,52,41
240 DATA 54,4F,52
```

Nach dem ersten einwandfreien Start gehen Sie nun bitte wieder in den Monitor und geben ein:

```
S"LADER",1,065E,06D1
```

Mit dieser Zeile erfolgt die Abspeicherung auf einem angeschlossenen Kassettenrecorder. Besitzer einer Floppy ändern zum Abspeichern einfach die Eins in eine Acht um. Geladen wird dann später im Direktmodus mit:

```
LOAD "LADER",1,1
```

oder bei der Floppy mit

```
LOAD "LADER",8,1
```

Wir kommen nun zum Assemblerlisting, mit dessen Hilfe erläutert wird, warum was programmiert wurde. Dieses Listing wird routinenweise erklärt. In den Erklärungen finden Sie auch wichtige Tips zur Benutzung einiger Betriebssystemroutinen. Wer sich hiermit noch nicht befassen möchte, kann diesen Teil auch überspringen und mit der Eingabe des nächsten Programmteils fortfahren.

Den meisten Zeilen folgt ein durch ein Semikolon abgetrennter Kommentar. Er soll Ihnen auch später noch das schnelle Verstehen dieses Programms ermöglichen. Sollten Sie das Programm mit Hilfe des Monitors verfolgen, so finden Sie ihn selbstverständlich nicht vor.

Das Assemblerlisting des Ladeprogramms

```
;***** Ende des BASIC-Speichers auf $FB00 heruntersetzen *****

. 065E     A9 00     LDA #$00     ; AKKU mit dem Wert 0 laden
. 0660     85 37     STA $37      ; AKKU speichern in $37
. 0662     85 33     STA $33      ; und in $33
. 0664     A9 FB     LDA #$FB     ; AKKU mit dem Wert $FB laden
. 0666     85 38     STA $38      ; AKKU speichern in $38
. 0668     85 34     STA $34      ; und in $34
```

Wie schon erwähnt, soll unser Hauptprogramm ab Adresse $FB00 im Speicher stehen. Wir müssen also, um ein Überschreiben durch BASIC zu verhindern, das BASIC-Ende heruntersetzen. Dazu wird der AKKU unmittelbar mit dem Wert $00 geladen, und dann sein Inhalt in den Adressen $37 (Low Byte des Zeigers auf BASIC-Ende) und $33 (Low Byte des Zeigers auf Beginn der Strings) abgespeichert. Nun erfolgt das Laden des AKKUs mit dem High Byte. Der Wert wird in den Adressen $38 (High Byte des Zeigers auf BASIC-Ende) und $34 (High Byte des Zeigers auf Beginn der Strings) abgelegt.

```
; ****** Laderoutine für das Hauptprogramm **********

. 066A   A9 01     LDA #$01     ; Geräteadresse laden
. 066C   85 AE     STA $AE      ; in $AE speichern
. 066E   A9 01     LDA #$01     ; Sekundäradresse laden
. 0670   85 AD     STA $AD      ; in $AD speichern
. 0672   A0 0D     LDY $0D      ; Länge des Filenamens
. 0674   84 AB     STY $AB      ; in $AB speichern
. 0676   88        DEY          ; Y-Reg. erniedrigen
. 0677   A9 5D     LDA #$5E     ; Adresse des Filenamens
. 0679   85 AF     STA $AF      ; Low Byte nach $AF
. 067B   A9 02     LDA #$02     ; High Byte der Adresse
. 067D   85 B0     STA $B0      ; nach $B0
. 067F   B9 C4 06  LDA $06C4,Y  ; Filenamen an die in $AF
. 0682   91 AF     STA ($AF),Y  ; und $B0 angegebene
. 0684   88        DEY          ; Adresse speichern
. 0685   10 F8     BPL $067F    ; nicht < 0? dann wieder $067F
. 0687   A9 00     LDA #$00     ; AKKU mit $00 laden
. 0689   85 9A     STA $9A      ; keine Meldungen beim Laden
. 068B   20 D5 FF  JSR $F  FD5  ; Aufruf von LOAD
. 068E   90 19     BCC $06A9    ; zur Verschieberoutine ($06A9)
. 0690   60        RTS          ; bei Ladefehler Ende
```

Diese Routine bereitet erst das Laden des Hauptprogramms vor und springt dann in die Laderoutine des Betriebssystems. Die erforderlichen Schritte im einzelnen:

Um ein Programm zu laden, benötigt der Computer die Angabe des Namens, der Gerätenummer und der Sekundäradresse. Einige dieser Angaben müssen nicht unbedingt eingegeben werden. Der Rechner setzt dann Werte voraus. Unser Hauptprogramm soll vom Kassettenrecorder geladen werden. Die Gerätenummer von ihm ist eins und wird vom Computer in der Adresse $AF erwartet. Besitzer einer Floppy müssen die Eins in eine Acht ändern. Da es sich bei dem zu ladenden Programm um ein Maschinenprogramm handelt, muß es absolut geladen werden. Die Sekundäradresse beträgt also ebenfalls eins. Die Sekundäradresse wird in der Speicherstelle $AD an das Betriebssystem übergeben.

Unser Ladeprogramm soll ja nur ein bestimmtes Programm laden. Wir benötigen darum einen FILE-Namen. Zuerst übergeben wir die Länge des Namens in der Spei-

cherstelle $AB. Als nächstes muß das Betriebssystem wissen, wo denn der Programmname zu finden ist. Unser Plus/4 besitzt auch hierfür einen Zeiger. Wir finden ihn in den Adressen $AF und $B0. Dieser wird auf den Speicherbereich $025E gesetzt.

Nachdem diese Parameter übergeben sind, beginnt eine kurze Speicherverschieberoutine. Sie steht im Lader ab Adresse $067F und endet bei $0685. Mit ihr wird der FILE-Namen, vorhanden in den Adressen $06C4 bis $06D0, in den Bereich ab $025E übertragen. Man hätte den Zeiger auf den FILE-Namen auch auf $06C4 »verbiegen« können, dies hätte aber kaum Speicherplatz und Zeit eingespart. Der Bereich ab $065E sollte möglichst für eigene, kurze Routinen freigehalten werden.

Das Y-Register enthält nach der Dekrementierung (siehe $0676) den Wert $0C. Der AKKU wird nun, absolut und mit dem Y-Register nachindiziert, geladen. Oh, welch ein Kauderwelsch! Es folgt die Übersetzung: Zu der absoluten Adresse ($06C4) wird das Y-Register ($0C) addiert. Der AKKU wird also mit dem Inhalt der Speicherstelle $06D0 geladen. Die Speicherung des Inhalts erfolgt indirekt, Y nachindiziert. In unserem Fall wird die Speicheradresse wie folgt berechnet: Hinter STA steht ($AF),Y. $AF ist ein Zeiger. Dieser Zeiger besteht aus Low und High Byte und hat den Inhalt $025E. Zu diesem Zeiger wird der Inhalt des Y-Registers ($0C) addiert. Der AKKU-Inhalt wird dadurch in der Speicherstelle $026A abgelegt.

Da die Verschiebung von oben nach unten erfolgt, wird das Y-Register dekrementiert, sprich heruntergezählt. Der Inhalt beträgt jetzt $0B. Der nächste Befehl heißt BPL $067F. Übersetzt: verzweige, wenn die letzte Operation ein positives Ergebnis hatte. Positiv ist hier mathematisch zu sehen (+). Wie Sie vielleicht schon wissen, spiegelt das N-Flag des Statusregisters das siebte Bit der Zahl wieder, mit der der Prozessor gerade operiert oder operiert hat. Ist dieses Flag gesetzt, so bedeutet dies für ihn, daß die Operation ein negatives (-) Ergebnis hatte. Das Y-Register erhielt durch die letzte Operation den Wert $0B. Bit sieben ist darum nicht gesetzt. Das Ergebnis ist also positiv (+), die Bedingung (verzweige wenn positiv) erfüllt, und es erfolgt der Sprung nach $067F.

In $067F wird jetzt der AKKU mit dem Inhalt von $06CF ($06C4+$0B) geladen und an der Adresse $0269 ($025E+$0B) abgespeichert. Das Y-Register wird wieder dekrementiert. Der sich hieraus ergebende Wert ($0A) ist für den Prozessor wiederum positiv (Vorzeichen +), so daß auch in diesem Fall die Verzweigung nach $067F ausgeführt wird. Das Spiel beginnt von vorn.

Diese Schleife wird so lange durchlaufen, bis das Y-Register den Wert $00 hat und erneut dekrementiert wird. Danach hat es den Wert $FF angenommen, Bit sieben und somit das N-Flag ist gesetzt, und die Bedingung BPL... ist nicht mehr erfüllt. Das Programm fährt mit der Abarbeitung ab Adresse $0687 fort.

Wir hoffen, daß es nicht zu schwer war. Lesen Sie ruhig noch einmal von vorn, und schlagen Sie dabei auch im Kapitel über die Einführung in die Maschinensprache nach.

Wir sind auch schon fast mit der Erklärung der Laderoutine fertig. Der AKKU wird unmittelbar mit $00 geladen und in $9A abgespeichert. Durch diese Befehlsfolge wird das Ausgeben der üblichen Meldungen, wie »SEARCHING FOR...«, verhindert. Wird mit der Datasette gearbeitet, entfällt auch der Druck auf die COMMODORE-Taste. Der AKKU muß aber auch noch aus einem anderen Grund den Inhalt $00 erhalten. Bei Inhalten >0 erfolgt ein »VERIFY« bei Aufruf der Betriebssystemroutine. Dieser Aufruf erfolgt auch sogleich mit JSR $FFD5. Wenn das Programm vom Laden des DATA-Erzeugers zurückkehrt und das Carry Flag (in diesem Fall das Fehlerflag) nicht gesetzt ist, wird zur Adresse $06A9 verzweigt. Ist Carry gesetzt, so wird das Ladeprogramm vorzeitig beendet.

Wir sind nun bei unseren Bankswitchingroutinen angelangt, die ebenfalls im Lader vorhanden sind. Beide Bankswitching Routinen, die jetzt noch ab Adresse $0691 bis Adresse $06A8 zu finden sind, werden später in den Bereich ab $065E kopiert. Dieser Bereich ist ja nach dem Laden des Hauptprogramms wieder verfügbar, da das Ladeprogramm dann nicht mehr benötigt wird. Sie brauchen sich also zum Betreiben des DATA-Erzeugers nur die SYS-Adresse 1630 zu merken.

```
;********** Bankswitching Routine eins ***************

. 0691   78       SEI          ;Interrupt sperren
. 0692   8D 3F FF STA $FF3F    ;RAM einschalten
. 0695   20 00 FB JSR $FB00    ;zum DATA-Programm
. 0698   8D 3E FF STA $FF3E    ;ROM einschalten
. 069B   58       CLI          ;Interrupt wieder zulassen
. 069C   60       RTS          ;zurück ins BASIC
```

Diese erste Routine ist schnell erklärt. Nachdem ein Interrupt unterbunden wurde, wird auf das RAM umgeschaltet. Der Prozessor hat somit auf unser Programm Zugriff. Das Programm steht ab $FB00 im Speicher, und dahin wird dann per Unterprogrammaufruf verzweigt. Das Hauptprogramm endet mit RTS. Der Rücksprung erfolgt daher zur Adresse, die JSR $FB00 folgt. Es wird jetzt wieder auf ROM umgeschaltet. Das muß geschehen, damit der Prozessor wieder Zugriff auf Interpreter und Betriebssystem erhält. Nun wird der Interrupt wieder zugelassen und die Tastatur somit wieder abgefragt. Mit dem nun folgenden RTS wird ins BASIC zurückgekehrt.

```
;******** Bankswitching Routine zwei **********

. 069D   8D 3E FF STA $FF3E          ;ROM einschalten
. 06A0   58       CLI               ;Interrupt wieder ein
. 06A1   20 00 00 JSR $0000         ;siehe Text
. 06A4   78       SEI               ;Interrupt aus
. 06A5   8D 3F FF STA $FF3F         ;RAM ein
. 06A8   60       RTS               ;zurück ins Hauptprogramm
```

Mit dem SYS-Befehl, mit dem das Programm später gestartet wird, werden außer der Startadresse auch noch Parameter übergeben. Da dieser Parameter über vorhandene Betriebssytemroutinen geholt wird, wurde diese zweite Bankswitching Routine notwendig.

Besondere Aufmerksamkeit müssen wir hier dem Befehl JSR $0000, jetzt Adresse $06A1 später $066E, widmen. Die Adresse, die diesem Befehl folgt, wurde willkürlich gewählt. Sie wird vom Hauptprogramm, den Erfordernissen entsprechend, geändert. Unser Prozessor verfügt über keine Möglichkeit indirekt Unterprogramme aufzurufen. Da wir aber vor dem Aufruf des Betriebssytems auf ROM umschalten müssen, wurde von uns eine Möglichkeit ersonnen, die es dem Prozessor nach der Umschaltung ermöglicht die gewünschte Adresse des Betriebssystems anzuspringen.

Vor dem Aufruf dieser Routine werden von unserem Hauptprogramm die beiden Byte, die JSR folgen so verändert, daß sie unser Sprungziel enthalten. So wird es möglich, auch vom RAM aus die Routinen des ROMs zu nutzen. Die anderen Befehle werden Sie sicher schon verstanden haben, so daß wir auf eine Erklärung verzichten können.

```
;******** Verschieberoutine für Bankswitchingroutinen ******

. 06A9   A0 17    LDY #$17          ;Anzahl der Byte = $18
. 06AB   B9 91 06 LDA $0691,Y       ;mit $0691 + Y laden
. 06AE   99 5E 06 STA $065E,Y       ;nach $065E + Y speichern
. 06B1   88       DEY               ;Y-Register dekrementieren
. 06B2   10 F7    BPL $06AB         ;nicht fertig, dann $06AB
. 06B4   60       RTS               ;Ende des Laders
```

Auch zu diesen Zeilen bedarf es noch einiger kurzer Beschreibungen. Diese Befehle bewirken das Verschieben der Bankswitching Routinen von $0691 nach $065E. Die Länge des zu verschiebenden Bereichs beträgt $18 Byte. Das Y-Register ist hier auch wieder sowohl Zähl- als auch Hilfsregister zur Adressierung. Da dies Programm auch noch fortfährt, wenn Y den Wert $00 angenommen hat, ergibt sich die Anzahl der Byte ($17+$01). Der AKKU wird absolut nachindiziert geladen und auch abgespeichert. Sollten Sie Schwierigkeiten haben, so ziehen Sie noch einmal die Beschreibung

unserer ersten Verschieberoutine und das Kapitel über die Einführung in die Maschinensprache zu Rate. Nach der Verschiebung kehrt das Programm durch RTS ins BASIC zurück.

Mit dem Lader zusammen wurde aber noch mehr geladen.

```
;****** OLD Programm *************

. 06B5   A9 01     LDA #$01       ;AKKU mit $01 laden
. 06B7   A0 00     LDY #$0D       ;Y-Register mit $0D laden
. 06B9   91 2B     STA ($2B),Y    ;indirekt Y indiziert speichern
. 06BB   20 18 88  JSR $8818      ;neue Koppeladresse berechnen
. 06BE   20 4B 88  JSR $884B      ;Variablenanfang neu berechnen
. 06C1   4C 9A 8A  JMP $8A9A      ;CLR ausführen
```

Das OLD-Programm wurde schon vorher beschrieben, darum soll an dieser Stelle nur auf das Warum eingegangen werden. Immer wenn von BASIC aus absolut geladen wird, mit Sekundäradresse 1, sind nach dem Ladevorgang wichtige Zeiger verbogen. Berichtig werden sie dann mit NEW. Damit dieses Programm im Bedarfsfall auch nachgeladen werden kann, wurde das OLD-Programm angehängt. Sie führen also NEW aus, und springen dann mit SYS 1717 in die OLD-Routine um Ihr BASIC-Programm zurückzuholen. Danach kann entweder weiter in BASIC gearbeitet oder in den DATA-Erzeuger mit SYS 1630 gesprungen werden.

Es fehlt nun nur noch der Programmnamen des Hauptprogramms.

M 06C4 06D0

```
>06C4 44 41 54 41 47 45 4E 45 :DATAGENE
>06CC 52 41 54 4F 52 :RATOR
```

Der Name steht hinter unserem Ladeprogramm und wird selbstverständlich auch gleich mitgeladen.

Wir sind am Ende der Beschreibung des Laders angekommen und hoffen, daß es nicht zu schwer war, uns zu folgen. Die Sicherheit kommt aber erst, wenn man Eigeninitiative ergreift und selbst programmiert, und für Verbesserungen ist unser Lader sicher gut. Vielleicht sind dem einen oder anderen auch schon welche eingefallen, also nichts wie ran. Kommen wir jetzt zum eigentlichen DATA-Erzeuger.

21.6.5 Das Hauptprogramm des DATA-Erzeugers

Auch für das Hauptprogramm des DATA-Generators haben wir einen BASIC-Lader geschrieben. Die DATA-Zeilen erzeugte schon das Programm. Toll, was? Tippen Sie nun erst einmal den Lader ein, speichern ihn, und starten dann den Monitor. Jetzt folgt:

>07F8 80

Dieser Befehl schaltet den Monitor auf RAM Zugriff.

S "DATAGENERATOR",1,FB00,FBFC

Um lange Wartezeiten beim Wiedereinladen des Programms zu vermeiden, sollten Besitzer eines Kassettenrecoders darauf achten, daß dieses Programm dem Ladeprogramm unmittelbar folgt. Das Programm muß unter dem Namen »DATAGENERATOR« abgespeichert werden, es sei denn, man ändert den Namen im Lader ebenfalls ab.

So, dies soll als Vorbemerkung reichen, mehr gibt's im Anschluß Ihrer Fleißarbeit.

Der BASIC-Lader

```
10 FOR I=DEC("FB00") TO DEC("FBFB")
20 READ A$
30 POKE I,DEC(A$)
40 B=B+DEC(A$)
50 NEXT
60 IF B<>31431 THEN PRINT"FEHLER IN DATAS"
70 :
100 DATA A2,06,8A,48,A9,91,8D,6F,06
110 DATA A9,94,8D,70,06,20,6A,06,A9
120 DATA E1,8D,6F,06,A9,9D,8D,70,06
130 DATA 20,6A,06,68,AA,A5,15,95,D0
140 DATA C4,94,D0,CA,D0,D8,D8,38,A5
150 DATA D6,48,A5,D3,E5,D5,85,D3,B0
160 DATA 05,E6,D6,F0,07,38,A5,D4,E5
170 DATA D6,B0,11,A2,0E,A9,83,8D,6F
180 DATA 06,A9,86,8D,70,06,20,6A,06
190 DATA 68,60,85,D4,68,85,D6,A5,2D
200 DATA E9,02,B0,02,C6,2E,85,2D,A0
210 DATA 02,A5,D1,91,2D,C8,A5,D2,91
220 DATA 2D,C8,A9,83,91,2D,C8,A9,20
230 DATA 91,2D,C8,A2,00,A1,D5,48,29
240 DATA F0,4A,4A,4A,4A,AA,BD,EC,FB
250 DATA 91,2D,C8,68,29,0F,AA,BD,EC
260 DATA FB,91,2D,C6,D3,D0,06,A5,D4
```

```
270 DATA F0,15,C6,D4,48,E6,D5,D0,02
280 DATA E6,D6,C0,1F,F0,09,C8,A9,2C
290 DATA 91,2D,68,D0,C7,48,C8,A9,00
300 DATA 91,2D,C8,98,48,A0,00,18,65
310 DATA 2D,91,2D,C8,A6,2E,90,01,E8
320 DATA 8A,91,2D,E6,D1,D0,02,E6,D2
330 DATA 68,18,65,2D,85,2D,90,02,E6
340 DATA 2E,68,D0,86,A0,00,A2,02,91
350 DATA 2D,E6,2D,D0,02,E6,2E,CA,D0
360 DATA F5,60,30,31,32,33,34,35,36
370 DATA 37,38,39,41,42,43,44,45,46
```

Läuft der BASIC-Lader nach dem Eingeben einwandfrei und haben Sie alle oben genannten Schritte ausgeführt, so steht einer Erprobung nichts mehr im Wege. Lassen Sie bitte für die folgenden Erklärungen den BASIC-Lader im Speicher stehen.

Laden Sie das Ladeprogramm mit:

LOAD "LADER",1,1 Diskettenbesitzer mit 8,1 in den Speicher.

Starten Sie nun den Lader mit :

SYS 1630

Jetzt müßte nach »PRESS PLAY ON TAPE« und dessen Befolgung das Hauptprogramm gesucht und dann geladen werden. Sollte bei Besitzern einer Diskstation diese Meldung ebenfalls erscheinen, so wurde vergessen im BASIC-Lader die Gerätenummer zu ändern. Lesen Sie dazu beim Ladeprogramm noch einmal nach. Nach dem Laden folgt:

NEW

und dann SYS 1717. Mit LIST muß jetzt der BASIC-Lader für das Hauptprogramm wieder auf dem Bildschirm erscheinen. So, jetzt kommt der Punkt, an dem der Frosch ins Wasser springt und sein Leben riskiert:

SYS 1630,DEC("FB00"),DEC("FBFC"),1000

Da durch den Zeilenerzeuger einige Zeiger nicht mehr den richtigen Inhalt besitzen, folgt noch:

CLR

Sollte sich nach diesem Befehl der Plus/4 mit »READY« zurückmelden, so ist das schon toll, erscheinen aber nach »LIST« in den Zeilen ab 1000 dieselben DATA-Zei-

len wie in den Zeilen von 100 bis 370, dann ist das sensationell. Ihr DATA-Generator funktioniert.

Die Syntax unseres SYS-Befehls lautet also:

SYS 1630, Von, Bis+1, erste Zeilennummer

Wird einer der Parameter falsch angegeben, oder wird ein Komma vergessen, erscheint die Fehlermeldung »SYNTAX ERROR«. Auch dieses Programm soll erklärt werden.

Unser Programm benötigt einige Zwischenspeicher. Dies sind:

```
Adresse                 Zweck (Inhalt)

$D1                     Low Byte der Zeilennummer
$D2                     Hight Byte der Zeilennummer
$D3                     Low Byte Bis+1, später LB Anzahl
$D4                     Hight Byte Bis+1, später HB Anzahl
$D5                     Low Byte Von
$D6                     Hight Byte Von
```

```
; ******* Werte die SYS folgen auswerten und speichern *******

. FB00   A2 06      LDX #$06      ;X unmittelbar mit $06 lad.
. FB02   8A         TAX           ;X in AKKU
. FB03   48         PHA           ;AKKU auf Stapel bringen
. FB04   A9 91      LDA #$91      ;LB Sprungadresse
. FB06   8D 6F 06   STA $066F     ;in $066F speichern
. FB09   A9 94      LDA #$94      ;HB Sprungadresse
. FB0B   8D 70 06   STA $0670     ;in $0670 speichern
. FB0E   20 6A 06   JSR $066A     ;Unterprogramm Bankswitching
. FB11   A9 E1      LDA #$E1      ;LB Sprungadresse
. FB13   8D 6F 06   STA $066F     ;in $066F speichern
. FB16   A9 9D      LDA #$9D      ;HB Sprungadresse
. FB18   8D 70 06   STA $0670     ;in $0670 speichern
. FB1B   20 6A 06   JSR $066A     ;Unterprgm. Bankswitching
. FB1E   68         PLA           ;AKKU vom Stapel holen
. FB1F   AA         TAX           ;AKKU in X transportieren
. FB20   A5 15      LDA $15       ;mit Inhalt von $15 laden
. FB22   95 D0      STA $D0,X     ;speichern in D0+X
. FB24   CA         DEX           ;X dekrementieren
. FB25   94 D0      STY $D0,X     ;Y in $D0+X speichern
. FB27   CA         DEX           ;X herunterzählen
. FB28   D0 D8      BNE $FB02     ;<>0, dann verzweigen
```

In diesem Programmteil wird das X-Register sowohl als Zähler als auch als Adressregister verwendet. Aus unserer Tabelle geht hervor, daß sechs Werte in den Bereich

$D1 bis $D6 zu übertragen sind. Daher erfolgt das Laden des X-Registers am Anfang des Programms mit $06. Von der Adresse $FB04 bis Adresse $FB0B wird der JSR-Befehl (JSR $0000) unserer Bankswitchingroutine manipuliert. Nach der Operation steht ab Adresse 066E jetzt JSR $9491. Es folgt der Sprung in die Bankswitchingroutine, und diese ruft nach dem Umschalten auf ROM mit JSR $9491 eine Betriebssystemroutine auf. Diese ist für das Überprüfen auf Komma verantwortlich. Es erfolgt dann der Rücksprung ins Bankswitching, dort wird auf RAM umgeschaltet und dann zur Adresse $FB11 unseres Programms zurückgesprungen.

Die Adressen $066F und $0670 werden erneut manipuliert, so daß das eigentliche Sprungziel nun JSR $9DE1 heißt. Wieder folgt ein Unterprogrammaufruf. Vom Bankswitching verzweigt das Programm ins Betriebssystem zur Adresse $9DE1. Hier wird der Ausdruck, der dem Komma folgt, ausgewertet. Der sich hieraus ergebende zwei Byte-Wert wird dem Y-Register und der Speicherstelle $15 übergeben. Mit dieser Routine lassen sich also nur Zahlen bis max 65535 verarbeiten. Ist die Zahl größer, so wird das Programm mit einer Fehlermeldung abgebrochen.

Nach der Rückkehr in unser Programm, werden die beiden Byte, durch die Nachindizierung mit dem X-Register, in den jeweiligen Speicherstellen gespeichert. Das X-Register wird dabei zweimal dekrementiert. Erreicht es dabei den Wert null, so wird mit der Abarbeitung des nächsten Befehls begonnen. Bei Ergebnissen ungleich null wird zur Adresse $FB02 verzweigt, und der nächste Ausdruck ausgewertet.

```
; ***** Anzahl berechnen, evtl. Fehlermeldung ausgeben *****

. FB2A   D8          CLD          ;Dezimalflag löschen
. FB2A   38          SEC          ;Carryflag setzen
. FB2C   A5 D6       LDA $D6      ;mit HB ANFANG laden
. FB2E   48          PHA          ;HB retten
. FB2F   A5 D3       LDA $D3      ;mit LB BIS+1 laden
. FB31   E5 D5       SBC $D5      ;AKKU minus LB ANFANG
. FB33   85 D3       STA $D3      ;ANZAHL LB in $D3 speichern
. FB35   B0 05       BCS $FB3C    ;nichts geborgt, weiter $FB3C
. FB37   E6 D6       INC $D6      ;sonst HB ANFANG inkrementieren
. FB39   F0 07       BEQ $FB42    ;falls Überlauf dann $FB42
. FB3B   38          SEC          ;Carry wieder setzen
. FB3C   A5 D4       LDA $D4      ;mit HB BIS+1 laden
. FB3E   E5 D6       SBC $D6      ;AKKU minus HB ANFANG
. FB40   B0 11       BCS $FB53    ;nichts geborgt, weiter $FB53
. FB42   A2 0E       LDX #$0E     ;X mit Fehlernummer laden
. FB44   A9 83       LDA #$83     ;LB Fehlerroutine
. FB46   8D 6F 06    STA $066F    ;LB des JSR-Befehls
. FB49   A9 86       LDA #$86     ;HB der Fehlerroutine
. FB4B   8D 70 06    STA $0670    ;HB unseres JSR-Befehls
. FB4E   20 6A 06    JSR $066A    ;Sprung ins Bankswitching
. FB51   68          PLA          ;Stapel bereinigen
. FB52   60          RTS          ;Programm vorzeitig beenden
```

```
. FB53   85 D4      STA $D4     ;HB ANZAHL speichern
. FB55   68         PLA         ;Stapelspitze in AKKU
. FB56   85 D6      STA $D6     ;altes HB ANFANG wieder in $D6
```

In diesem Programmteil wird die Anzahl der zu erstellenden DATAs errechnet. Wurde ANFANG größer als BIS+1 gewählt, wird das X-Register mit der Fehlernummer $0E geladen. Über unsere Bankswitchingroutine erfolgt die Verzweigung zur Fehlerausgabe des Betriebssystems (Adresse $8683). Die Fehlermeldung »ILLEGAL QUANTITY« erscheint auf dem Bildschirm und das Programm wird, nachdem der Stapel bereinigt wurde, vorzeitig beendet.

Ist das Ergebnis in Ordnung, folgt die Fortsetzung ab Adresse $FB53. Da der Speicherinhalt von $D6 evtl. erhöht wurde (Adresse $FB37), erhält das HB von ANFANG wieder den alten Wert. Die Anzahl der DATAs befindet sich jetzt in $D3 und $D4. Der Wert BIS+1 wird nicht mehr benötigt.

Denken Sie daran, daß beim Aufruf der Fehlerroutine die Fehlernummer immer im X-Register stehen muß. Eine Aufstellung aller verfügbaren Meldungen und deren Nummern finden Sie im Anhang.

```
; ***** Einrichten eines Zeigers NACH ******

. FB58   A5 2D    LDA $2D     ;LB Variablenanfang
. FB5A   E9 02    SBC #$02    ;minus $02
. FB5C   B0 02    BCS $FB60   ;nichts geborgt, weiter $FB60
. FB5E   C6 2E    DEC $2E     ;sonst HB dekrementieren
. FB60   85 2D    STA $2D     ;neues LB in $2D
```

Jetzt, da die Vorarbeiten geleistet sind, können wir voll loslegen. Da das Programm wissen muß, an welcher Stelle des Speichers die Zeilennummern, Koppeladressen und DATAs abzulegen sind, wird der Zeiger NACH eingerichtet. In der oben stehenden Routine wird dazu der Zeiger auf den Variablenanfang »verbogen«. Er zeigt dann auf das Byte hinter der ersten Null, die den Abschluß einer BASIC-Zeile bildet. Dieser Schritt ermöglicht uns den DATA-Generator auch dann zu starten, wenn sich bereits ein BASIC-Programm im Speicher befindet.

```
; ****** Zeilennummer erzeugen *****

. FB62   A0 02     LDY #$02       ;Y mit Startwert laden
. FB64   A5 D1     LDA $D1        ;LB der Zeilennummer
. FB66   91 2D     STA ($2D),Y    ;in Zeiger+Y speichern
. FB68   C8        INY            ;Y inkrementieren
. FB69   A5 D2     LDA $D2        ;HB der Zeilennummer
. FB6B   91 2D     STA ($2D),Y    ;in Zeiger+Y speichern
```

Byte eins und zwei der zu erstellenden BASIC-Zeile bleiben vorerst frei. In Byte drei und vier wird das Low und High Byte der Zeilennummer abgelegt. Dazu wird zum

Zeiger NACH der Inhalt des Y-Registers addiert (indirekt nachindizierte Adressierung).

```
; ****** DATA-Token in die Zeile bringen ********

. FB6D   C8        INY            ;Y um eins erhöhen
. FB6E   A9 83     LDA #$83       ;mit DATA-Token laden
. FB70   91 2D     STA ($2D),Y    ;in Zeiger+Y speichern
```

Das DATA-Token wird als fünftes Byte in der Zeile gespeichert

```
; ****** SPACE einfügen *******

. FB72   C8        INY            ;Y erhöhen
. FB73   A9 20     LDA #$20       ;CHR$ von SPACE
. FB75   91 2D     STA ($2D),Y    ;in Zeiger+Y speichern
```

Um die Übersichtlichkeit der erstellten Zeile zu erhöhen, wird ein Leerzeichen zwischen DATA und dem ersten DATA-Element eingefügt.

```
; ****** DATAs erzeugen und speichern ********

. FB77   C8         INY           ;Y erhöhen
. FB78   A2 00      LDX #$00      ;X auf 0 setzen
. FB7A   A1 D5      LDA ($D5,X)   ;mit ANFANG+X laden
. FB7C   48         PHA           ;AKKU retten
. FB7D   29 F0      AND #$F0      ;Bits 0-3 ausblenden
. FB7F   4A         LSR           ;Bits 4-7 in Bits 0-3
. FB80   4A         LSR           ;schieben
. FB81   4A         LSR           ;
. FB82   4A         LSR           ;
. FB83   AA         TAX           ;Ergebnis ins X-Register
. FB84   BD EC FB   LDA $FBEC,X   ;mit entsprechendem Wert laden
. FB87   91 2D      STA ($2D),Y   ;und in Zeile speichern
. FB89   C8         INY           ;Y erhöhen
. FB8A   68         PLA           ;geretteten Wert vom Stapel
. FB8B   29 0F      AND #$0F      ;Bits 4-7 löschen
. FB8D   AA         TAX           ;Ergebnis ins X-Register
. FB8E   BD EC FB   LDA $FBEC,X   ;2.Wert des DATAs
. FB91   91 2D      STA ($2D),Y   ;in Zeile speichern
```

Da der Prozessor über keinen indirekten Ladebefehl verfügt, wird das X-Register auf null gesetzt und der AKKU dann indirekt indiziert geladen. Er erhält also den Wert, auf den der Zeiger ANFANG zeigt.

Jedes DATA soll später aus zwei Zeichen zusammengesetzt sein (HEX-Darstellung). Der AKKU-Inhalt wird zur Erzeugung des ersten Zeichens umgewandelt, muß aber auch noch später für das zweite Zeichen verfügbar sein. Darum wird der Inhalt auf den Stapel gebracht. Jetzt wird der AKKU mit dem Wert $F0 AND verknüpft, d.h., die Bits 0-3 werden gelöscht. Durch viermaliges Rechtsschieben des Ergebnisses, teilen der Zahl durch 16, erhalten wir einen Zeiger, mit dem das erste Zeichen aus der ASCII-Tabelle gelesen werden kann. Diese steht ab Adresse $FBEC. Der Zeiger wird ins X-Register transportiert und der AKKU mit dem Element geladen, welches in $FBEC+X enthalten ist. Dieses Element wird dann an der entsprechenden Stelle der Programmzeile abgelegt.

Um den Zeiger auf das zweite DATA-Element zu erhalten, wird der Inhalt der Stapelspitze in den AKKU gebracht, dieser dann mit $0F AND verknüpft, und das Ergebnis wieder ins X-Register transportiert. Die nun folgenden sind mit den eben beschriebenen identisch. Um das eben Erklärte zu verdeutlichen, hier die Tabelle:

M FBEC FBFB

```
>FBEC 30 31 32 33 34 35 36 37 :01234567
>FBF4 38 39 41 42 43 44 45 46 :89ABCDEF
```

Es geht weiter mit:

```
; ****** ANZAHL dekrementieren ******

. FB93   C6 D3    DEC $D3       ;LB ANZAHL dekrementieren
. FB95   D0 06    BNE $FB9D     ;kein Überlauf, dann $FB9D
. FB97   A5 D4    LDA $D4       ;HB ANZAHL
. FB99   F0 15    BEQ $FBB0     ;auch Null, dann fertig
. FB9B   C6 D4    DEC $D4       ;sonst HB ANZAHL dekrement.
```

Hier wird die Anzahl der noch zu erzeugenden DATAs um eins heruntergezählt. Aus dieser Routine ist ersichtlich, warum der Wert BIS als BIS+1 eingegeben werden muß.

```
; ****** Zeiger ANFANG erhöhen *******

. FB9D  48       PHA           ;Flag retten
. FB9E  E6 D5    INC $D5       ;LB ANFANG erhöhen
. FBA0  D0 02    BNE $FBA4     ;kein Überlauf, dann $FBA4
. FBA2  E6 D6    INC $D6       ;sonst HB ANFANG erhöhen
```

Zuerst wird der AKKU auf den Stapel gebracht. Sein Inhalt dient uns später als Flag. Es folgt das Heraufzählen von ANFANG, an dieser Stelle wäre der Name VON wohl besser.

```
; ****** Y-Register auf Höchstwert überprüfen *****

. FBA4   C0 1F   CPY #$1F      ;Inhalt von Y=$1F ?
. FBA6   F0 09   BEQ $FBB1     ;ja, dann $FBB1
. FBA8   C8      INY           ;Y erhöhen
. FBA9   A9 2C   LDA #$2C      ;CHR$ 'Komma'
. FBAB   91 2D   STA ($2D),Y   ;in Zeile speichern
. FBAD   68      PLA           ;AKKU von Stapel
. FBAE   D0 C7   BNE $FB77     ;immer, da immer <>0
```

Warum $1F? Na, rechnen wir doch einmal, und dies weil's leichter geht, im Dezimalsystem:

Jede DATA-Zeile soll neun DATAs enthalten. Jedes DATA-Element besteht aus zwei Werten. Das macht 18 Byte. Zwischen Element 1 und 9 steht jeweils ein Komma. Das macht zusammen 26 Byte. Dazu kommt ein Space, ein DATA-Token und je zwei Byte für die Koppeladresse und die Zeilennummer. Es ergibt sich somit der Bedarf von 32 Byte pro Zeile (ohne die Null für das Zeilenende). Das Y-Register wird später Null, daher der Wert 31 oder $1F.

Möchten Sie mehr oder auch weniger Datas pro Zeile, so muß dieser Wert nur in Dreierschritten erhöht oder vermindert werden. Da bei der »Eingabe« der Interpreter ausgeschaltet ist, können so Zeilen erzeugt werden, die länger als 88 Zeichen sind.

Machen wir weiter:

Ist der Inhalt des Y-Registers kleiner, so wird ein Komma in die Zeile geschrieben. Jetzt wird der Inhalt der Stapelspitze in den AKKU geholt. Der Inhalt ist, wenn das Programm an dieser Stelle »landet«, immer <>0. Sehen Sie sich dazu noch einmal die vorangegangene Routine an. Das Programm verzweigt wieder zur Erzeugung der DATA-Elemente.

```
; ****** Null für Ende der Programmzeile erzeugen *******

. FBB0   48      PHA           ;Flag auf Stapel
. FBB1   C8      INY           ;Y erhöhen
. FBB2   A9 00   LDA #$00      ;Ende der Zeile
. FBB4   91 2D   STA ($2D),Y   ;an NACH+Yspeichern
```

Ist das Programm fertig, so muß erst noch das Flag gerettet werden. Ist lediglich eine Zeile zu Ende, so beginnt der Einstieg bei $FBB1. In beiden Fällen wird die Zeile, wie es sich gehört, mit einer Null abgeschlossen.

```
; ** Koppeladresse berechnen und am Zeilenanfang speichern. **

. FBB6   C8       INY              ;Y erhöhen
. FBB7   98       TYA              ;Inhalt von Y
. FBB8   48       PHA              ;retten
. FBB9   A0 00    LDY #$00         ;Y auf 0 setzen
. FBBB   18       CLC              ;Carry löschen
. FBBC   65 2D    ADC $2D          ;AKKU+Inhalt $2D
. FBBE   91 2D    STA ($2D),Y      ;Ergebnis=LB Koppeladresse
. FBC0   C8       INY              ;Y erhöhen
. FBC1   A6 2E    LDX $2E          ;X mit Inhalt $2E laden
. FBC3   90 01    BCC $FBC6        ;kein Übertrag, weiter $FBC6
. FBC5   E8       INX              ;sonst X erhöhen
. FBC6   8A       TXA              ;X in AKKU transportieren
. FBC7   91 2D    STA ($2D),Y      ;HB der Koppeladresse
```

Ob nun das Programm zu Ende ist oder lediglich die maximale Anzahl von DATAs pro Zeile erreicht wurde, die Koppeladresse muß auf jeden Fall berechnet werden. Das Ergebnis der Berechnung wird in die ersten beiden Byte der Zeile eingetragen.

Um beiden Fällen Rechnung zu tragen, wurde diese Routine etwas länger. Vielleicht können Sie es besser? Nur zu, aber lesen Sie vorher bitte noch weiter, damit Sie erfahren, warum auch das X-Register von uns benutzt wurde.

```
; ****Zeilennummer und Zeiger NACH erhöhen *****

. FBC9   E6 D1    INC $D1          ;LB der Zeilennr. erhöhen
. FBCB   D0 02    BNE $FBCF        ;kein Überlauf, dann $FBCF
. FBCD   E6 D2    INC $D2          ;sonst HB Zeilennr. erhöhen
. FBCF   68       PLA              ;ehemaliges Y vom Stapel
. FBD0   18       CLC              ;Carry löschen
. FBD1   65 2D    ADC $2D          ;AKKU+Inhalt von $2D
. FBD3   85 2D    STA $2D          ;Ergebnis in LB NACH speichern
. FBD5   90 02    BCC $FBD9        ;kein Übertrag, weiter $FBD9
. FBD7   E6 2E    INC $2E          ;sonst $2E erhöhen
. FBD9   68       PLA              ;Flag vom Stack
. FBDA   D0 86    BNE  $FB62       ;nicht fertig, neue Zeile
```

In diesem Programmteil erfolgt zunächst das Erhöhen der Zeilennummer. Dies geschieht auch, wenn das Programm mit der Erzeugung der DATAs fertig ist. Als nächstes muß noch der Zeiger NACH dem aktuellen Stand angepaßt werden, da das Y-Register bei der Erstellung einer weiteren Zeile wieder auf den Anfangswert $02 gebracht wird. Mit PLA wird unser Flag vom Stapel geholt. Ist es 0, dann wird das Programm nach Abarbeitung der nächsten Routine beendet. Ist es <> 0, so wird zum Anfang der Zeilenerzeugung verzweigt.

```
; ****** Programmende *******

. FBDC   A0 00   LDY #$00       ;Y Null setzen
. FBDE   A2 02   LDX #$02       ;X auf $02 setzen
. FBE0   91 2D   STA ($2D),Y    ;zweite Null
. FBE2   E6 2D   INC $2D        ;Zeiger NACH erhöhen
. FBE4   D0 02   BNE $FBE8      ;kein Überlauf, dann $FBE8
. FBE6   E6 2E   INC $2E        ;sonst HB NACH erhöhen
. FBE8   CA      DEX            ;X dekrementieren
. FBE9   B0 F5   BNE $FBE0      ;nicht fertig, 3. Null erzeugen
. FBEB   60      RTS            ;Ende des DATA-Generators
```

Diese Zeilen werden erst bei der Beendigung des Programms abgearbeitet. Da der letzten Programmzeile immer zwei Nullen folgen müssen, werden sie hier erzeugt. Das X-Register fungiert als Schleifenzähler. In der Schleife wird auch unser Zeiger NACH wieder in den Zeiger auf den Variablenanfang umgewandelt. Dieser zeigt danach wieder auf das Byte, das der letzten Null folgt.

Wir sind am Ende unserer Programmbeschreibung angekommen und hoffen, Ihnen die Maschinensprache noch ein Stück nähergebracht zu haben. Wenn Sie die vorgestellten Routinen und deren Funktionen verstanden haben, sollten Sie versuchen, das Programm zu verbessern. Angriffspunkte bietet es genug, so wird es z. B. nicht auf das Speicherende überprüft, und so kann es passieren, daß sich das Programm selbst überschreibt. Wir wollten aber keinen Programmierwettbewerb gewinnen, sondern Ihnen lediglich an Hand eines nutzbringenden Programms den Einsatz der meisten Befehle erklären.

So, von jetzt an kann Ihr Plus/4 auch selbst BASIC-Zeilen erzeugen.

22 Der User-Port

Alle Commodore-Computer (bis auf C16 und C116) haben eine besondere Schnittstelle, den User-Port. Wie der Name schon sagt, handelt es sich dabei um eine User-(Benutzer)-Schnittstelle. Diese Schnittstelle zeichnet sich dadurch aus, daß sie universell programmierbar ist. So können damit z. B. problemlos irgendwelche externen Geräte ein- oder ausgeschaltet werden. Man kann den User-Port auch als serielle Schnittstelle verwenden. Die Möglichkeiten sind also vielfältig und kaum beschränkt.

So schön diese Schnittstelle auch sein mag, einen Wermutstropfen gibt es doch dabei: Die Commodore-Konstrukteure haben für jeden Commodore-Computer einen neuen User-Port konstruiert, nur von Kompatibilität kann dabei leider keine Rede sein. Das bedeutet, daß Geräte, die am User-Port des C64 einwandfrei funktionieren, am Plus/4 nicht angeschlossen werden können. Es wäre wohl zuviel verlangt, diese Schnittstelle kompatibel zu machen.

Leider können wir das auch nicht ändern. Aber dafür zeigen wir Ihnen jetzt, was Sie mit dem User-Port alles machen können, und wie er programmiert wird. In diesem Kapitel beschreiben wir auch, wie man einen Drucker mit Centronics-Schnittstelle am Plus/4 anschließt. Somit müssen Sie sich nicht mehr auf die Commodore-Drucker beschränken und können weitaus leistungsfähigere, handelsübliche Drucker verwenden.

Hier zunächst die Stiftbelegung des User-Ports:

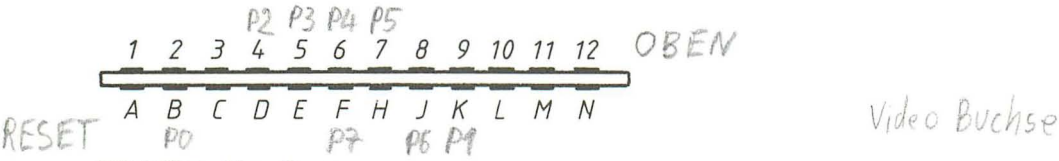


Bild 22.1: *User-Port*

Tabelle 22.1: *Stiftbelegung des User-Ports*

Stift	Bezeichnung	Stift	Bezeichnung
1	Masse	A	Masse
2	+5 Volt	B	P0
3	-BRESET	C	RxD
4	P2	D	RTS
5	P3	E	DTR
6	P4	F	P7
7	P5	H	DCD
8	RxC	J	P6
9	ATN	K	P1
10	9 Volt AC	L	DSR
11	9 Volt AC	M	TxD
12	Masse	N	Masse

Zunächst sehen wir uns den Stift mit der Bezeichnung ATN an. Es handelt sich dabei um den Anschluß P2 des Mikroprozessors. Das ist eine der sieben programmierbaren Ausgangsleitungen des 7501. Diese liegt über einem Inverter an ATN. ATN ist aber auch am seriellen Bus angeschlossen. Wenn dort ein Diskettenlaufwerk angeschlossen ist, können wir, wenn ATN umprogrammiert wird, eigenartige Fehlermeldungen erhalten und auf das Diskettenlaufwerk zugreifen. Also müssen wir mit ATN etwas vorsichtig umgehen.

An den Stiften 10 und 11 liegt eine Wechselspannung von 9 Volt. Diese kann für angeschlossene Geräte verwendet werden. Bedenken Sie aber dabei, daß das Plus/4-Netzteil die Wechselspannung mit maximal 1 Ampere liefert, dann aber auch sehr heiß wird.

BRESET ist die RESET-Leitung für extern angeschlossene Geräte. Wird vom Plus/4 ein RESET durchgeführt, wird diese Leitung »0«. Es handelt sich also um ein low-aktives Signal. Wird dagegen von einem externen Gerät ein RESET durchgeführt, bleibt der Plus/4 davon unbetroffen. Zwei ICs belegen die übrigen Stifte des User-Ports. Diese Chips tragen die Bezeichnung 6551 und 6529.

Übrigens besitzt der Plus/4 schon die Software für eine RS-232-Schnittstelle. Die Bedienung ist ausführlich im Plus/4-Bedienungshandbuch beschrieben. Dort werden auch zwei Register des 6551 erläutert. Leider vergaß man die Adressen der Register anzugeben. Das IC 6551 ist ein ACIA (Asynchronous Communications Interface Adapter). Dieses IC dient als Schnittstelle zwischen Computer und seriellen Datengeräten oder Modems.

Das 6551 belegt folgende Stifte des User-Ports:

Stift	Bezeichnung
M	TxD
E	DTR
D	RTS
8	RxC
H	DCD
L	DSR
C	RxD

Mit dem 6551 können Daten mit Bitgeschwindigkeiten von 50 Baud (Bit pro Sekunde) bis 19200 Baud seriell übertragen werden. Am IC ist ein 1,8432-MHz-Quarz angeschlossen, von dem die Bitgeschwindigkeiten abgeleitet werden.

Das IC besitzt vier Register mit folgenden Adressen und Bezeichnungen:

$FD00	Register 0	Data Register
$FD01	Register 1	Status Register
$FD02	Register 2	Command Register
$FD03	Register 3	Control Register

Mit dem 6551 ist Voll- oder Halbduplex, Signalübertragungen von 5 bis 9 Bit und eine serielle Echo-Betriebsart usw. möglich. Wir können hier nicht das ganze Datenblatt zum 6551 abdrucken. Wer sich dafür interessiert, kann sich das Datenblatt im Elektronikversand oder in besseren Elektronikgeschäften besorgen. Für denjenigen, der das IC programmieren möchte und ein Datenblatt besorgt hat, aber keinen Plus/4-Schaltplan besitzt, sei hier noch bemerkt, daß einige Anschlüsse des 6551 über Inverter an den User-Port angeschlossen sind. Pin 11 (-DTR) und Pin 8 (-RTS) sind über Inverter an Stift E und D des User-Ports angeschlossen. Pin 5 (RxC) und Pin 12 (RxD) sind direkt angeschlossen. Dagegen sind die User-Port-Stifte H und L über Inverter mit Pin 16 (-DCD) und Pin 17 (-DSR) und über 10KOhm-Widerstände an +5 Volt angeschlossen. Pin 9 des ICs (-CTS) liegt an Masse.

Kommen wir nun zum 6529. Dieses IC ist im Gegensatz zum 6551 recht spartanisch ausgefallen. Es handelt sich hierbei um einen 8-Bit-Parallelschnittstellenbaustein. Die acht Ausgänge des ICs sind am User-Port mit P0 bis P7 bezeichnet. Das 6529 besitzt nur ein Register, das die Adresse $FD10 belegt.

22.1 Die Programmierung

Die Programmierung des 6529 ist recht einfach und braucht auch kein Datenblatt. Wenn wir das Register mit einem Wert beschreiben, nehmen die acht Ausgangsleitungen sofort den entsprechenden Zustand an. Geben wir z. B. folgenden Befehl:

```
POKE 64784,255
```

oder in Maschinensprache

```
LDA #$FF ;Bitmuster %11111111
STA $FD10
```

Hierdurch werden alle acht Ausgänge des ICs »1«. Sollen dagegen nur die Ausgänge P4 bis P7 den Zustand »1« bekommen, brauchen wir den Befehl:

```
POKE 64784,240
```

oder in Maschinensprache

```
LDA #$F0 ;Bitmuster %11110000
STA $FD10
```

So können wir jede Ausgangsleitung ein- und ausschalten, indem wir das entsprechende Bitmuster im Register des 6529 ablegen.

Die Ausgänge des ICs können auch als Eingänge verwendet werden. Dazu gibt es zwei Möglichkeiten: Die erste wäre, die entsprechende Ausgangsleitung im Register des 6529 auf »0« zu setzen. Wenn dann von außen an diese Leitung eine »1«, also +5 Volt anliegt, dann wird das entsprechende Bit im Register »1«. Das läßt sich ganz einfach feststellen, indem man das Register ausliest. Damit das funktioniert, müssen allerdings enorm hohe Ströme fließen, die auf längere Sicht unseren 6529 und eventuell auch die Peripherie zerstören. Deshalb: VERGESSEN SIE DIESE ERSTE MÖGLICHKEIT.

Sehr gut funktioniert dagegen folgendes: Wir programmieren die entsprechende Ausgangsleitung auf »1«. Wenn nun die Leitung extern auf Masse gelegt wird, also den Zustand »0« bekommt, wird beim Lesen des 6529-Registers auch das entsprechende Bit »0«. Dazu ein Beispiel:

Am Anschluß P3 des User-Ports haben wir einen Schalter angeschlossen, z. B. einen Fensterkontakt (Alarmanlage). Dieser Schalter legt, wenn er betätigt wird, die Leitung P3 an Masse. Mit folgendem Programm können wir das feststellen:

```
10 PRINT CHR$(147)
20 POKE 64784,8:REM BITMUSTER 00001000
30 IF PEEK(64784)>0THEN30
40 PRINT "KONTAKT AN P3 IST BETAETIGT"
50 VOL 8
60 SOUND1,950,10
70 SOUND1,980,10
80 GOTO 60
```

Sobald der Schalter betätigt wird, meldet das der Computer auf dem Bildschirm und gleichzeitig durch ein akustisches Signal. Der Plus/4 als Alarmanlage? Warum nicht, er könnte so auch während der Nacht sinnvoll eingesetzt werden. Wenn dabei der Fernseher ständig eingeschaltet sein muß, ist die ganze Geschichte natürlich nicht mehr sinnvoll, denn das Gerät bekommt so zusätzlich viele Betriebsstunden, die es unnötig altern lassen. Aber auch dafür gibt es Abhilfe, wir können nämlich dann, wenn der Schalter geschlossen ist, eine andere Ausgangsleitung als Alarmleitung verwenden. Dort schließen wir eine kleine Transistorschaltung an, die uns eventuell über ein Relais eine Lampe oder ein Signalton einschaltet. So lassen sich also am 6529 maximal sieben Eingangsleitungen und eine Alarmausgangsleitung realisieren. Wir könnten sogar noch zwei Eingangsleitungen mehr schalten, denn im Statusregister des 6551 lassen sich die Zustände der Anschlüsse DCD und DSR abfragen.

Der Vollständigkeit halber sei gesagt, daß man sogar zehn Eingangsleitungen und eine Ausgangsleitung am User-Port realisieren kann: Die acht Eingänge des 6529, zusätzlich die zwei Eingänge des 6551 (DCD und DSR) und als Alarmleitung der Anschluß RTS des 6551. Wenn man dann auch noch den User-Port-Anschluß ATN verwendet, könnte man sogar ... Sie sehen schon, mit dem User-Port kann man eine

Menge anfangen. Wir zeigen Ihnen jetzt das Programm für eine Alarmanlage mit zehn Eingängen.

22.2 Der Plus/4 als Alarmanlage

Hier ist zunächst das Programm. Die Bytes $4000 bis $4009 sind Datenbytes.

```
>4000 31 32 33 34 35 36 37 38
>4008 39 30

. 400A  A9 0E     LDA #$0E
. 400C  A2 CE     LDX #$CE
. 400E  78        SEI
. 400F  8D 14 03  STA $0314   ;IRQ-Vektor zurücksetzen
. 4012  8E 15 03  STX $0315
. 4015  58        CLI
. 4016  A9 93     LDA #$93    ;Bildschirm löschen
. 4018  20 4C FF  JSR $FF4C   ;PRINT-Routine
. 401B  A2 00     LDX #$00
. 401D  BD 00 40  LDA $4000,X
. 4020  9D 00 0C  STA $0C00,X ;Zahlen 1 bis 9 und 0 auf
. 4023  E8        INX         ;den Bildschirm
. 4024  E0 0A     CPX #$0A
. 4026  D0 F5     BNE $401D
. 4028  A9 FF     LDA #$FF    ;%11111111 in
. 402A  8D 10 FD  STA $FD10   ;6529
. 402D  A9 00     LDA #$00
. 402F  85 D0     STA $D0
. 4031  85 D1     STA $D1
. 4033  85 D2     STA $D2
. 4035  A9 00     LDA #$00
. 4037  8D 11 FF  STA $FF11   ;Sound-Register im TED
. 403A  A9 FF     LDA #$FF    ;Register des 6529 Exclusiv-ODER
. 403C  4D 10 FD  EOR $FD10   ;verknüpfen
. 403F  85 D0     STA $D0     ;Ergebnis in $D0
. 4041  A9 60     LDA #$60    ;Bit 5 und 6 des Statusregister
. 4043  2D 01 FD  AND $FD01   ;des 6551 abfragen
. 4046  4A        LSR         ;fünfmal nach rechts schieben
. 4047  4A        LSR
. 4048  4A        LSR
. 4049  4A        LSR
. 404A  4A        LSR
. 404B  85 D1     STA $D1     ;Ergebnis in $D1
. 404D  20 11 DB  JSR $DB11   ;Tastaturabfrage
. 4050  D0 B8     BNE $400A   ;Taste gedrückt
. 4052  A5 D0     LDA $D0     ;waren Schalter geschlossen?
. 4054  D0 04     BNE $405A   ;ja
. 4056  A5 D1     LDA $D1     ;
. 4058  F0 E0     BEQ $403A   ;
. 405A  A5 D2     LDA $D2     ;ist Flag für Alarm gesetzt?
```

```
. 405C  D0 1A      BNE $4078   ;ja
. 405E  78         SEI         ;nein. Alarmton einschalten
. 405F  A9 A0      LDA #$A0    ;Anfangsadresse der
. 4061  A2 40      LDX #$40    ;Interruptroutine für
. 4063  8D 14 03   STA $0314   ;Alarmton in IRQ-Vektor
. 4066  8E 15 03   STX $0315   ;eintragen
. 4069  58         CLI
. 406A  A9 C7      LDA #$C7
. 406C  8D 12 FF   STA $FF12   ;Sound-Register des TED
. 406F  A9 1F      LDA #$1F    ;setzen
. 4071  8D 11 FF   STA $FF11
. 4074  A9 01      LDA #$01
. 4076  85 D2      STA $D2
. 4078  A2 FF      LDX #$FF
. 407A  A9 80      LDA #$80
. 407C  E8         INX
. 407D  E0 08      CPX #$08
. 407F  F0 0D      BEQ $408E
. 4081  46 D0      LSR $D0
. 4083  90 F5      BCC $407A
. 4085  1D 00 08 0 RA $0800,X  ;für jeden geschlossenen
. 4088  9D 00 08   STA $0800,X ;Schalter Ziffer auf dem
. 408B  4C 7A 40   JMP $407A   ;Bildschirm blinkend darstellen
. 408E  46 D1      LSR $D1
. 4090  90 06      BCC $4098
. 4092  1D 00 08   ORA $0800,X
. 4095  9D 00 08   STA $0800,X
. 4098  E8         INX
. 4099  E0 0A      CPX #$0A
. 409B  D0 F1      BNE $408E
. 409D  4C 3A 40   JMP $403A   ;zur nächsten Schalterabfrage
. 40A0  A2 1F      LDX #$1F    ;neue Interruptroutine
. 40A2  A9 08      LDA #$08    ;erzeugt Alarmton
. 40A4  4D 02 FD   EOR $FD02
. 40A7  8D 02 FD   STA $FD02
. 40AA  A0 1F      LDY #$1F
. 40AC  88         DEY
. 40AD  D0 FD      BNE $40AC
. 40AF  CA         DEX
. 40B0  D0 F0      BNE $40A2
. 40B2  A9 FF      LDA #$FF
. 40B4  4D 0E FF   EOR $FF0E
. 40B7  8D 0E FF   STA $FF0E
. 40BA  4C 0E CE   JMP $CE0E   ;zur alten Interruptroutine
```

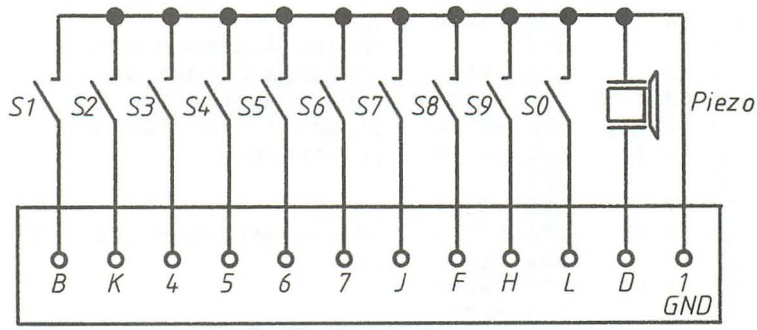


Bild 22.2: *Beschaltung des User-Ports für die Alarmanlage*

Mit dieser Alarmanlage können also zehn Schalter abgefragt werden. Wenn einer der Schalter betätigt wird, ertönt ein Signalton. Außerdem wird auf auf dem Bildschirm die Ziffer des Schalters blinkend dargestellt. Die Ziffer blinkt auch dann weiter, wenn der Schalter wieder geöffnet wird. Auch der Alarmton bleibt eingeschaltet. Sie können so jederzeit kontrollieren, ob Schalter betätigt wurden. Den Alarm ausschalten können Sie durch Drücken einer beliebigen Taste der Tastatur.

Das Besondere an diesem Programm ist, daß Sie den Fernseher auch ausschalten können, denn der Alarmton kann auch am User-Port mittels Lautsprecher abgegriffen werden. Dazu schließen Sie einen Piezolautsprecher (keinen anderen!) am Stift D (RTS) und an Masse des User-Ports an. Dieser Lautsprecher gibt bei einem Alarm einen durchdringenden, Tote aufweckenden (!) Ton von sich. Wenn Ihnen dieser Ton noch nicht laut genug erscheint, können Sie den Lautsprecher auch an der Kassettenrecorderbuchse zwischen Stift 3 und Stift 1 anschließen. Hierbei handelt es sich um die Stromzuführung für den Motor des Kassettenrecorders. Hier liegt eine Spannung von nahezu 9 Volt an. Ein- und ausschalten läßt sich die Spannung mit dem Ausgang P3 des 7501-Mikroprozessors. Dazu müssen Sie Bit 3 des Datenrichtungsregisters (Adresse $0000) auf »1« setzen. Sie können durch Setzen bzw. Löschen des Bit 3 im Datenregister ($0001) die Spannung ein- und ausschalten. Die Piezolautsprecher gibt es recht preiswert im Elektronikfachhandel. Sehr gut geeignet ist ein Mitteltöner.

Das Programm starten Sie mit

SYS 16394

oder in Maschinensprache mit

G 400A

Vorher sollten Sie das Programm mit dem TEDMON abspeichern. Bedenken Sie dabei aber, daß Sie das Programm beginnend ab $ speichern.

Wenn Ihnen die zehn Eingänge dieser Alarmanlage nicht reichen, läßt sich der Plus/4 sehr leicht erweitern. Wir haben dazu einen Ein-/Ausgabechip am Expansionport angeschlossen. Bei unserem Versuchsaufbau haben wir den Typ 6522 verwendet, der 16 Ein- und Ausgabeleitungen hat. Diesen Chip kann man beim Plus/4 z. B. als Modul anschließen (siehe Kapitel 21). Er belegt dann zwar den gesamten Adreßbereich von $8000 bis $BFFF, aber das stört ja niemanden.

Zu unserem Alarmanlagenprogramm ist eigentlich nichts weiter zu sagen, es enthält ja auch genügend Erläuterungen. Zum Alarmton wäre noch anzumerken, daß dieser in einer kurzen Routine erzeugt wird, die bei jedem Interrupt vor der eigentlichen Interruptroutine des Betriebssystems abgearbeitet wird.

22.3 Centronics-Drucker am User-Port

Mittlerweile gibt es auf dem Markt eine unübersehbare Auswahl an Druckern. Wer mit dem Computer intensiv und sinnvoll arbeiten möchte, wird sich früher oder später einen Drucker kaufen, denn was nützt z. B. eine Textverarbeitung, wenn Sie Ihre Texte nicht zu Papier bringen können? Ganz klar, ein Drucker muß her.

Beim Kauf sollten Sie aber genau überlegen, was Sie jetzt und eventuell später mit dem Drucker anfangen möchten. Sie können Sich entweder einen speziellen Commodore-Drucker mit serieller Schnittstelle anschaffen oder einen der zahlreichen Drukker mit Centronics-Schnittstelle (eine parallele Schnittstelle).

Der Plus/4 bietet Ihnen die serielle Schnittstelle, an der z. B. auch die Floppy 1541 angeschlossen ist. Eine Centronics-Schnittstelle kennt der Plus/4 nicht. Sie kann aber mit dem User-Port nachgebildet werden. Dazu muß natürlich jedesmal ein Programm geladen werden. Das ist sicher nachteilig, aber denken Sie doch auch einmal daran, daß Sie diesen Drucker auch später noch brauchen können, wenn Sie eventuell auf einen anderen Computer umsteigen. Z.B. hat der Commodore AMIGA schon eine Centronics-Schnittstelle serienmäßig eingebaut.

Sie müssen natürlich selbst wissen was Sie machen. Wir werden Ihnen jetzt zeigen, wie man den User-Port zur Centronics-Schnittstelle umfunktioniert (leider funktioniert unser Beispielprogramm nicht zusammen mit der im Plus/4 eingebauten Software).

22.4 Was ist Centronics?

Die Centronics-Schnittstelle wurde von der Firma Centronics für ihre Drucker entwickelt. Die Schnittstelle wurde sehr bald zu einer Art Standard und ist mitlerweile für fast alle Drucker zu erhalten bzw. schon in fast allen Druckern eingebaut. Bei der Centronics-Schnittstelle werden die Daten parallel zum Drucker gesendet. Zusätzlich zu den acht Datenleitungen gibt es noch eine Reihe anderer Steuerleitungen, die zusammen auf einem 36poligen Stecker liegen. Für eine funktionierende Centronics-Schnittstelle benötigen wir jedoch nur folgende Leitungen: Acht Datenleitungen, BUSY, -STROBE und Masse.

Wir brauchen zunächst ein Kabel mit mindestens elf Adern, einen User-Port-Stecker und einen Centronics-Stecker. Das bekommen Sie alles in Elektronikläden und kostet zusammen etwa DM 15. Sie können sich auch ein fertiges Kabel für den C64 kaufen, müssen dann allerdings den User-Port-Stecker umlöten. Die Stecker werden folgendermaßen miteinander verbunden:

User-Port-Stecker Stift	Bezeichnung	Centronics-Stecker Stift
A	Masse	16
B	PØ	2
K	P1	3
4	P2	4
5	P3	5
6	P4	6
7	P5	7
J	P6	8
F	P7	9
H	(DCD) BUSY	11
D	(RTS) -STROBE	1

Das Kabel hätten wir damit. Jetzt fehlt nur noch das passende Programm:

```
;INITIALISIERUNG
;
. 065E  A9 00     LDA #$00     ;Ausgangsleitungen
. 0660  8D 10 FD  STA $FD10    ;des 6529 auf '0'
. 0663  A9 08     LDA #$08     ;-STROBE auf '1'
. 0665  8D 02 FD  STA $FD02    ;-STROBE = RTS des 6551
. 0668  A9 75     LDA #$75     ;Anfangsadresse Low des Programms
. 066A  A2 06     LDX #$06     ;       "       High       "
. 066C  78        SEI
. 066D  8D 24 03  STA $0324    ;Anfangsadresse in
. 0670  8E 25 03  STX $0325    ;Ausgabevektor
. 0673  58        CLI
. 0674  60        RTS
;
;CENTRONICS PROGRAMM
;
. 0675  48        PHA          ;Zeichen im Akku auf Stack
. 0676  A5 99     LDA $99      ;Aktuelles Ausgabegerät
. 0678  C9 04     CMP #$04     ;Drucker ist Gerät #4
. 067A  F0 03     BEQ $067F    ;Gerät #4
. 067C  4C 4E EC  JMP $EC4E    ;Kein Drucker - zurück
. 067F  68        PLA          ;Zeichen vom Stack holen
. 0680  8D 10 FD  STA $FD10    ;Zeichen in 6529
. 0683  A9 00     LDA #$00
. 0685  8D 02 FD  STA $FD02    ;-STROBE auf '0'
. 0688  A9 08     LDA #$08
. 068A  8D 02 FD  STA $FD02    ;-STROBE wieder auf '1'
. 068D  A9 20     LDA #$20
. 068F  2C 01 FD  BIT $FD01    ;Ist BUSY noch '1'?
. 0692  F0 FB     BEQ $068F    ;Ja
. 0694  18        CLC          ;Drucker ist fertig,
. 0695  60        RTS          ;also zurück
```

Das Programm starten Sie mit:

SYS 1630

Dann können Sie mit

OPEN 1,4

den Druckerkanal öffnen. Die Ausgabe erfolgt über die Centronics-Schnittstelle. Wenn Sie alles richtig gemacht haben, müßte es funktionieren. Sie können jetzt z. B. dieses Centronics-Programm zu Papier bringen:

CMD 1
MONITOR
D 065E 0695

Mit dem hier abgedruckten Programm steuerte der Plus/4 (mit angeschlossener Floppy 1541) einen EPSON-LX80 und einen Speedy 100-80 einwandfrei an. Andere Drucker konnten leider nicht getestet werden.

Noch einige Worte zur Centronics-Schnittstelle. Wir haben das Programm ja schon mit einigen Kommentaren versehen, möchten jetzt aber noch erklären, wieso das eigentlich funktioniert.

Zunächst müssen die Datenleitungen geschaltet werden, im Programm bei Adresse $0680. Anschließend muß dem Drucker mitgeteilt werden, daß gültige Daten auf den Datenleitungen anliegen. Das geschieht mit -STROBE. -STROBE ist ja ein low-aktives Signal, also muß es auf »0« gesetzt werden. Das Signal muß mindestens 0,5 Mikrosekunden aktiv sein, dann kann es wieder auf »1« gesetzt werden. Nun ist unser Plus/4 nicht so schnell, also können wir im Programm -STROBE unmittelbar hintereinander auf »0« und wieder auf »1« setzen.

Unmittelbar nachdem -STROBE aktiv war, wird vom Drucker das Signal BUSY auf »1« gesetzt. BUSY ist solange »1«, bis der Drucker das Zeichen gedruckt hat und wieder auf ein neues Zeichen warten kann. Im Programm wird die BUSY-Leitung bei Adresse $068F abgefragt.

Übrigens wird vom Plus/4 kein LF (LINE FEED, Zeilenvorschub) gesendet. Sie können bei Ihrem Drucker aber durch Einstellung der Mikroschalter den sogenannten AUTO FEED-Modus einschalten. Dann wird bei jedem Carriage-Return, das der Drucker empfängt, automatisch noch ein Zeilenvorschub gemacht. Wenn sich Ihr Drucker nicht dementsprechend einstellen läßt, müssen Sie das Centronics-Programm ändern. Dann muß jedes Zeichen darauf überprüft werden, ob es sich dabei um ein CR ($0D) handelt. Dann muß nach dem CR noch ein LF ($0A) ausgegeben werden.

Anhang A

Die BASIC-Fehlermeldungen

Der Plus/4 kennt verschiedene Fehlermeldungen, um Ihnen das Auffinden von Fehlern zu erleichtern.

In der Variablen ER steht die Fehlernummer, die Stringvariable ERR$(ER) enthält die Fehlermeldung. Taucht ein Fehler im Direktmodus auf, so wird ERR$ vom Rechner ein Fragezeichen vorangestellt. Befindet sich ein Fehler in einem Programm und wird einem Programmabbruch nicht mit dem Befehl TRAP entgegengetreten, wird der Meldung auch noch die Zeilennummer angehängt. Die Zeilennummer, in der der Fehler auftrat, steht in der Variablen EL. Wir beziehen uns an dieser Stelle auf den Inhalt von ER und ERR$(ER).

ER	Inhalt von ERR$	Fehlerbeschreibung
1	TOO MANY FILES	Mit dem Plus/4 können maximal 10 Files geöffnet werden. Tritt dieser Fehler auf, so schließen Sie ein File mit CLOSE Filenummer.
2	FILE OPEN	Sie dürfen ein bereits geöffnetes File nicht zum zweiten Mal öffnen. Sollte ein neues File geöffnet werden, geben Sie ihm eine andere Filenummer.
3	FILE NOT OPEN	Wenn Sie versuchen aus einer Nicht-Datei zu lesen oder in eine solche zu schreiben, erfolgt diese Meldung. Abhilfe siehe Befehl OPEN.
4	FILE NOT FOUND	Es wurde versucht, ein Programm oder eine Datei zu lesen, die nicht auf der Diskette vorhanden ist. Bei der Datasette bedeutet sie, daß eine Bandendemarkierung gelesen wurde, ohne daß das im Namen genannte Programm oder die Datei gefunden wurde.

5	DEVICE NOT PRESENT	Es sollten Daten an ein Zusatzgerät übermittelt werden, obwohl dieses nicht eingeschaltet oder vorhanden ist.
6	NOT INPUT FILE	Es wurde versucht, aus einer mit Sekundäradresse 1 oder 2 geöffneten Datei zu lesen. Lesen geht nur mit der Sekundäradresse 0. Mehr siehe OPEN.
7	NOT OUTPUT FILE	Es sollte in eine mit der Sekundäradresse 0 geöffnete Datei geschrieben werden. Geht nur bei Sekundäradresse 1 oder 2. Siehe auch OPEN.
8	MISSING FILE NAME	Bei der Diskstation wird immer ein Programm oder Dateiname benötigt.
9	ILLEGAL DEVICE NUMBER	Gerätenummer und Befehl können nicht zusammenarbeiten. Beispiel: SAVE "VERSUCH",4. Vier ist die Gerätenummer für den Drucker.
10	NEXT WITHOUT FOR	Es wurde NEXT angetroffen, ohne daß eine Schleife mit FOR...TO geöffnet wurde, oder dem NEXT einer Schleife folgt ein falscher Variablenname.
11	SYNTAX	Es wurde ein BASIC-Befehl falsch geschrieben, unzulässige Zeichen verwandt oder auch ein Doppelpunkt vergessen.
12	RETURN WITHOUT GOSUB	Das Programm traf auf ein RETURN, ohne daß mit GOSUB in ein Unterprogramm verzweigt wurde.
13	OUT OF DATA	Mit dem READ-Befehl sollten mehr DATA's gelesen werden als vorhanden sind. Vielleicht fehlt ein RESTORE im Programm.

14	ILLEGAL QUANTITY	Die Zahl, die im Argument steht, ist zu groß oder zu klein. Beispiel: POKE 3400,300. Erlaubt wäre maximal 255. Oder POKE 3400,-1 erlaubt wäre minimal 0.
15	OVERFLOW	Es wird mit Zahlen gearbeitet oder es werden Rechenergebnisse erziehlt, die größer als 1.701411833E+38 sind.
16	OUT OF MEMORY	Das Programm kann zu lang sein oder es wurden zu viele Variablen oder zu große Variablen-Arrays definiert. Tritt auch auf, wenn zu viele Schleifen geöffnet, Funktionen definiert oder zu viele Unterprogramme verschachtelt wurden.
17	UNDEF'D STATEMENT	Es wurde mit RUN, GOTO, GOSUB oder RESUME eine Programmzeile angesprochen, die nicht existiert.
18	BAD SUBSCRIPT	Ein Variablenindex, der im Variablenfeld nicht existiert, sollte angesprochen werden.
19	REDIM'D ARRAY	Ein Feld sollte zum zweiten Mal dimensioniert werden. Siehe auch DIM.
20	DIVISION BY ZERO	Eine Division durch 0 ist nicht erlaubt.
21	ILLEGAL DIRECT	Kommt immer dann, wenn Befehle wie zum Beispiel INPUT oder GET nicht im Programm, sondern im Direktmodus verwandt werden.
22	TYPE MISMATCH	Statt Zahlen wurden Strings oder statt Strings Zahlen eingegeben.
23	STRING TOO LONG	Bei der Eingabe von Strings oder Programmzeilen wurde die Länge von 80 Zeichen oder bei einer Stringoperation von 255 Zeichen überschritten.

24	FILE DATA	Es sollten numerische Werte statt Zeichen oder umgekehrt von Disk oder Datasette gelesen werden. Siehe auch TYPE MISMATCH.
25	FORMULA TOO COMPLEX	Ihre Berechnung ist zu kompliziert. Bilden Sie Teilergebnisse, mit denen Sie dann weiterrechnen.
26	CAN'T CONTINUE	Wird ein Programm während BREAK verändert, kann es nicht mehr mit CONT gestartet werden.
27	UNDEF'D FUNCTION	Die aufgerufene Funktion wurde nicht mit DEF FN definiert.
28	VERIFY	Die zu vergleichenden Daten zwischen Speicher und Disk oder Kassette stimmen nicht überein.
29	LOAD	Ein Programm oder eine Datei konnte nicht fehlerfrei geladen werden.
30	BREAK	Das Programm wurde mit STOP oder der Stop-Taste unterbrochen.
31	CAN'T RESUME	Programm traf auf RESUME, ohne daß die Fehlermeldung aufgerufen wurde.
32	LOOP NOT FOUND	
33	LOOP WITHOUT DO	Programm traf auf LOOP oder EXIT, obwohl keine Schleife mit DO geöffnet wurde.
34	DIREKT MODE ONLY	Es sollte ein Befehl abgearbeitet werden, der nur im Direktmodus verwandt werden darf, zum Beispiel DELETE.

35	NO GRAPHICS MODE	Es wurde ein Grafikbefehl gelesen, ohne vorher mit GRAPHIC einen Grafikmodus einzuschalten.
36	BAD DISK	Es trat ein Fehler beim Formatieren einer Diskette auf.

Der Fehlernummer 32 folgt keine Erklärung, weil dieser Fehler bei unseren Arbeiten mit dem Plus/4 nie auftrat.

Anhang B

Disketten-Fehlermeldungen

Tritt beim Arbeiten mit dem Diskettenlaufwerk ein Fehler auf, so zeigt die Floppy das mit dem Blinken der roten LED an. Läßt man die reservierten Variablen DS und DS$ ausdrucken, wird der Fehlerkanal (15) der Floppy ausgelesen. In DS und DS$ steht die Fehlernummer; zusammen in DS$ der Fehlertext, Spur- und Sektornummer der Diskette, bei denen der Fehler aufgetreten ist bzw. sich bemerkbar gemacht hat.

Die Fehlernummern bedeuten im einzelnen:

Nr.	Fehlermeldung	Fehlerbeschreibung
00	OK	Kein Fehler
01	FILES SCRATCHED	Nach einem SCRATCH-Befehl wird hier die Anzahl der gelöschten Files angegeben.
20	READ ERROR	Blockheader nicht gefunden. Ursache: unerlaubte Sektornummer, zerstörter Blockheader, defekte Diskette.
21	READ ERROR	Synchronisationszeichen nicht gefunden. Verursacht durch fehlende oder unformatierte Diskette oder dejustierten Schreib/Lesekopf.
22	READ ERROR	Ein Block sollte gelesen werden, der eine fehlerhafte Prüfsumme im Blockheader aufweist.
23	READ ERROR	Wie 22, jedoch wurde der Block in den DOS-Speicher gelesen.
24	READ ERROR	Wie 23.
25	WRITE ERROR	Ein Block wurde fehlerhaft abgespeichert. Eventuell fehlerhafte Diskette.

Nr.	Fehlermeldung	Fehlerbeschreibung
26	WRITE PROTECTION	Die Diskette enthält einen Schreibschutz. Ein Schreibversuch ergibt diese Fehlermeldung.
27	READ ERROR	Wie 22.
28	WRITE ERROR	Diskette nicht korrekt formatiert. Der Fehler tritt erst beim Beschreiben der Diskette auf.
29	DISK ID MISMATCH	ID einer Diskette stimmt nicht mit der ID im DOS-Speicher überein.
30	SYNTAX ERROR	Ein falscher Befehl wurde eingegeben.
31	SYNTAX ERROR	Wie 30.
32	SYNTAX ERROR	Eine Befehlszeile mit mehr als 40 Zeichen wurde eingegeben.
33	SYNTAX ERROR	Unerlaubter Dateiname, verursacht durch falsche Verwendung der Jokerzeichen.
34	SYNTAX ERROR	Der Filename wurde vom DOS nicht erkannt. Z.B. wurden Befehl und Name nicht mit (:) getrennt.
39	FILE NOT FOUND	Falscher Filename wurde eingegeben.
50	RECORD NOT PRESENT	Es wurde auf einen Eintrag (Record) einer relativen Datei zugegriffen, der noch nicht angelegt ist. Die Fehlermeldung erscheint auch, wenn die Datei erweitert werden soll, kann dann aber ignoriert werden.
51	OVERFLOW IN RECORD	Die maximale Recordlänge wurde überschritten (z.B. gehört das Carriage Return-Zeichen, CHR$(13), zum Record!).

Nr.	Fehlermeldung	Fehlerbeschreibung
52	FILE TOO LARGE	Die angegeben Recordnummer ist zu groß (Diskettenkapazität überschritten).
60	WRITE FILE OPEN	Es wurde versucht, ein nicht geschlossenes File zu öffnen.
61	FILE NOT OPEN	Ein File wurde angesprochen, das vorher nicht geöffnet wurde.
62	FILE NOT FOUND	Das angesprochene File ist auf der Diskette nicht vorhanden.
63	FILE EXISTS	Ein bestehendes File sollte mit gleichem Namen überschrieben werden.
64	FILE TYPE MISMATCH	Es wurde ein falscher Filetyp angegeben.
65	NO BLOCK	Es wurde versucht, einen Block mit dem B-A-Befehl als belegt zu kennzeichnen. Der Block ist aber bereits belegt. In DS$ werden Spur und Sektor des nächsten freien Blocks angegeben (»0«, wenn alle höhernumerierten Blocks belegt sind).
66	ILLEGAL TRACK OR SECTOR	Bei einem Block-Befehl wurde ein nicht vorhandener Block angesprochen.
67	ILLEGAL TRACK OR SECTOR	Es trat eine fehlerhafte Koppeladresse in einem Block auf.
70	NO CHANNEN	Alle Kanäle der Floppy sind belegt.
71	DIR ERROR	BAM und Directory stimmen nicht überein. Abhilfe schafft der Befehl INITIALIZE.

Nr.	Fehlermeldung	Fehlerbeschreibung
72	DISK FULL	Es sind mindestens 662 Blocks belegt oder im Directory sind 144 Einträge (Maximalzahl).
73	DOS MISMATCH	Die Diskette ist mit einem anderen Laufwerk beschrieben worden und kann mit der 1541 oder der 1551 nicht gelesen werden.

Anhang C

Die Abkürzungen der BASIC-Befehle

Die meisten BASIC-Worte lassen sich abkürzen. Beim Programmieren kann man daher Zeit sparen. Durch die Abkürzungen wird es auch ermöglicht, mehr als 80 Zeichen in einer Programmzeile unterzubringen. Wie dies gemacht wird, erfahren Sie im Kapitel über das Programmieren. Der Nebeneffekt bei dieser Programmierung: Es wird auch Speicherplatz gespart, da Zeilennummern eingespart werden. Programmzeilen mit mehr als 80 Zeichen sollten aber die Ausnahme bilden, da sie nicht nur unübersichtlich, sondern auch umständlich zu editieren sind. Nach LIST erscheinen alle abgekürzten Befehlsworte in ihrer richtigen Länge. Beachten Sie bitte, daß in der Abkürzung von TAB und SPC die erste Klammer schon enthalten ist.

Alle Befehle sind im Groß-/Kleinschriftmodus abgedruckt. Großbuchstaben erhalten Sie also über die SHIFT-Taste. Wird im Großschrift-Grafikmodus programmiert, muß ebenfalls die SHIFT-Taste und der entsprechende Buchstabe gedrückt werden. Man erhält dann als letztes Zeichen ein Grafikzeichen.

Name	Abk.	Bezug	Name	Abk.	Bezug
abs	aB	Rechnen	mid$	mI	Strings
and	aN	Logik	monitor	mO	Masch.spr.
asc	aS	Strings	next	nE	Schleifen
atn	aT	Rechnen	not	nO	Logik
auto	aU	Prg.Hilfe	open	oP	Peripherie
backup	bA	Disk	paint	pA	Grafik
box	bO	Grafik	peek	pE	Speicher
char	chA	Grafik/Text	poke	pO	Speicher
chr$	cH	Strings	print	?	Ausgabe
circle	cL	Grafik	print#	pR	Peripherie
close	clO	Disk/Cass	printusing	?usI	Ausgabe
clr	cL	Variablen	pudef	pU	Print Using
cmd	cM	Ausgabeger.	rclr	rC	Grafik
collect	colL	Disk	rdot	rD	Grafik
color	coL	Farbe	read	rE	Data
cont	cO	Prg.Stop	rename	reN	Disk
copy	coP	Disk	renumber	renU	Prg.Hilfe
data	dA	Programm	restore	reS	Data
def	dE	Rechnen	resume	resU	Fehler
delete	deL	Löschen	return	reT	Prg.Sprung
dim	dI	Felder	rgr	rG	Grafik
directory	diR	Disk	right$	rI	Strings
dload	dL	Disk	rlum	rL	Grafik
draw	dR	Grafik	rnd	rN	Zahlen
dsave	dS	Disk	run	rU	Prg.Start
end	eN	Programm	save	sA	Disk/Cass
err$	eR	Variable	scale	scA	Grafik
exp	eX	Rechnen	scnclr	sC	Grafik
for	fO	Schleifen	scratch	scR	Disk
fre	fR	Speicher	sgn	sG	Zahlen
get	gE	Eingabe	sin	sI	Rechnen
getkey	getkE	Eingabe	sound	sO	Musik
get#	gE#	Disk/Cass	spc(	sP	Ausgabe
gosub	goS	Prg.Sprung	sqr	sQ	Rechnen
goto	gO	Prg.Sprung	sshape	sS	Grafik
graphic	gR	Grafik	stop	sT	Programm
gshape	gS	Grafik	str$	stR	Strings
header	heA	Disk	sys	sY	Masch.Prg
hex$	hE	Zahlen	tab(	tA	Ausgabe
input#	iN	Disk/Cass	trap	tR	Fehler
instr	inS	Strings	troff	troF	Fehler
joy	jO	Eingabe	tron	trO	Fehler
key	kE	Tastatur	until	uN	Schleifen
left$	leF	Strings	usr	uS	Masch.Prg
let	lE	Variablen	val	vA	Strings
list	lI	Prg.Hilfe	verify	vE	Disk/Cass
load	lO	Disk/Cass	vol	vO	Musik
locate	loC	Grafik	wait	wA	Speicher
loop	loP	Schleifen	while	wH	Schleifen

Anhang D

Die BASIC-Token

Bevor eine Programmzeile in den Speicher übernommen wird, wird sie nach BASIC-Schlüsselwörtern durchsucht. Diese werden zur Speicherplatzersparnis in Zahlen zwischen 128 und 253 umgewandelt. Diese Zahlen werden Token genannt

Zahl	Befehl	Zahl	Befehl	Zahl	Befehl	Zahl	Befehl
128	END	161	GET	194	PEEK	227	GSHAPE
129	FOR	162	NEW	195	LEN	228	SSHAPE
130	NEXT	163	TAB(	196	STR$	229	DRAW
131	DATA	164	TO	197	VAL	230	LOCATE
132	INPUT#	165	FN	198	ASC	231	COLOR
133	INPUT	166	SPC(	199	CHR$	232	SCNCLR
134	DIM	167	THEN	200	LEFT$	233	SCALE
135	READ	168	NOT	201	RIGHT$	234	HELP
136	LET	169	STEP	202	MID$	235	DO
137	GOTO	170	+	203	GO	236	LOOP
138	RUN	171	-	204	RGR	237	EXIT
139	IF	172	*	205	RCLR	238	DIREKTORY
140	RESTORE	173	/	206	RLUM	239	DSAVE
141	GOSUB	174	^	207	JOY	240	DLOAD
142	RETURN	175	AND	208	RDOT	241	HEADER
143	REM	176	OR	209	ON	242	SCRATCH
144	STOP	177	>	210	HEX$	243	COLLECT
145	ON	178	=	211	ERR$	244	COPY
146	WAIT	179	<	212	INSTR	245	RENAME
147	LOAD	180	SGN	213	ELSE	246	BACKUP
148	SAVE	181	INT	214	RESUME	247	DELETE
149	VERIFY	182	ABS	215	TRAP	248	RENUMBER
150	DEF	183	USR	216	TRON	249	KEY
151	POKE	184	FRE	217	TROFF	250	MONITOR
152	PRINT#	185	POS	218	SOUND	251	USING
153	PRINT	186	SQR	219	VOL	252	UNTIL
154	CONT	187	RND	220	AUTO	253	WHILE
155	LIST	188	LOG	221	PUDEF		
156	CLR	189	EXP	222	GRAPHIC		
157	CMD	190	COS	223	PAINT		
158	SYS	191	SIN	224	CHAR		
159	OPEN	192	TAN	225	BOX		
160	CLOSE	193	ATN	226	CIRCLE		

Anhang E

Bildschirmaufbau

Wir zeigen Ihnen zwei Bilder, die Ihnen den Bildschirmspeicher und den Farbspeicher verdeutlichen sollen. Das erste Bild zeigt den Bildschirmspeicher. Die Zahlen auf der linken Seite geben die Anfangsadresse der jeweiligen Zeile an. Das zweite Bild zeigt den Adreßbereich des Farbspeichers

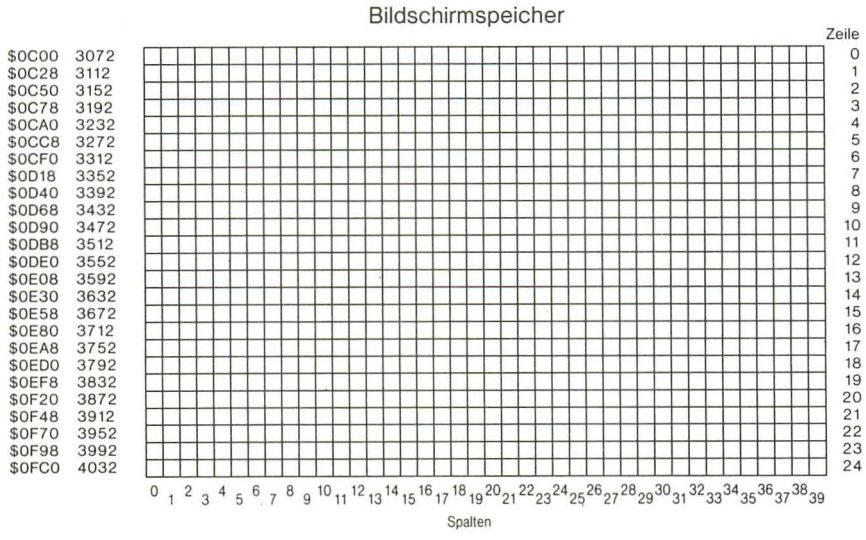


Bild E.1: *Adreßbereich des Bildschirmspeichers*

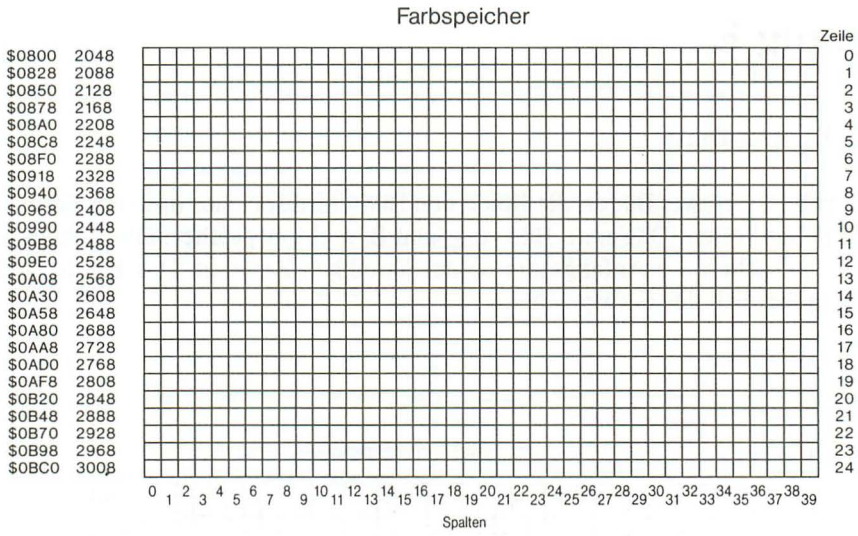


Bild E.2: *Adreßbereich des Farbspeichers*

Anhang F

Die Farbgebung mit COLOR und POKE

Beim COLOR-Befehl muß zuerst die Farbzone bestimmt werden. Es dürfen hier die Zahlen 0 bis 4 auftreten. Sie haben dabei folgende Bedeutung:

Farbzone	Bezeichnung
0	Bildschirmhintergrund
1	Farbe der Zeichen
2	Mehrfarben 1
3	Mehrfarben 2
4	Bildschirmrand

Für die Farben werden Zahlen von 1 bis 16 verwandt. Dabei bedeuten:

Farb-Nr.	Farbe
1	Schwarz
2	Weiß
3	Rot
4	Cyan
5	Purpur
6	Grün
7	Blau
8	Gelb
9	Orange
10	Braun
11	Gelb/Grün
12	Rosa
13	Blau/Grün
14	Hellblau
15	Dunkelblau
16	Hellgrün

Diese Farben können auch noch in der Helligkeit verändert werden. Dies entspricht dem dritten Parameter des COLOR-Befehls. Es dürfen Zahlen von 0 bis 7 benutzt werden. 0 entspricht dem dunkelsten Farbton, 7 dem hellsten. Die anderen Zahlen entsprechen Zwischenwerten.

Wird in den Farbspeicher gepoket, dann sind die Zahlenwerte von COLOR nicht mehr gültig. Die Farben dürfen dann Werte von 0 bis 255 annehmen. Die Helligkeit muß nicht extra gepoket werden, da sie mit dem Farbwert übergeben wird.

Es ergeben sich folgende Farbnummern:

Farb-Nr.	Farbe	Wert für Helligkeitsstufe							
		0	1	2	3	4	5	6	7
1	Schwarz	0	16	32	48	64	80	96	112
2	Weiß	1	17	33	49	65	81	97	113
3	Rot	2	18	34	50	66	82	98	114
4	Cyan	3	19	35	51	67	83	99	115
5	Purpur	4	20	36	52	68	84	100	116
6	Grün	5	21	37	53	69	85	101	117
7	Blau	6	22	38	54	70	86	102	118
8	Gelb	7	23	39	55	71	87	103	119
9	Orange	8	24	40	56	72	88	104	120
10	Braun	9	25	41	57	73	89	105	121
11	Gelb/Grün	10	26	42	58	74	90	106	122
12	Rosa	11	27	43	59	75	91	107	123
13	Blau/Grün	12	28	44	60	76	92	108	124
14	Hellblau	13	29	45	61	77	93	109	125
15	Dunkelblau	14	30	46	62	78	94	110	126
16	Hellgrün	15	31	47	63	79	95	111	127

Schwarz hat nur eine Helligkeitsstufe. Die sich rechnerisch ergebenen Werte wurden der Vollständigkeit halber mit aufgeführt.

Durch Addition der Zahl 128 zur gewünschten Helligkeitsstufe erhält man blinkende Zeichen.

Anhang G

CHR$-Codes

Hier ist die Tabelle mit allen CHR$-Codes Ihres Plus/4. Sie erhalten das entsprechende Zeichen, wenn Sie den Befehl CHR$(x) verwenden. Es handelt sich dabei nicht nur um Zeichen, sondern auch um die sogenannten Steuerzeichen, z.B. Return. Die dargestellten Zeichen erscheinen auf Ihrem Bildschirm so wie in der Tabelle abgebildet, im Großschrift-Grafik-Modus, dem Einschaltzustand. Wenn Sie auf die Klein-/Großschrift umschalten, z. B. mit PRINT CHR$(14) oder durch Drücken der SHIFT/COMMODORE-Taste, erscheinen für die CHR$-Codes 65 bis 90 die entsprechenden Kleinbuchstaben. Für die CHR$-Codes 97 bis 122 werden dann Großbuchstaben dargestellt. Außerdem erscheinen für die Codes 126, 127, 169, 186, 233, 250 und 255 andere Grafikzeichen.

Für das Space (Leertaste) gibt es zwei verschiedene Codes, zunächst das »normale« Space mit CHR$(32) und dann das sogenannte »geschiftete« Space, also das gleichzeitige Drücken der SHIFT- und der Leertaste, mit CHR$(160). Sie können umgekehrt mit dem Befehl ASC(»x«) die CHR$-Codes der gedrückten Tasten erhalten. Das läßt sich sehr gut in Programmen verwenden, wenn einzelne Tasten ausgewertet werden sollen.

Noch eine interessante Möglichkeit für die Tastaturabfrage: Es besteht die Möglichkeit durch Tastenkombinationen der CONTROL-Taste mit einer Buchstabentaste die CHR$-Codes von 1 bis 29 zu bekommen. Auch das läßt sich sinnvoll in Programmen einsetzen. Sie erhalten so eine Menge neuer »Funktionstasten«. Z. B. ergibt CONTROL/K den CHR$-Wert 11. In der folgenden Tabelle zeigen wir Ihnen alle Tastenkombinationen und die dazugehörigen Codes.

Doch zuvor noch etwas: Im sogenannten Anführungszeichen-Modus ist es möglich, die CHR$-Codes von 1 bis 29 durch reverse Buchstaben darzustellen. Sie müssen dazu nur ein Anführungszeichen eintippen und dann die entsprechende Tastenkombination mit CONTROL/Buchstabentaste wählen. Bei den Codes 0, 10, 13, 19, 20 und 27 funktioniert das nicht ganz so einfach. Bei diesen Codes müssen Sie den entsprechenden reversen Buchstaben im Reverse-Modus in Ihr Programmlisting eingeben.

CHR$	Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$	Zeichen
0		32	Space	64	@	96	─
1		33	!	65	A	97	♠
2		34	"	66	B	98	│
3		35	#	67	C	99	─
4		36	$	68	D	100	─
5	Weiß	37	%	69	E	101	─
6		38	&	70	F	102	─
7		39	'	71	G	103	│
8	verr. Sh/Com.	40	(	72	H	104	│
9	entr. Sh/Com.	41	)	73	I	105	╮
10		42	*	74	J	106	╰
11		43	+	75	K	107	╯
12		44	,	76	L	108	╭
13	Return	45	-	77	M	109	╲
14	Kleinschrift	46	.	78	N	110	╱
15		47	/	79	O	111	╭
16		48	0	80	P	112	╮
17	Cursor ab	49	1	81	Q	113	●
18	Reverse ein	50	2	82	R	114	─
19	Home	51	3	83	S	115	♥
20	Del	52	4	84	T	116	│
21		53	5	85	U	117	╯
22		54	6	86	V	118	╳
23		55	7	87	W	119	○
24		56	8	88	X	120	♣
25		57	9	89	Y	121	│
26		58	:	90	Z	122	♦
27	Escape	59	;	91	[	123	┼
28	Rot	60	<	92	£	124	┼
29	Cursor rechts	61	=	93	]	125	│
30	Grün	62	>	94	↑	126	▪
31	Blau	63	?	95	←	127	◥

CHR$	Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$	Zei.
128		160		192		224	
129	Orange	161		193		225	
130	Flash on	162		194		226	
131		163		195		227	
132	Flash off	164		196		228	
133	F1	165		197		229	
134	F3	166		198		230	
135	F5	167		199		231	
136	F7	168		200		232	
137	F2	169		201		233	
138	F4	170		202		234	
139	F6	171		203		235	
140	F8	172		204		236	
141	Shift/Return	173		205		237	
142	Umsth. Großsch.	174		206		238	
143		175		207		239	
144	Schwarz	176		208		240	
145	Cursor hoch	177		209		241	
146	Reverse aus	178		210		242	
147	CLR	179		211		243	
148	INST	180		212		244	
149	Braun	181		213		245	
150	Gelbgrün	182		214		246	
151	Rosa	183		215		247	
152	Blaugrün	184		216		248	
153	Hellblau	185		217		249	
154	Dunkelblau	186		218		250	
155	Hellgrün	187		219		251	
156	Purpur	188		220		252	
157	Cursor links	189		221		253	
158	Gelb	190		222		254	
159	Cyn	191		223		255	

Bild G.1

CHR$	Tasten	Zeichen	Funktion
0			
1	CTRL/A		
2	CTRL/B		
3	CTRL/C		
4	CTRL/D		
5	CTRL/E		Weiß
6	CTRL/F		
7	CTRL/G		
8	CTRL/H		verriegelt Shift/Commodore
9	CTRL/I		
10	CTRL/J		
11	CTRL/K		
12	CTRL/L		
13	CTRL/M		Return
14	CTRL/N		Umschaltung Kleinschrift
15	CTRL/O		
16	CTRL/P		
17	CTRL/Q		Cursor ab
18	CTRL/R		Reverse ein
19	CTRL/S		Home
20	CTRL/T		Del
21	CTRL/U		
22	CTRL/V		
23	CTRL/W		
24	CTRL/X		
25	CTRL/Y		
26	CTRL/Z		
27	CTRL/:		Escape
28	CTRL/£		Rot
29	CTRL/;		Cursor rechts

Bild G.2

Anhang H

Wichtige Speicheradressen des Plus/4

Die ersten 4072 Bytes stehen dem Programmierer für BASIC-Programme nicht zur Verfügung. In diesen Speicherzellen werden vom Betriebssystem selbständig Werte abgespeichert, abgerufen oder verändert. Von BASIC hat man mit POKE die Möglichkeit diese zu manipulieren. Es sollte aber beachtet werden, daß durch falsche Pokes der Rechner abstürzen kann. Oft hilft dann nur noch ein Reset.

HEX	DEZ	Beschreibung
$0000	0	Datenrichtungsregister des 7501
$0001	1	Ein-/Ausgabe-Port des 7501
$0002	2	Flag für Schleifen
$0003-0004	3-4	Neue Startadresse bei Renumber
$0005-0006	5-6	Schrittweite bei Renumber
$0007	7	Gesuchtes Zeichen
$0008	8	Flag für Anführungszeichenmodus
$0009	9	TAB Spaltenzähler
$000A	10	Flag 0=Load, 1=Verify
$000B	11	Zeiger für Eingabepuffer, Zahl der Elemente
$000C	12	Flag für Standart-DIM
$000D	13	Datentyp $FF=String $00=Numerisch
$000E	14	Datentyp $80=Integer $00=Fließkomma
$000F	15	Flag für DATA, LIST
$0010	16	Flag Element, FNx Flag
$0011	17	Flag $00=INPUT, $40=GET, $98=READ
$0012	18	Flag Vorzeichen des ATN
$0013	19	Flag INPUT Prompt (Stop des Programms)
$0014-0015	20-21	Int.Adresse
$0016	22	Zeiger auf temporären Stringstapel
$0017-0018	23-24	Letzter temporärer Stringvektor
$0019-0021	25-33	Stapel für temporäre Strings
$0022-0023	34-35	Bereich für Hilfszeiger 1
$0024-0025	36-37	Bereich für Hilfszeiger 2
$0026-002A	38-42	Bereich für Produkt bei Multiplikationen
$002B-002C	43-44	Zeiger auf BASIC Anfang
$002D-002E	45-46	Zeiger auf Variablen Anfang
$002F-0030	47-48	Zeiger auf Beginn der Arrays
$0031-0032	49-50	Zeiger auf Ende der Arrays +1
$0033-0034	51-52	Zeiger auf Stringspeicher
$0035-0036	53-54	Hilfszeiger für Strings
$0037-0038	55-56	Zeiger auf Speichergrenze
$0039-003A	57-58	Laufende BASIC-Nummer
$003B-003C	59-60	Textpointer
$003D-003E	61-62	Zeiger auf BASIC-Statement für CONT

HEX	DEZ	Beschreibung
$003F-0040	63-64	Nummer der aktuellen DATA-Zeile
$0041-0042	65-66	Adresse des aktuellen DATA-Elements
$0043-0044	67-68	Sprungvektor für INPUT
$0045-0046	69-70	Aktueller Variablenname
$0047-0048	71-72	Adresse der aktuellen Variablen
$0049-004A	73-74	Variablenzeiger für FOR...NEXT
$004B-004C	75-76	Zwischenspeicher für BASIC-Zeiger
$004D	77	Akkumulator für Vergleichssymbole
$004E-0052	78-82	Arbeitsbereich (Zeiger usw.)
$0053	83	Grafik Modus
$0054-0056	84-86	Sprungvektor für Funktionen
$0057-0060	87-96	Bereich für nummerische Operationen
$0061	97	Fließkomma Akkumulator 1: Exponent
$0062-0065	98-101	Fließkomma Akkumulator 1: Mantisse
$0066	102	Fließkomma Akkumulator 1: Vorzeichen
$0067	103	Zeiger für Polynomauswertung
$0068	104	Fließkomma Akkumulator 1: Überlauf
$0069-006E	105-110	Fließkomma Akkumulator 2: Exponent...
$006F	111	Vorzeichenvergleich Akku 1 mit Akku 2
$0070	112	Fließkomma Akku 1 niederwertige Stelle (Rundung)
$0071-0072	113-114	Kassettenpuffer Länge/Zeiger
$0073-0074	115-116	AUTO (Zeileninkrement) 0=AUS
$0075	117	Grafik Flag
$0076-0078	118-120	Arbeitsbereich
$0079-007B	121-123	Darstellung von DS$
$007C-007D	124-125	BASIC Pseudo Stack Pointer
$007E-008F	126-143	Arbeitsbereich (Sound)
$0090	144	Statuswort ST
$0091	145	Flag Stoptaste=127 / RVS Taste
$0094	148	Flag Serieller Bus-Ausgabe Zeichenpuffer
$0095	149	Zeichenspeicher-Serieller Bus
$0096	150	Zwischenspeicher
$0097	151	Anzahl der geöffneten Files
$0098	152	Eingabegerät (Normal 0)
$0099	153	Ausgabegerät (Normal 3)
$009A	154	Flag $80=Direkt- $00=Programmodus
$009B-009C	155-156	Zeiger für Kassettenpuffer/Scrolling
$009D-009E	157-158	Zeiger auf Ende des Programms
$009F-00A0	159-160	Temporärer Datenspeicher
$00A1-00A2	161-162	" "
$00A3-00A5	163-165	Echtzeit Uhr
$00A6	166	Register für seriellen Bus
$00A7	167	Register für Kassettenroutine
$00A8	168	Register für seriellen Bus
$00A9	169	Temporärer Farbvektor
$00AA	170	Register für Kassettenroutine
$00AB	171	Anzahl der Zeichen im Filenamen
$00AC	172	Aktuelle logische Filenummer
$00AD	173	Aktuelle Sekundäradresse

HEX	DEZ	Beschreibung
$00AE	174	Aktuelle Gerätenummer
$00AF-00B0	175-176	Zeiger auf Filenamen
$00B1	177	benutzt Fehlerroutine
$00B2-00B3	178-179	I/O Startadresse
$00B4-00B5	180-181	Basis Ladeadresse
$00B6-00B7	182-183	Zeiger Ladeendadresse
$00BA-00BB	186-187	Zeiger auf Zeichen im Kassettenpuffer
$00BE-00BF	190-191	Register für Long Fetch Routine
$00C0-00C1	192-193	Register für Scrolling
$00C2	194	RVS Flag (Bildschirm)
$00C3	195	Zeiger Ende der Zeile bei INPUT
$00C4-00C5	196-197	Cursorposition (Input), Reihe/Spalte
$00C6	198	Taste, die gedrückt wird ($40=keine Taste)
$00C7	199	Flag Eingabe Bildschirm/Tastatur
$00C8-00C9	200-201	Zeiger auf Bildschirmzeile
$00CA	202	Cursorposition in akt. Bildschirmzeile
$00CB	203	Flag Anführungsz.modus ($00=aus $0F=ein)
$00CC	204	Länge der aktuellen Bildschirmzeile
$00CD	205	Zeile, in der sich Cursor befindet
$00CE	206	Letztes Zeichen (I/O)
$00CF	207	Anzahl der Zeichen (Insertmodus)
$00D0-00E8	208-232	Reserviert für Programmierer und Software
$00E9	233	Arbeitsbereich
$00EA-00EB	234-235	Bildschirmeditor (Farbe)
$00EC-00EE	236-238	Arbeitsbereich (Bildschirm)
$00EF	239	Anzahl der Zeichen im Tastaturpuffer
$00F0	240	Flag für CTRL-S (verschieden 0 Prg. hält)
$00F1-00F2	241-242	Register für Monitor
$00F5	245	Register für Prüfsumme
$00FA	250	Register für X bei Stopptastentest
$00FB	251	Aktuelle Bank Konfiguration
$00FE	254	Arbeitsregister (Editor)
$0100-01FF	256-511	Prozessorstack
$0200-0258	512-600	Eingabepuffer
$0259-025C	601-604	BASIC-Puffer
$025D-02AC	605-684	BASIC-/DOS Arbeitsbereich
$025D	605	DOS Schleifenzähler
$025E-026D	606-621	Bereich für Filenamen
$026E	622	1. Filename (Länge)
$026F	623	DOS (Laufwerk 1)
$0270-0271	624-625	1. Filename (Adresse)
$0272	626	2. Filename (Länge)
$0273	627	DOS (Laufwerk 2)
$0274-0275	628-629	2. Filename (Adresse)
$0276	630	DOS logiche Adresse
$0277	631	DOS (Geräteadresse)
$0278	632	DOS (Sekundäradresse)

HEX	DEZ	Beschreibung
$0279-027A	633-634	DOS (Disketten ID)
$027B	635	ID Flag
$027C	636	DOS (Ausgabepuffer)
$027D-02AC	637-684	DOS (Arbeitsbereich)
$02AD-02B0	685-688	Grafik Cursor
$02B1-02B4	689-692	Grafik Cursor (Register)
$02B5-02CB	693-715	Grafik Register
$02CC-02E3	716-739	PRINT USING, Grafik Arbeitsbereich
$02E4	740	High Byte Adresse des Charakter ROM
$02EB	747	TRACE Flag ($00=AUS, größer $80=EIN)
$02F0	752	Anzahl der Grafikparameter
$02F1	753	Parameter Relativ=1, Absolut=0
$02F2-02F3	754-755	Fließkomma Vektor
$02F4-02F5	756-757	Integer Vektor
$0300-0301	768-769	Sprungvektor: Fehlermeldung ($8686)
$0302-0303	770-771	Sprungvektor: BASIC-Warmstart ($8712)
$0304-0305	772-773	Sprungvektor: Token Generierung ($8956)
$0306-0307	774-775	Sprungvektor: Keywort erzeugen ($8B6E)
$0308-0309	776-777	Sprungvektor: Hauptschleife ($8BD6)
$030A-030B	778-779	Sprungvektor: Eval ($9417)
$030C-030D	780-781	Sprungvektor: Tokengenerierung User ($896A)
$030E-030F	782-783	Sprungvektor: Keywort erzeugen ($8B88)
$0310-0311	784-785	Sprungvektor: User Token bearbeiten ($8C8B)
$0312-0313	786-787	Sprungvektor: Interrupt (Uhr) ($CE42)
$0314-0315	788-789	Sprungvektor: Interrupt ($CE0E)
$0316-0317	790-791	Sprungvektor: Break Interrupt ($F44C)
$0318-0319	792-793	Sprungvektor: Open ($FF5B)
$031A-031B	794-795	Sprungvektor: Close ($EE5D)
$031C-031D	796-797	Sprungvektor: Kanal für Eigabe öffnen ($ED18)
$031E-031F	798-799	Sprungvektor: Kanal für Ausg. öffnen ($ED60)
$0320-0321	800-801	Sprungvektor: I/O zurücksetzen ($EF04)
$0322-0323	802-803	Sprungvektor: Input ($EBE8)
$0324-0325	804-805	Sprungvektor: Ausgabe ($EC4B)
$0326-0327	806-807	Sprungvektor: Abfrage der Stoptaste ($F265)
$0328-0329	808-809	Sprungvektor: Getin Routine ($EBD9)
$032A-032B	810-811	Sprungvektor: Schließen aller Files ($EF08)
$032C-032D	812-813	Sprungvektor: Monitor Break ($F446)
$032E-032F	814-815	Sprungvektor: Lade Routine ($F04A)
$0330-0331	816-817	Sprungvektor: Save Routine ($F1A4)
$0332-03F2	818-1010	Kassettenpuffer
$03F3-03F5	1011-1012	Datenzähler (Write)
$03F5-03F6	1013-1014	Datenzähler (Read)
$03F7-0436	1015-1078	RS-232 Input Puffer (64 Byte)
$0437-0454	1079-1108	Systemarbeitsbereich

HEX	DEZ	Beschreibung
$0455-0472	1109-1138	Systemarbeitsbereich
$0473-0478	1139-1144	Chrget Routine
$0455-0484	1145-1156	Chrgot Routine
$0494-04E6	1172-1254	Bankswitching Routinen
$04E7-04EA	1255-1258	PRINT USING Parameter
$04EF-04F6	1263-1270	Trap und Error Flags
$0500-0502	1280-1282	USR Sprungbefehl
$0503-0507	1283-1287	Startwert für RND
$0508	1288	Flag für Kalt- oder Warmstart
$0509-0512	1289-1298	Tabelle der Logischen Filenummern
$0513-051C	1299-1308	Tabelle der Gerätenummern
$051D-0526	1309-1318	Tabelle der Sekundäradressen
$0527-0530	1319-1328	Tastaturpuffer
$0531-0532	1329-1330	BASIC-Anfang
$0533-0534	1331-1332	BASIC-Ende
$0539	1337	Zeiger Kassettenpuffer
$053A	1338	Typ des Kassettenfiles
$0540	1344	Tastenwdh.$80=alle,$40=keine$0=SPC,DEL, CUR
$0541-0542	1345-1346	Zähler für Tastenwiederholung
$0543	1347	Shift Flag
$0544	1348	Letztes Shift Zeichen
$0545-0546	1349-1350	Sprungvektor: Keylog
$0547	1351	Text/Grafik Flag
$0548	1352	Scroll Flag
$054B-0551	1355-1372	CPU Arbeitsbereich
$0552-0557	1362-1367	CPU Register
$0558-055C	1368-1372	CPU Arbeitsbereich
$055D	1373	Zähler für Funktionstasten
$055E	1374	Zeiger Text Funktionstasten
$055F-05E6	1375-1510	Speicher für Funktionstasten
$05EC-06EB	1516-1571	1. Page für Bankingroutinen
$05EC-05EF	1516-1519	Adreßtabelle
$05F0-05F1	1520-1521	Long Jump (Adresse)
$05F2	1522	Long Jump (Akkumulator)
$05F3	1523	Long Jump (X-Register)
$05F7	1524	Long Jump (Statusregister)
$05F5-065D	1525-1629	RAM Bereich für Bankswitching
$065E-06EB	1630-1771	RAM Bereich für Benutzersoftware
$06EC-07AF	1792-1967	BASIC Pseudo Stack
$07B0-07CC	1968-1996	Kassetten Arbeitsbereich
$07CD-07D0	1997-2000	RS-232 Arbeitsbereich
$07D1	2001	RS-232 Zeiger auf Anfang Eingabepuffer
$07D2	2002	RS-232 Zeiger auf Ende Eingabepuffer
$07D3	2003	Anzahl der Zeichen im Eingabepuffer
$07E5	2021	Bildschirm: unterer Rand
$07E6	2022	Bildschirm: oberer Rand
$07E7	2023	Bildschirm: linker Rand
$07E8	2024	Bildschirm: rechter Rand
$07E9	2025	Scroll $00=Ein, $80=Aus

HEX	DEZ	Beschreibung
$07EA	2026	Automatisches Einfügen 00=Aus,ab 80=Ein
$07EB	2027	Wurde ESC gedrückt? $27=ja
$07EE-07F1	2030-2033	Bit Tabelle
$07F2	2034	A Register retten bei SYS
$07F3	2035	X Register retten bei SYS
$07F4	2036	Y Register retten bei SYS
$07F5	2037	Status Register retten bei SYS
$07F6	2038	Register für Tastaturabfrage
$07F7	2039	Flag CTRL-S $00=offen,$06=gesperrt
$07F8	2040	RAM/ROM Umschaltg Monitor $0=ROM,$80=RAM
$07FC	2044	Motorsteuerung
$0800-0BFF	2048-3071	Farbspeicher (Text)
$0C00-0FE7	3072-4071	Bildschirmspeicher (Text)

Lage des BASIC-RAMs vor Aufruf der Grafik:

$1000-FCFF	4096-64767	BASIC-RAM des Plus/4

Belegung des RAMs nach Aufruf der Grafik:

$4000-FCFF	16384-64767	BASIC-RAM des Plus/4
$1800-1BFF	6144-7167	Luminanztabelle (Grafik)
$1C00-1FFF	7168-8191	Farbtabelle (Grafik)
$2000-3FFF	8192-16383	Grafikbildschirm

Anhang I

Wichtige Register des 7360 (TED)

Im Plus/4 findet ein neuer Chip Verwendung. Es ist ein Baustein mit der Bezeichnung 7360 oder TED (TExt Display). Dieser Chip ist erstaunlich leistungsfähig. Er ist für die Speicherverwaltung (maximal 64 Kbyte), Bild, Ton, Tastatur- und Joystickabfrage zuständig.

Die Basisadresse (die erste Adresse) des TED ist $FF00 (65280). Dieser folgen eine Reihe von Registern, von denen wir hier die wichtigsten beschreiben möchten.

```
$FF00 65280  TIMER 1 Low-Byte
$FF01 65281  TIMER 1 High-Byte
$FF02 65282  TIMER 2 Low-Byte
$FF03 65283  TIMER 2 High-Byte
$FF04 65284  TIMER 3 Low-Byte
$FF05 65285  TIMER 3 High-Byte
$FF06 65286  Bit 0 - 2: Vertikale Position des
             Bildschirminhalts (ermöglicht Verschieben des
             Bildschirms um 8 Punkte)
             Bit 3: 0 = 24 Zeile, 1 = 25 Zeilen
             Bit 4: 0 = Bildschirm aus, 1 = Bildschirm ein
             Bit 5: 0 = HIRES aus, 1 = HIRES ein
             Bit 6: 0 = Extendet-Color-Modus aus, 1 = ECM ein
             Bit 7: ist immer 0
$FF07 65287  Bit 0 - 2: Horizontale Position des Bildschirms
             Bit 3: 0 = 38 Spalten, 1 = 40 Spalten
             Bit 4: 0 = Mehrfarbenmodus aus, 1 = Mehrfarben ein
             Bit 5: 1 = Freeze (Einfrieren) der horz. Position
             Bit 6: 0 = PAL, 1 = NTSC
             Bit 7: 0 = Reverse möglich, 1 = Reverse nicht
             möglich
$FF08 65288  Keyboard-Latch
$FF09 65289  Interrupt-Request-Register
             Bit 0: unbenutzt
             Bit 1: 1 = Rasterinterrupt
             Bit 2: unbenutzt
             Bit 3: 1 = TIMER 1 Unterlauf
             Bit 4: 1 = TIMER 2 Unterlauf
             Bit 5: unbenutzt
             Bit 6: 1 = TIMER 3 Unterlauf
             Bit 7: Interrupt-Flag
```

```
$FF0A 65290  Bit 0: Bit 8 des Registers $FF0B (65291)
             die anderen Bits haben die gleiche Belegung wie
             Register 65289. Bei Übereinstimmung mindestens
             eines Bits wird ein Interrupt (IRQ) ausgelöst.
$FF0B 65291  Bit 0 - 7: Rasterzeile (Bit 8 in Reg. 65290)
$FF0C 65292  Bit 0: Bit 8 des Hardware-Cursors
             Bit 1: Bit 9 des Hardware-Cursors
             Bit 2 - 7 unbenutzt
$FF0D 65293  Bit 0 - 7 des Hardware-Cursors
$FF0E 65294  Bit 0 - 7: Frequenz Oszillator 1
$FF0F 65295  Bit 0 - 7: Frequenz Oszillator 2
$FF10 65296  Bit 0: Bit 8 der Frequenz Osz. 2
             Bit 1: Bit 9 der Frequenz Osz. 2
             Bit 2 - 7 unbenutzt
$FF11 65297  Bit 0 - 3: Lautstärke
             Bit 4: Stimme 1 ein/aus
             Bit 5: Stimme 2 ein/aus
             Bit 6: Rauschen (Stimme 2) ein/aus
             Bit 7: 1 = Einschaltzustand der Oszillatoren
             wiederherstellen
$FF12 65298  Bit 0: Bit 8 der Frequenz Osz. 1
             Bit 1: Bit 9 der Frequenz Osz. 1
             Bit 2: 1 = Daten aus ROM
                    0 = Daten aus RAM
             Bit 3 - 5: Basisadresse Bit Map
             Bit 6 - 7 unbenutzt
$FF13 65299  Bit 0: Status, 1 = ROM
             Bit 1: Single Clock
             Bit 2 - 7: Basisadresse des Zeichengenerators
$FF14 65300  Bit 0 - 2 unbenutzt
             Bit 3 - 7: Basisadresse für Bild- und Farbspeicher
$FF15 65301  Hintergrundfarbe 0
             Bit 0 - 3: Farbe
             Bit 4 - 6: Helligkeit
             Bit 7: unbenutzt
$FF16 65302  Hintergrundfarbe 1
             Belegung wie Reg. $FF15 (65301)
$FF17 65303  Hintergrundfarbe 2
             Belegung wie Reg. $FF15 (65301)
$FF18 65304  Hintergrundfarbe 3
             Belegung wie Reg. $FF15 (65301)
$FF19 65305  Hintergrundfarbe 4 (Rahmenfarbe)
             Belegung wie Reg. $FF15 (65301)
```

Anhang K

Anschlußbelegungen

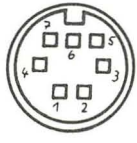
Der Plus/4 hat eine Reihe von Anschlüssen für externe Geräte. Wir zeigen Ihnen hier nun die Steckerbelegung folgender Anschlußbuchsen:

- **Datasette**
- **Audio/Video**
- **Joystick**
- **Serielle Schnittstelle**
- **Erweiterungs-Port**
- **User-Port**

Datasette

Stift	Belegung
1	Masse
2	+ 5V
3	Motor
4	Read
5	Write
6	Schalter
7	Masse

Datasette



Audio/Video

Stift	Belegung
1	Helligkeit
2	Masse
3	Audio Ausgang
4	Video Ausgang
5	Audio Eingang
6	Farbe Ausgang
7	+ 5V
8	frei

Audio / Video

Joystick 1

Stift	Belegung
1	Key 0
2	Key 1
3	Key 2
4	Key 3
5	+ 5V
6	Feuerknopf, Key 6
7	Masse
8	D2

Joystick 2

Stift	Belegung
1	Key 0
2	Key 1
3	Key 2
4	Key 3
5	+ 5V
6	Feuerknopf, Key 7
7	Masse
8	D1

Seriell

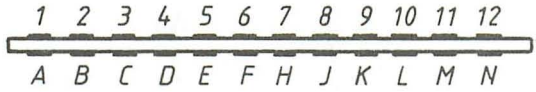
Stift	Belegung
1	Service Request
2	Masse
3	ATN (Attention)
4	Clock
5	DATA
6	RESET

Erweiterungs-Port

Stift	Belegung	Stift	Belegung
1	Masse	A	Masse
2	+ 5V	B	C1 Low
3	+ 5V	C	BRESET
4	IRQ	D	RAS
5	R/W	E	I0
6	C1 High	F	A15
7	C2 Low	H	A14
8	C2 High	J	A13
9	CS1	K	A12
10	CS0	L	A11
11	CAS	M	A10
12	MUX	N	A9
13	BA	P	A8
14	D7	R	A7
15	D6	S	A6
16	D5	T	A5
17	D4	U	A4
18	D3	V	A3
19	D2	W	A2
20	D1	X	A1
21	D0	Y	A0
22	AEC	Z	frei
23	Extern Audio	AA	frei
24	02	BB	frei
25	Masse	CC	Masse

User-Port

Stift	Belegung	Stift	Belegung
1	Masse	A	Masse
2	+ 5 Volt	B	P0
3	BRESET	C	RxD
4	P2	D	RTS
5	P3	E	DTR
6	P4	F	P7
7	P5	H	DCD
8	RxC	J	P6
9	ATN	K	P1
10	9 Volt AC	L	DSR
11	9 Volt AC	M	TxD
12	Masse	N	Masse



Anhang L

Umwandlungstabelle für HEX-, Dezimal- und Binärzahlen

Mit dieser Tabelle können Sie Zahlen der drei gebräuchlichsten Zahlensysteme sehr schnell in ein anderes Zahlensystem umsetzen. Bei Binärzahlen steht an der äußersten rechten Stelle Bit 0. Links davon steht Bit 1, Bit 2,.....bis Bit 7.

	HEX	BINÄR	DEZ	HEX	BINÄR	DEZ	HEX	BINÄR
0	$00	%00000000	32	$20	%00100000	64	$40	%01000000
1	$01	%00000001	33	$21	%00100001	65	$41	%01000001
2	$02	%00000010	34	$22	%00100010	66	$42	%01000010
3	$03	%00000011	35	$23	%00100011	67	$43	%01000011
4	$04	%00000100	36	$24	%00100100	68	$44	%01000100
5	$05	%00000101	37	$25	%00100101	69	$45	%01000101
6	$06	%00000110	38	$26	%00100110	70	$46	%01000110
7	$07	%00000111	39	$27	%00100111	71	$47	%01000111
8	$08	%00001000	40	$28	%00101000	72	$48	%01001000
9	$09	%00001001	41	$29	%00101001	73	$49	%01001001
10	$0A	%00001010	42	$2A	%00101010	74	$4A	%01001010
11	$0B	%00001011	43	$2B	%00101011	75	$4B	%01001011
12	$0C	%00001100	44	$2C	%00101100	76	$4C	%01001100
13	$0D	%00001101	45	$2D	%00101101	77	$4D	%01001101
14	$0E	%00001110	46	$2E	%00101110	78	$4E	%01001110
15	$0F	%00001111	47	$2F	%00101111	79	$4F	%01001111
16	$10	%00010000	48	$30	%00110000	80	$50	%01010000
17	$11	%00010001	49	$31	%00110001	81	$51	%01010001
18	$12	%00010010	50	$32	%00110010	82	$52	%01010010
19	$13	%00010011	51	$33	%00110011	83	$53	%01010011
20	$14	%00010100	52	$34	%00110100	84	$54	%01010100
21	$15	%00010101	53	$35	%00110101	85	$55	%01010101
22	$16	%00010110	54	$36	%00110110	86	$56	%01010110
23	$17	%00010111	55	$37	%00110111	87	$57	%01010111
24	$18	%00011000	56	$38	%00111000	88	$58	%01011000
25	$19	%00011001	57	$39	%00111001	89	$59	%01011001
26	$1A	%00011010	58	$3A	%00111010	90	$5A	%01011010
27	$1B	%00011011	59	$3B	%00111011	91	$5B	%01011011
28	$1C	%00011100	60	$3C	%00111100	92	$5C	%01011100
29	$1D	%00011101	61	$3D	%00111101	93	$5D	%01011101
30	$1E	%00011110	62	$3E	%00111110	94	$5E	%01011110
31	$1F	%00011111	63	$3F	%00111111	95	$5F	%01011111
96	$60	%01100000	138	$8A	%10001010	180	$B4	%10110100
97	$61	%01100001	139	$8B	%10001011	181	$B5	%10110101
98	$62	%01100010	140	$8C	%10001100	182	$B6	%10110110
99	$63	%01100011	141	$8D	%10001101	183	$B7	%10110111
100	$64	%01100100	142	$8E	%10001110	184	$B8	%10111000
101	$65	%01100101	143	$8F	%10001111	185	$B9	%10111001

DEZ	HEX	BINÄR	DEZ	HEX	BINÄR	DEZ	HEX	BINÄR
102	$66	%01100110	144	$90	%10010000	186	$BA	%10111010
103	$67	%01100111	145	$91	%10010001	187	$BB	%10111011
104	$68	%01101000	146	$92	%10010010	188	$BC	%10111100
105	$69	%01101001	147	$93	%10010011	189	$BD	%10111101
106	$6A	%01101010	148	$94	%10010100	190	$BE	%10111110
107	$6B	%01101011	149	$95	%10010101	191	$BF	%10111111
108	$6C	%01101100	150	$96	%10010110	192	$C0	%11000000
109	$6D	%01101101	151	$97	%10010111	193	$C1	%11000001
110	$6E	%01101110	152	$98	%10011000	194	$C2	%11000010
111	$6F	%01101111	153	$99	%10011001	195	$C3	%11000011
112	$70	%01110000	154	$9A	%10011010	196	$C4	%11000100
113	$71	%01110001	155	$9B	%10011011	197	$C5	%11000101
114	$72	%01110010	156	$9C	%10011100	198	$C6	%11000110
115	$73	%01110011	157	$9D	%10011101	199	$C7	%11000111
116	$74	%01110100	158	$9E	%10011110	200	$C8	%11001000
117	$75	%01110101	159	$9F	%10011111	201	$C9	%11001001
118	$76	%01110110	160	$A0	%10100000	202	$CA	%11001010
119	$77	%01110111	161	$A1	%10100001	203	$CB	%11001011
120	$78	%01111000	162	$A2	%10100010	204	$CC	%11001100
121	$79	%01111001	163	$A3	%10100011	205	$CD	%11001101
122	$7A	%01111010	164	$A4	%10100100	206	$CE	%11001110
123	$7B	%01111011	165	$A5	%10100101	207	$CF	%11001111
124	$7C	%01111100	166	$A6	%10100110	208	$D0	%11010000
125	$7D	%01111101	167	$A7	%10100111	209	$D1	%11010001
126	$7E	%01111110	168	$A8	%10101000	210	$D2	%11010010
127	$7F	%01111111	169	$A9	%10101001	211	$D3	%11010011
128	$80	%10000000	170	$AA	%10101010	212	$D4	%11010100
129	$81	%10000001	171	$AB	%10101011	213	$D5	%11010101
130	$82	%10000010	172	$AC	%10101100	214	$D6	%11010110
131	$83	%10000011	173	$AD	%10101101	215	$D7	%11010111
132	$84	%10000100	174	$AE	%10101110	216	$D8	%11011000
133	$85	%10000101	175	$AF	%10101111	217	$D9	%11011001
134	$86	%10000110	176	$B0	%10110000	218	$DA	%11011010
135	$87	%10000111	177	$B1	%10110001	219	$DB	%11011011
136	$88	%10001000	178	$B2	%10110010	220	$DC	%11011100
137	$89	%10001001	179	$B3	%10110011	221	$DD	%11011101
222	$DE	%11011110	234	$EA	%11101010	246	$F6	%11110110
223	$DF	%11011111	235	$EB	%11101011	247	$F7	%11110111
224	$E0	%11100000	236	$EC	%11101100	248	$F8	%11111000
225	$E1	%11100001	237	$ED	%11101101	249	$F9	%11111001
226	$E2	%11100010	238	$EE	%11101110	250	$FA	%11111010
227	$E3	%11100011	239	$EF	%11101111	251	$FB	%11111011
228	$E4	%11100100	240	$F0	%11110000	252	$FC	%11111100
229	$E5	%11100101	241	$F1	%11110001	253	$FD	%11111101
230	$E6	%11100110	242	$F2	%11110010	254	$FE	%11111110
231	$E7	%11100111	243	$F3	%11110011	255	$FF	%11111111
232	$E8	%11101000	244	$F4	%11110100			
233	$E9	%11101001	245	$F5	%11110101			

Anhang M

Maschinensprachebefehle des 7501

In der Tabelle benutzte Abkürzungen:

unm	unmittelbare Adressierung
abs	absolute Adressierung
Z	Zero-Page-Adressierung
Accu	Akkumulatorbezogene Adressierung
impl	implizierte Adressierung
indX	indiziert indirekte Adressierung
indY	indirekt indizierte Adressierung
Z,X	Zero-Page-X-Register indizierte Adressierung
ab,X	absolut-X-Register indizierte Adressierung
ab,Y	absolut-Y-Register indizierte Adressierung
rel	relative Adressierung
ind	indirekte Adressierung
Z,Y	Zero-Page-Y-Register indizierte Adressierung
A	Akkumulator
M	Memory
S	Stapelzeiger
Ms	Stapelspeicher
SR	Statusregister

Bef./Wirkg.	unm	abs	Z	Accu	impl	indX	indY	Z,X	ab,X	ab,Y	rel	ind	Z,Y	NZCIDV
Transportbefehle														
LDA M in Accu	A9	AD	A5			A1	B1	B5	BD	B9				xx
STA Accu in M		8D	85			81	91	95	9D	99				
LDX M in X	A2	AE	A6							BE			B6	xx
STX X in M		8E	86										96	
LDY M in Y	AØ	AC	A4					B4	BC					xx
STY Y in M		8C	84					94						
TAX Accu in X					AA									xx
TAY Accu in Y					A8									xx
TXA X in Accu					8A									xx
TYA Y in Accu					98									xx
TXS X in S					9A									
TSX S in X					BA									xx
PLA Ms in Accu					68									xx
PHA Accu in Ms					48									
PLP Ms in SR					28									von Ms
PHP S in Ms					Ø8									
Statusregisterbefehle														
CLC C-Flag = Ø					18									Ø
SEC C-Flag = 1					38									1
CLD D-Flag = Ø					D8									Ø
SED D-Flag = 1					F8									1
CLI I-Flag = Ø					58									Ø
SEI I-Flag = 1					78									1
CLV V-Flag = Ø					B8									Ø
Arithmetische Befehle														
ADC A=A+M+C	69	6D	65			61	71	75	7D	79				xxx x
SBC A=A-M-C	E9	ED	E5			E1	F1	F5	FD	F9				xxx x
INC M=M-1		EE	E6					F6	FE					xx
DEC M=M+1		CE	D6					D6	DE					xx
INX X=X+1					E8									xx
DEX X=X-1					CA									xx
INY Y=Y+1					C8									xx
DEY Y=Y-1					88									xx
Logische Befehle														
AND A=A UND M	29	2D	25			21	31	35	3D	39				xx
ORA A=A OR M	Ø9	ØD	Ø5			Ø1	11	15	1D	19				xx
EOR A=A EOR M	49	4D	45			41	51	55	5D	59				xx

Bef./Wirkg.	unm	abs	Z	Accu	impl	indX	indY	Z,X	ab,X	ab,Y	rel	ind	Z,Y	NZCIDV
Verzweigungsbefehle														
BCC Verzw. bei C=Ø											9Ø			
BCS Verzw. bei C=1											BØ			
BNE Verzw. bei Z=Ø											DØ			
BEQ Verzw. bei Z=1											FØ			
BPL Verzw. bei N=Ø											1Ø			
BMI Verzw. bei N=1											3Ø			
BVC Verzw. bei V=Ø											5Ø			
BVS Verzw. bei V=1											7Ø			
Sprungbefehle														
JMP	4C											6C		
JSR	2Ø													
RTS Return					6Ø									
RTI Ret. (v. Int.)					4Ø									
Vergleichsbefehle														
CMP A mit M	C9	CD	C5			C1	D1	D5	DD	D9				xxx
CPX X mit M	EØ	EC	E4											xxx
CPY Y mit M	CØ	CC	C4											xxx
BIT A UND M		2C	24											7x 6
Schiebebefehle														
ASL		ØE	Ø6	ØA				16	1E					xxx
LSR		4E	46	4A				56	5E					xxx
ROL		2E	26	2A				36	3E					xxx
ROR		6E	66	6A				76	7E					xxx
Sonstige Befehle														
NOP No Operation					EA									
BRK Softw.-Int.					ØØ									1

Anhang N

Kleines Computerlexikon

In der Computertechnik (und auch in unserem Buch) erscheinen viele spezielle Begriffe, mit denen ein Anfänger zunächst Schwierigkeiten haben wird. Wir haben uns bemüht, die meisten Begriffe und Bezeichnungen in unserem Buch verständlich zu erläutern.

Dieses Buch soll Ihnen auch ein umfangreiches Nachschlagewerk sein. Deshalb bringen wir in dieser Anlage die wichtigsten Begriffe der »Computerei« in alphabetischer Reihenfolge.

Adresse
Die Speicherplätze des Computers sind durchnumeriert. Jeder Speicherplatz bekommt dadurch eine Adresse. Die Höchstanzahl der Adressen richtet sich nach der Anzahl der Leitungen des Adreßbus.

alphanumerisch
Buchstaben und Zahlen, also alle Zeichen.

Arbeitsspeicher
der Teil des Speichers, der aus RAM-Speicherbauteilen besteht. Der Plus/4 hat 64 Kbyte Arbeitsspeicher.

ASCII
Abkürzung für American Standart Code for Information Interchange. Genormter Code für die Zeichendarstellung (z.B. mit CHR$). Beim Plus/4 stimmt dieser Code nicht ganz mit der Tabelle überein, was evtl. zu Schwierigkeiten führen kann, wenn ein Drucker angeschlossen wird, der nicht für Commodore-Computer gedacht ist.

Assembler
Damit bezeichnet man die Maschinensprache und Programme, die den Assemblercode in Maschinensprache umwandeln. Siehe TEDMON.

BASIC
Abkürzung für Beginner's All-Purpose Symbolic Instruction Code. Einfache Programmiersprache für Anfänger. Die BASIC-Dialekte sind im Laufe der Jahre so sehr verbessert worden, daß sie inzwischen allen Ansprüchen gerecht werden.

Baud
Die Schrittgeschwindigkeit bei der seriellen Übertragung von Daten. Baud wird angegeben in Bit pro Sekunde.

Betriebssystem
Die Systemprogramme, die das Arbeiten mit dem Computer möglich machen. Also Zeicheneingabe, Zeichenausgabe, Bedienung der Peripheriegeräte usw.

Bit
Die kleinste im Computer darstellbare Einheit. Abkürzung von Binary Digit = Binärziffer.

Bus
Das Leitungssystem zur Informationsübertragung zwischen den Baugruppen eines Computers. Es gibt den Adreß-, den Daten- und den Steuerbus.

Byte
Ein Byte besteht aus acht Bits. Das Byte wird in der Regel als Hexadezimalzahl dargestellt. 1 Kbyte sind 1024 Byte, also 2^10 Byte. 1 MByte (Megabyte) sind 1024 KByte = 1048576 Byte.

Centronics-Schnittstelle
Schnittstelle zur parallelen Zeichenübertragung zu Druckern.

Character Generator
Der Zeichengenerator des Computers. Dort wird angegeben, wie die Zeichen auf dem Bildschirm dargestellt werden. Beim Plus/4 liegt dieser Zeichengenerator im Bereich von $D000 bis $D7FF.

Compiler
Das Übersetzungsprogramm, das ein in einer höheren Programmiersprache geschriebenes Programm in ein Maschinenprogramm übersetzt. Zur Ausführung des Programms wird der Compiler im Gegensatz zum Interpreter nicht benötigt.

CPU
Central Processing Unit, übersetzt Zentraleinheit. Bei Home- und Personal-Computern ist die CPU der Mikroprozessor.

Datei
Zusammengehörige Daten auf einem Datenträger nennt man Datei oder File.

DFÜ
Datenfernübertragung, z. B. mittels Akustikkoppler über das Telefonnetz.

Directory
Inhaltsverzeichnis einer Diskette.

Disassembler
Programm, das ein Maschinenprogramm in die Assemblerschreibweise (mnemonische Schreibweise, z. B. LDA statt $A9) übersetzt.

DOS
Disk Operating System. Das Betriebssystem der Disketten- oder Festplatten-Laufwerke.

Editor
Der Plus/4 besitzt einen bildschirmorientierten Editor, d.h., man kann den Cursor mit den Cursortasten auf dem Bildschirm bewegen und Daten eingeben bzw. berichtigen.

EPROM
Erasable Programmable Read Only Memory. Programmierbarer Nur-Lese-Speicher, der auch wieder gelöscht werden kann (durch Bestrahlung mit ultraviolettem Licht).

File
Siehe Datei.

Hardcopy
Ein auf Papier gebrachter Ausdruck des Bildschirminhalts (Text- oder Grafik-Bildschirm).

Hardware
Alle Bauteile, die zum Computersystem gehören, nennt man Hardware.

Hexadezimal
Das Zahlensystem zur Basis 16. Hexadezimalzahlen werden in der Computertechnik hauptsächlich zur Darstellung der Maschinensprachebefehle benutzt. Eine Hexadezimalziffer stellt vier Bit dar.

Integer-Variable
Ganzzahlige Variable, gekennzeichnet mit dem Prozentzeichen, z. B. A%. Diese Variablen können Zahlen von -32768 bis +32767 beinhalten.

Interpreter
Programm, das ein in einer höheren Programmiersprache geschriebenes Programm in Maschinensprache übersetzt. Dabei wird Befehl für Befehl übersetzt und sofort ausgeführt.

Interrupt
Unterbrechung des normalen Programmablaufs, um ein sogenanntes Interruptprogramm auszuführen. Anschließend wird das unterbrochene Programm fortgeführt. Beim Plus/4 findet jede 60stel Sekunde eine Programmunterbrechung statt. Dann wird z. B. die Tastatur abgefragt.

Klammeraffe
Dieses Zeichen (@) ermöglicht beim Plus/4 das Überschreiben von schon bestehenden Programmen auf Kassette oder Diskette mit gleichem Namen.

Konstante
Fest im Computer gespeicherter Wert, z. B. PI (3,14159265).

Maschinensprache
Die Programmiersprache, die der Mikroprozessor direkt verstehen und ausführen kann.

Operationscode
Operationscode oder Objectcode ist die Bezeichnung für einen Maschinensprachebefehl.

Parallel-Schnittstelle
Hierbei werden alle Bits eines Zeichens (in der Regel acht Bit) gleichzeitig übertragen. D. h., für jedes Bit steht eine eigene Leitung zur Verfügung.

Parameter
Zusätzliche Information zu einem Befehl, die diesen näher spezifiziert.

Peripherie
Alle Geräte, die an einen Computer angeschlossen werden können, aber nicht unmittelbar Bestandteil des Computers selbst sind (Bildschirm, Drucker, Disketten-Laufwerk Modem usw.).

Pufferspeicher
Zwischenspeicher für Daten. Der Plus/4 besitzt z.B. einen Tastaturpuffer und einen Kassettenpuffer.

RAM
Random Access Memory. Der Arbeitsspeicher des Computers. Dieser Speicher kann beschrieben und ausgelesen werden. Der Speicherinhalt wird beim Ausschalten des Computers gelöscht.

RESET-Taste
Durch Drücken dieser Taste wird der Computer wieder in den Einschaltzustand versetzt.

ROM
Read Only Memory. Übersetzt: Nur-Lese-Speicher. In diesem Speicher ist das Betriebssystem, der BASIC-Interpreter und der Character-Generator gespeichert, beim Plus/4 zusätzlich die »eingebaute« Software.

RS-232
Bezeichnung für eine serielle Schnittstelle nach dem RS232C-Standard. Beim Plus/4 steht diese Schnittstelle am User-Port zur Verfügung (allerdings nur mit TTL-Pegel, also 5 Volt und 0 Volt).

Seriell
Datenverarbeitung oder Datenübertragung, bei dem die Daten nacheinander, also Bit für Bit, übertragen werden.

Software
Mit Software bezeichnet man alle Programme für einen Computer. Software und Hardware zusammen ergeben ein funktionstüchtiges Computersystem.

String
Ein String ist eine Folge alphanumerischer Zeichen. String-Variablen bekommen zur Erkennung ein Dollarzeichen ($) in den Variablennamen.

Stichwortverzeichnis

Weitere Fachbücher aus unserem Verlagsprogramm

COMMODORE 16/116

W. Besenthal/J. Muus

Alles über den C16

Juli 1986, 292 Seiten

Dieses Buch ist ein Lern- und Nachschlagewerk für jeden Commodore-Anwender. Es ist übersichtlich gegliedert und enthält alle Informationen, die für die praktische Arbeit am Computer notwendig sind: BASIC-Kurs mit Beispielen, Strukturiertes Programmieren, Dateiverwaltung, Grafikprogrammierung, Tips & Tricks.

Best.-Nr. MT 90385, ISBN 3-89090-385-1
(sFr. 35,90/öS 304,20) **DM 39,-**

COMMODORE 64

F. Ende

Das große Spielebuch - Commodore 64

1984, 141 Seiten

46 Spielprogramme · Wissenswertes über Programmiertechnik · praxisnahe Hinweise zur Grafikherstellung · alles über Joystick- und Paddleansteuerung · das Spielebuch mit Lerneffekt.

Best.-Nr. MT 603, ISBN 3-922120-63-6
(sFr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 604 (Beispiele auf Diskette)
(sFr. 38,-/öS 342,-) **DM 38,-***
* inkl. MwSt. Unverbindliche Preisempfehlung.

S. Krute

Grafik & Musik auf dem Commodore 64

1984, 336 Seiten

68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.

Best.-Nr. MT 743, ISBN 3-89090-033-X
(sFr. 35,-/öS 296,40) **DM 38,-**

H. L. Schneider/W. Eberl

Das C64-Profihandbuch

1985, 413 Seiten

Ein Buch, das alle wichtigen Informationen für professionelle Anwendungen mit dem C64 enthält. Mit allgemeinen Algorithmen, die auch auf andere Rechner übertragbar sind, und vielen Utilities, getrennt nach BASIC- und Maschinenprogrammen. Besonders nützlich: erweiterte PEEK- und POKE-Funktionen.

Best.-Nr. MT 749, ISBN 3-89090-110-7
(sFr. 47,80/öS 405,60) **DM 52,-**

W.-J. Becker/M. Folprecht

Programmieren unter CP/M mit dem C64

1985, 290 Seiten

Wenn Sie wissen wollen, wie das Betriebssystem CP/M-2.2 auf dem C64 implementiert ist, außerdem einiges über Turbo-Pascal, Nevada-Fortran, MBASIC-80 erfahren wollen, dann ist dieses Buch genau richtig für Sie! Mit Schaltplänen zur eigenen Fertigung des CP/M-Moduls. Für eingefleischte C64-Profis.

Best.-Nr. MT 751, ISBN 3-89090-091-7
(sFr. 47,80/öS 405,60) **DM 52,-**

J. Mihalik

35 ausgesuchte Spiele für Ihren Commodore 64

1984, 141 Seiten

Programmieren Sie selbst 35 faszinierende Spiele · geschrieben in Commodore-64-BASIC · mit Farbe, Grafiken und Ton · Vorschläge zur Programmabwandlung · für kreative Computerfans, die ihre Programmierkenntnisse vertiefen wollen!

Best.-Nr. MT 774, ISBN 3-89090-064-X
(sFr. 23,-/öS 193,40) **DM 24,80**

W. Kassera/F. Kassera

C64 - Programmieren in Maschinensprache

1985, 327 Seiten inklusive Beispieldiskette

In diesem Buch finden Sie über 100 Beispiele zur Assembler-Programmierung mit viel Kommentar und Hintergrundinformationen: Das Schreiben von Maschinenprogrammen · Rechnen und Texten mit vorhandenen Routinen · Bedienung von Drucker und Floppy · Wie man BASIC- und Maschinenprogramme verknüpft · Erstellen von eigenen Befehlen in Modulform. Für Profis!

Best.-Nr. MT 830, ISBN 3-89090-168-9
(sFr. 47,80/öS 405,60) **DM 52,-**

P. W. Dennis/G. Minter

Spiele für den Commodore 64

1984, 196 Seiten

Bewährte alte und raffinierte neue Spiele für Ihren Commodore 64 · klar und übersichtlich gegliederte Programme im Commodore-BASIC · Sie lernen: wie man Unterprogramme einsetzt · eine Tabelle aufbauen und verarbeiten · Programme testen · mit vielen Programmiertricks · für Anfänger.

Best.-Nr. MT 90074, ISBN 3-89090-074-7
(sFr. 23,-/öS 193,40) **DM 24,80**
Best.-Nr. MT 795 (Beispiele auf Diskette)
(sFr. 38,-/öS 342,-) **DM 38,-***
* inkl. MwSt. Unverbindliche Preisempfehlung.

K. Schramm

Die Floppy 1541

1985, 434 Seiten

Für alle Programmierer, die mehr über ihre VC-1541-Floppystation erfahren wollen. Der Vorgang des Formatierens · das Schreiben von Files auf Diskette · die Funktionsweise von schnellen Kopier- und Ladeprogrammen · viele fertige Programme · Lesen und Beschreiben von defekten Disketten · Für Einsteiger und für fortgeschrittene Maschinensprache-Programmierer.

Best.-Nr. MT 90098, ISBN 3-89090-098-4
(sFr. 45,10/öS 382,20) **DM 49,-**
Best.-Nr. MT 710 (Beispiele auf Diskette)
(sFr. 29,90/öS 269,10) **DM 29,90***
* inkl. MwSt. Unverbindliche Preisempfehlung.

Die angegebenen Preise sind Ladenpreise

607326/1

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

S. Baloui

C 64-Fischertechnik: Messen, Steuern, Regeln

Februar 1986, 174 Seiten

Ziel dieses Buches ist es, jedem Besitzer eines Commodore 64/VC 20 eine neue Welt zu erschließen: Die Welt der Roboter, der computergesteuerten Fertigungsstraßen. Alles, was Sie benötigen, ist einer der beiden genannten Computer und der Fischertechnik-Computing-Baukasten mit dazugehörigem Interface.

Best.-Nr. MT 90194, ISBN 3-89090-194-8
(sFr. 27,60/öS 233,20) **DM 29,90**

F. Matthes

Pascal mit dem C 64

Juni 1986,
215 Seiten inklusive Diskette

Buch und Compiler ermöglichen jedem Besitzer eines C 64 den Einstieg in die moderne Programmiersprache Pascal. Dem Anfänger wird ein Einführungskurs in Pascal geboten, wobei viele überschaubare Beispiele aus der Praxis und Übungsaufgaben zum aktiven Lernen mit dem C 64 auffordern.

Für den Pascal-Profi gibt es neben nützlichen Beispielprogrammen ein spezielles Kapitel mit Tips und Tricks.

Der Compiler akzeptiert den gesamten Sprachumfang mit einigen Erweiterungen. Übersetzte Programme laufen ohne weitere Hilfsprogramme auf jedem C 64, nutzen den gesamten Programmspeicher des C 64 und sind 3–4mal schneller als vergleichbare Programme in BASIC.

- Dem Buch liegt ein leistungsfähiges PASCAL-SYSTEM mit einigen Pascal-Programmen auf Diskette bei.

Best.-Nr. MT 90222, ISBN 3-89090-222-7
(sFr. 47,80/öS 405,60) **DM 52,-**

M. Hegenbarth/R. Trierscheid

BASIC-Grundkurs mit dem C 64

1985, 377 Seiten

Der Computerneuling kann mit diesem Buch lernen, mit seinem C 64 in BASIC zu arbeiten, und wird auf die Besonderheiten seines Computers hingewiesen. Dabei müssen nicht unendlich viele und umfangreiche Beispielprogramme mühsam abgetippt werden; es ist sogar denkbar, die Kapitel erst durchzulesen und das Gelernte dann am Computer auszuprobieren. Erwähnenswert ist auch ein Kapitel, welches die Kommunikation zweier C 64 beschreibt, und der Anhang, in dem neben der Kurzbeschreibung der reservierten Worte des BASIC V2 (mit Beispielen) eine Liste nützlicher PEEKs, POKEs und SYS und noch vieles mehr enthalten ist.

Best.-Nr. MT 90361, ISBN 3-89090-361-4
(sFr. 40,50/öS 343,20) **DM 44,-**

H. Ponnath

C 64: Wunderland der Grafik

1985, 232 Seiten inklusive Beispieldiskette

Dieses Buch zeigt eine Vielzahl sehr interessanter Lösungen, um die grafischen Möglichkeiten des Commodore 64 optimal zu nutzen. Als Krönung enthält es ein zuschaltbares Assemblerprogramm, das umfangreiche grafische und einige neue BASIC-Befehle anbietet. Im zweiten Teil des Buches wird eine Möglichkeit gezeigt, wie man bis zu 70 verschiedene Farben erzeugen kann. Viele Beispielprogramme begleiten die Reise durch das Wunderland der Grafik.

Best.-Nr. MT 90363, ISBN 3-89090-363-0
(sFr. 45,10/öS 382,20) **DM 49,-**

Commodore Sachbuch

Alles über den C 64

Juli 1986, 514 Seiten

Das umfangreiche Grundlagenbuch für den Commodore 64, ein nützliches Werkzeug, damit das künftige Programmieren auch Spaß macht: BASIC-Lexikon mit allen Befehlen, Anweisungen und Funktionen in alphabetischer Reihenfolge – Programmierung in Maschinensprache und Einbindung von Maschinensprache-Routinen in BASIC-Programme – Bestandteil des Betriebssystems: Das Kernal – Ein- und Ausgabeprogrammierung von SPRITES und Sonderzeichen – Erzeugung von Laufbildern in hochauflösender Farbgrafik – Musiksynthese und Klangeffekte – Betriebssystem CP/M sowie weitere anspruchsvolle Sprachen – GEOS.

Best.-Nr. MT 90379, ISBN 3-89090-379-7
(sFr. 54,30/öS 460,20) **DM 59,-**

R. West

C-64/SX-64-Computer-Handbuch

1985, 688 Seiten

Das Buch reicht von den professionellen Aspekten der BASIC-Programmierung (Entwicklung klarer und strukturierter Problemlösungen und/oder effizienter Programme) über sehr systemnahe Informationen (Änderungen am eingebauten BASIC, am Betriebssystem etc.) bis hin zur Hardware (Schnittstellen, Kassettengeräte, Floppy) und allen Fragen, die damit zusammenhängen. Besonders wichtig bei dieser Fülle an Informationen: der klare Aufbau des Buches, der den schnellen Zugriff auf die benötigte Information garantiert und so das Buch zur idealen Arbeitsgrundlage macht.

- Eine Enzyklopädie der Profi-Programmierung auf dem C 64.

Best.-Nr. PW 80324, ISBN 3-921803-24-1
(sFr. 60,70/öS 514,80) **DM 66,-**

R. Valentine

C-64-Programmsammlung 50 Lehr-, Spiel- und Nutzprogramme

1985, 200 Seiten

Praxisorientierte Programme und interessante Tips für den 64-User, der schon Erfahrungen mit seinem Computer gesammelt hat und sein Wissen (und auch seine Programmsammlung) erweitern möchte. PEEK, POKE, Bit- und Bytemanipulationen werden an ebenso leicht verständlichen Beispielen erklärt wie die Verwendung der eingebauten Zeituhr und der Sound- und Grafikfeatures Ihres C 64. Abgerundet wird die ganze Sache durch ein kleines Datenverwaltungsprogramm, einen Pilot-Interpreter (!) und viele Spiele. Sämtliche Programme sind in BASIC geschrieben und gut erklärt – somit auch leicht eigenen Anforderungen anzupassen.

Best.-Nr. PW 80346, ISBN 3-921803-46-2
(sFr. 27,50/öS 232,40) **DM 29,80**

Die angegebenen Preise sind Ladenpreise

607326/2

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

Reparaturanleitung Computer: C64 - Technische Servicedaten für Ihren Computer

Einzigartige Serviceunterlagen für Reparaturen und Entwicklungsarbeiten am C64. Enthält Schaltpläne, Bauteile- und Vergleichstypenliste; Prüfpunkte mit Oszillogrammen der Signalformen, Logiktabellen, Spannungsangaben; schnelle Servicetests, Anleitung zur systematischen Fehlersuche.

Best.-Nr. PW 80355, ISBN 3-921803-55-1
(sFr. 27,50/öS 232,40) **DM 29,80**

COMMODORE 128/128 D

G. Jürgensmeier

WordStar für den Commodore 128 PC

1985, 435 Seiten

WordStar ist ein umfangreiches und leistungsfähiges Textverarbeitungsprogramm und damit sicherlich zu Recht das meistverkaufte Programm seiner Art. Doch bedeutet dies nicht unbedingt, daß es auch einfach zu bedienen ist. Hier setzt dieses Buch an: Es macht in vorbildlicher Weise mit allen Möglichkeiten von WordStar und MailMerge vertraut und ist damit eine ideale Ergänzung zum Handbuch. Es versammelt alle wichtigen Informationen für den effektiven Einsatz dieser Programme auf dem Commodore 128 PC.

Best.-Nr. MT 780, ISBN 3-89090-181-6
(sFr. 45,10/öS 382,20) **DM 49,-**

Dr. P. Albrecht

Multiplan für den Commodore 128 PC

1985, 226 Seiten

Multiplan wurde ursprünglich für das 16-Bit-Betriebssystem MS-DOS entwickelt. Inzwischen ist aber auch die in diesem Buch beschriebene CP/M-Version für den Commodore 128 PC auf dem Markt, die den vollen Leistungsumfang der 16-Bit-Version enthält.
Das vorliegende Buch soll eine praktische Einführung in den Umgang mit Multiplan auf dem Commodore 128 PC geben. Anhand von praxisnahen Beispielen werden alle Befehle und Funktionen in der Reihenfolge beschrieben, die der Arbeit in der Praxis entspricht. Bereits nach Abschluß des ersten Kapitels werden Sie in der Lage sein, eigene kleine Multiplan-Anwendungen zu realisieren.

Best.-Nr. MT 836, ISBN 3-89090-187-5
(sFr. 45,10/öS 382,20) **DM 49,-**

Dr. P. Albrecht

dBASE II für den Commodore 128 PC

1985, 280 Seiten

Das vorliegende Buch gibt nach einer kurzen Einführung in den Komplex »Datenbanken« eine Anleitung für den praktischen Umgang mit dBASE II. Schon nach Beherrschung weniger Befehle ist der Anwender in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integrierter Reportgenerator, der im Dialog mit dem Benutzer Berichte gestaltet und in Tabellenform ausdruckt.

Best.-Nr. MT 838, ISBN 3-89090-189-1
(sFr. 45,10/öS 382,20) **DM 49,-**

H. Haberl

Mini-CAD mit Hi-Eddi plus auf dem C64/C128

1985, 230 Seiten inklusive Diskette

Neben den »Standardbefehlen« zum Setzen und Löschen von Punkten, dem Zeichnen von Linien, Kreisen und Rechtecken sowie dem Ausfüllen unregelmäßiger Flächen und dem Verschieben und Duplizieren von Bildschirmbereichen bietet Hi-Eddi eine Reihe von Besonderheiten, die dieses Programm von anderen Grafikprogrammen abheben: Bis zu sieben Grafikbildschirme stehen gleichzeitig zur Verfügung; es besteht die Möglichkeit, Text in die Grafik einzufügen, die Bildschirme zu verknüpfen oder in schneller Folge durchzuschalten.

Best.-Nr. MT 90136, ISBN 3-89090-136-0
(sFr. 44,20/öS 374,40) **DM 48,-**

J. Hückstädt

BASIC 7.0 auf dem Commodore 128

1985, 239 Seiten

Ganz gleich, ob Sie bereits über Programmierkenntnisse verfügen oder nicht, dieses Buch wird Ihnen helfen, den größtmöglichen Nutzen aus dem leistungsstarken BASIC 7.0 des Commodore 128 PC zu ziehen. Sie eignen sich bei der Durcharbeitung dieses Buches alle notwendigen Kenntnisse an, um immer anspruchsvollere Aufgabenstellungen zu bewältigen: Listenverarbeitung, indexsequentielle Dateiverwaltung, Grafikdarstellungen und Sounderzeugung. Ein unentbehrliches Lehrbuch, das sich auch für den geübten Anwender als Nachschlagewerk eignet.

Best.-Nr. MT 90149, ISBN 3-89090-149-2
(sFr. 47,80/öS 405,60) **DM 52,-**

K. Schramm

Die Floppy 1570/1571

Juni 1986, 470 Seiten

Dieses Buch soll es sowohl dem Einsteiger als auch dem fortgeschrittenen Programmierer ermöglichen, die vielfältigen Möglichkeiten dieses neuen Gerätes voll auszuschöpfen. Sämtliche Betriebsarten und Diskettenformate werden ausführlich erläutert. Anhand vieler Beispiele werden Sie in die Dateiverwaltung mit dieser Floppy eingeführt. Der Benutzer lernt die zahlreichen Systembefehle kennen und erfährt zugleich wichtige Grundlagen für das Arbeiten mit dem Betriebssystem CP/M.

Best.-Nr. MT 90185, ISBN 3-89090-185-9
(sFr. 47,80/öS 405,60) **DM 52,-**

P. Rosenbeck

Das Commodore-128-Handbuch

1985, 383 Seiten

In diesem Buch finden Sie einen Querschnitt durch alle wichtigen Funktions- und Anwendungsbereiche des Commodore 128. Sie werden mit dem C64/C128-Modus und der Benutzung von CP/M-3.0 vertraut gemacht, erfahren alles über die Grafik- und Soundmöglichkeiten des C128, lernen die Techniken der Speicherverwaltung und das Banking kennen und werden in die Programmierung mit Assemblersprache sowie die Grafikprogrammierung des 80-Zeichen-Bildschirms eingeführt. Ein umfassendes Handbuch, das Sie immer griffbereit haben sollten!

Best.-Nr. MT 90195, ISBN 3-89090-195-6
(sFr. 47,80/öS 405,60) **DM 52,-**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Wilhelm Besenthal
Jens Muus

Alles über den Plus 4

Die Autoren:

WILHELM BESENTHAL, geboren 1958, beschäftigt sich seit 1980 mit der Maschinensprache des 6502. Er begann 1981 mit der Programmierung in BASIC. Sein Interesse für Commodore-Computer wurde erweckt, als 1982 der Commodore 64 erschien. Heute ist er Experte für C 64, C 16 und Plus/4, da er auf jahrelange Arbeit mit diesen Rechern zurückblicken kann.

JENS MUUS, geboren 1956, begann seine Arbeit mit Computern ebenfalls 1980 mit einem programmierbaren Taschenrechner, dem HP 41 CV. Seit 1981 programmiert er in BASIC. Zunächst arbeitete er mit einem VC20, stieg dann aber, nachdem der Commodore 64 erschien, auf diesen Computer um. Da er bald an die Grenzen der Programmierung mit BASIC stieß, befaßte er sich sehr schnell mit der Maschinensprache des 6502. Jens Muus erarbeitete sich umfassende Kenntnisse in der Interrupttechnik dieser Computer.

1984 brachte Commodore den neuentwickelten Plus/4 auf den Markt. Der Computer hat seither eine weite Verbreitung gefunden.
Dieses Buch enthält alle Informationen, die für das Arbeiten mit dem Plus/4 in BASIC und Maschinensprache erforderlich sind. Zunächst wird das hervorragende BASIC, Version 3.5, des Plus/4 anhand von vielen Beispielen genauestens erläutert. Anschließend bildet ein kompletter Kurs über die Maschinensprache des Plus/4 den zweiten Hauptteil des Buches. Neben einer ausführlichen Befehlsbeschreibung enthält dieser Teil auch eingehende Erläuterungen der Interruptprogrammierung und des Bankswitching. Zahlreiche nützliche und genau beschriebene Beispielprogramme erleichtern den Einstieg in die Maschinensprache.
Ein weiterer wichtiger Abschnitt ist der Hardware des Plus/4 gewidmet. Beschrieben wird die Belegung aller Anschlüsse und die Programmierung des User-Ports. Konkrete Beispiele sind: der User-Port als Centronics-Schnittstelle, der Anschluß externer Speicher und der Plus/4 als Alarmanlage.
Dieses Buch ist für den fortgeschrittenen Programmierer dank der Ausführungen über Hardware und Maschinensprache ebenso interessant, wie der BASIC-Kurs für den unerfahrenen Anwender. Die Anhänge mit wichtigen Informationen, beispielsweise über den Bildschirmaufbau und vielen Tabellen, z. B. Umwandlung für HEX-, Dezimal- und Binärzahlen, machen das Buch zum umfangreichen Nachschlagewerk.

Aus dem Inhalt:

- Kompletter BASIC-Kurs
- Strukturiertes Programmieren
- Fehlerbehandlung
- Maschinensprache-Kurs
- Hardware-Erweiterungen
- Viele interessante Beispielprogramme zum Abtippen

Hardware-Anforderungen:
Commodore Plus/4, Datasette oder Diskettenlaufwerk 1551, 1541, 1570, 1571

ISB N 3-89090-410-6

Markt&Technik

4 001057 904101

DM 39,-
sFr. 35,90
öS 304,20