Commodore
Sachbuch

Wilhelm Besenthal
Jens Muus

Alles über den C16

Lernen und Nachschlagen

BASIC-Kurs mit Beispielen ★ Strukturiertes Programmieren
★ Dateiverwaltung ★ Grafikprogrammierung
★ Tips & Tricks

Alles über den C16

Commodore Sachbuch

Alles über den C16

Lernen und Nachschlagen

- BASIC-Kurs mit Beispielen
- Strukturiertes Programmieren
- Dateiverwaltung
- Grafikprogrammierung
- Tips&Tricks

Wilhelm Besenthal
Jens Muus

Markt&Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Besenthal, Wilhelm:
Alles über den C16 : lernen u. nachschlagen ; BASIC-Kurs mit Beispielen ; strukturiertes Programmieren ; Dateiverwaltung ; Grafikprogrammierung ; Tips&Tricks / Wilhelm Besenthal ; Jens Muus. –
Haar bei München : Markt-und-Technik-Verlag, 1986.
ISBN 3-89090-385-1
NE: Muus, Jens:

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

ISBN 3-89090-385-1

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder, Gersthofen
Printed in Germany

Inhaltsverzeichnis

Verzeichnis der Abbildungen

Vorwort

Dieses Buch ist für den Programmieranfänger und für den fortgeschrittenen Programmierer gedacht, der mehr über den Commodore 16 bzw. Commodore 116 erfahren möchte.

Die zum Rechner gelieferte Anleitung ist, wie so oft, denkbar kurz gehalten, und es bleiben viele Fragen offen. Wir, die Autoren, haben uns bemüht, Schwächen dieses Systems und der Anleitung, die bei einem Rechner dieser Preisklasse zwangsläufig vorhanden sind, aufzuzeigen und diese nach Möglichkeit zu beseitigen.

Zur Gliederung dieses Buches:

Jedes Kapitel ist in sich abgeschlossen. Sie können also, wenn Sie das Kapitel »Grafik« interessiert, gleich dieses Kapitel aufschlagen und es durcharbeiten.

Lesen Sie aber das Buch von Anfang an, so wird es Ihnen eine große Hilfe beim Erlernen der Programmiersprache BASIC sein. Wir zeigen Ihnen, wie man Programme erstellt und erklären alle wichtigen BASIC-Befehle des C16. Im Kapitel »Tastatur« finden Sie eine genaue Beschreibung der ESC-Taste.

Da der C16 über excellente Möglichkeiten verfügt, farbige Grafiken einfach und schnell darzustellen, handelt hiervon das längste Kapitel. Sie finden eine ausführliche Beschreibung der Grafik-Befehle. Unterstützt wird alles durch genau beschriebene Programmbeispiele.

Einige Bemerkungen zu den Programmen:

Alle in diesem Buch abgedruckten Programme sind bewußt sehr ausführlich beschrieben worden, um einerseits dem Anfänger das Verstehen, und andererseits dem Anwender Änderungen zu ermöglichen. Obwohl Sie in diesem

Buch Maschinensprachroutinen finden, wurde auf eine Beschreibung der Maschinensprachbefehle verzichtet. Diese hätte den Rahmen des Buches gesprengt. Die Autoren verweisen an dieser Stelle auf die reichlich vorhandene Literatur über den Mikroprozessor 6502, von dem sich ein kompatibler Typ in Ihrem Rechner befindet.

Da im C16 ein Monitorprogramm (für die Programmierung in Maschinensprache) bereits eingebaut ist, haben wir dieses Programm natürlich auch beschrieben. Alle Programme, die Sie in diesem Buch finden, sind auf dem C16 und dem C116 lauffähig und wurden auf beiden Rechnern getestet. Sollten wider Erwarten Schwierigkeiten auftreten, dann überprüfen Sie bitte erst einmal das Programm, das Sie eingetippt haben, bevor Sie auf die Autoren schimpfen.

So, und nun wünschen wir Ihnen viel Spaß beim Durchlesen des Buches.

Uelzen, März 1986

Wilhelm Besenthal Jens Muus

Vorwort zur zweiten Auflage

Commodore 16 und Commodore 116 sind zur Zeit sehr gefragte Einsteigermodelle. Der günstige Preis und die sehr guten Leistungsmerkmale machen sich in den Verkaufszahlen stark bemerkbar.

Inzwischen werden C16 und C116 zum Teil auch schon mit 64KByte RAM verkauft. Das kommt der Kompatibilität zugute und macht diese Rechner noch attraktiver.

Abschließend bedanken wir uns bei allen, die uns zur Verwirklichung dieses Buches tatkräftig unterstützt haben. Insbesondere danken wir Frau Baumann vom Markt & Technik Verlag für die freundliche Unterstützung.

Uelzen, Juli 1986

Wilhelm Besenthal Jens Muus

1
Commodore 16 und Commodore 116

Mit dem C16 bzw. C116 hat der schon fast legendäre VC-20 überaus leistungsstarke Nachfolger bekommen. Beim C16 wurde die sehr gute Tastatur des VC-20 und des C64 übernommen. Der C116 ist dagegen nur mit einer »Radiergummi-Tastatur« ausgerüstet, die doch sehr gewöhnungsbedürftig ist und ein »blindes« Schreiben fast unmöglich macht. C16/116 sind kompatibel zum Plus/4, haben aber leider nur 16 KByte RAM. Die Rechner zeichnen sich durch hervorragende Grafikeigenschaften aus. Die hochauflösende Grafik wird durch starke BASIC-Befehle unterstützt (VC-20- und C64- Anwender würden vor Neid erblassen). Das neue BASIC 3.5 beinhaltet aber nicht nur Grafik-Befehle, sondern auch Befehle zur Tonerzeugung und für die Arbeit mit Diskettenlaufwerken. Strukturierte Programmierung ist ebenso möglich wie Hilfe bei der Programmerstellung durch RENUMBER, AUTO usw..

Leider fehlt dem C16 und dem C116 der so nützliche User-Port. Damit ist die Verbindung zur Außenwelt stark beeinträchtigt. Man ist sicher davon ausgegangen, daß ein Anfänger, denn dieser Anwenderkreis wird diese Rechner überwiegend nutzen, mit einem User-Port nicht viel anfangen kann. Schade ist es trotzdem, war es doch durch diese Schnittstelle möglich, diverse Ein- und Ausgabegeräte zu betreiben.

Die Programme anderer Commodore-Rechner, die in BASIC geschrieben wurden und keine Pokes, SYS- und WAIT-Befehle enthalten, sind auf dem C16/116 lauffähig. Die einzige Einschränkung dabei ist der zu kleine Speicher des C16/116. Es werden schon mehrere Speichererweiterungen angeboten.

Zur Tastatur ist zu bemerken, daß diese keine **RESTORE**-Taste besitzt. Dafür haben diese neuen Rechner eine **RESET**-Taste, die es ermöglicht, den »abgestürzten« Rechner wieder zum Leben zu erwecken. Notwendig wurde diese Taste auch durch den vorhandenen Maschinensprachmonitor.

Hinzugekommen ist die **ESC**-Taste, die für Bildschirmfenster gebraucht wird und umfangreiche Editierfunktionen ermöglicht.

Erstaunlich ist, daß Commodore die Anschlußbuchsen für die Datasette und für die Joysticks beim C16 geändert hat. Das Ergebnis ist, daß Adapter angeboten werden, um z.B. die große Zahl der auf dem Markt befindlichen Joysticks benutzen zu können. Im Anhang finden Sie einen Belegungsplan der Anschlußbuchsen. Als Bezugsquelle für C16/116-Zubehör können wir Ihnen die Firma Metz Electronic, Schuhstr.11, 3110 Uelzen 1, nennen.

Geht es aber um das Programmieren, dann zeigt der C16 seine volle Stärke. Mit Erscheinen dieser neuen Rechner ging man bei Commodore einen ganz anderen Weg, denn es wurde endlich das »Minimal-BASIC« der Rechner VC-20/C64 abgeschafft und das komfortable BASIC 3.5 eingeführt. Gleichzeitig wurde auch ein Maschinensprachmonitor implementiert.

Noch ein paar Worte zur Speichergröße

Von Haus aus wird der Rechner »nur« mit 16 KByte RAM ausgestattet. Nach dem Einschalten des Grafikmodus schrumpft der für Programme verfügbare Speicherbereich auf 2 KByte zusammen. Dies reicht für den Anfänger sicherlich aus, hat man aber schon ein wenig Erfahrung im Programmieren und schreibt längere Programme, dann wird der Speicherbereich sicher bald verbraucht sein.

Wenn nichts mehr hilft, kann man sich ja eine Speichererweiterung zulegen, die allerdings nicht in das Gehäuse des C116 paßt. Eine Erweiterung auf 64 KByte kostet ca. 200 DM.

Es folgt eine zeichnerische Darstellung des Speicherbereiches. Auf der linken Seite stehen die Adressen in dezimaler Schreibweise und rechts in hexadezimaler Schreibweise.

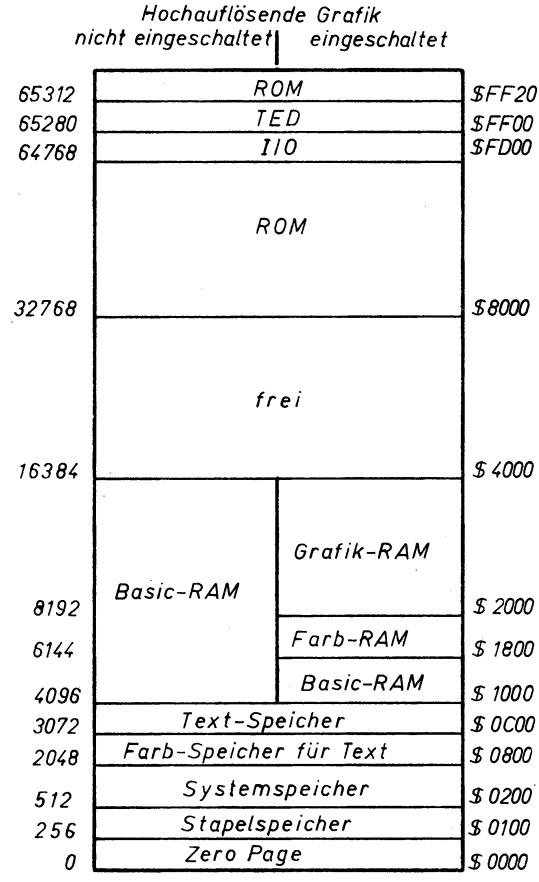

Hochauflösende Grafik
nicht eingeschaltet
eingeschaltet
65312
ROM
$FF20
65280
TED
$FF00
64768
I/O
$FD00
ROM
32768
$8000
frei
16384
$4000
Basic-RAM
Grafik-RAM
8192
$2000
6144
Farb-RAM
$1800
Basic-RAM
4096
$1000
3072
Text-Speicher
$0C00
2048
Farb-Speicher für Text
$0800
512
Systemspeicher
$0200
256
Stapelspeicher
$0100
0
Zero Page
$0000

Bild 1.1

2
Der Aufbau eines Computers

Ihnen als Computer-Neuling steht nun ein schwarzer Kasten gegenüber. Das Ding nennt sich Commodore 16, mehr wissen Sie noch nicht darüber. Vielleicht meinen Sie, wer Programmieren lernen möchte, braucht nicht zu wissen, was in einem Computer vor sich geht. Das ist aber falsch, Sie müssen zumindest den grundsätzlichen Aufbau Ihres Computers kennen. Sie müssen wissen, worum es geht, wenn Ihnen etwas von einem »RAM« oder vom »Betriebssystem« oder vom »Farbspeicher« gesagt wird.

Keine Angst, jetzt kommt keine hochtechnische Beschreibung des Computerinnenlebens. Der Grundaufbau eines Computers ist viel einfacher, als man denkt: 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 vorhergesagte aber noch etwas genauer ansehen, denn mit unseren Bezeichnungen (Nur-Lese-Speicher usw.) können Sie sicher noch nicht viel anfangen. Wir müssen uns also noch über die Zusammenarbeit der einzelnen Baugruppen untereinander und über die genauen Bezeichnungen der Teile unterhalten.

Wir haben dazu eine Zeichnung angefertigt, die Ihnen das Innenleben Ihres Commodore 16 grob verdeutlichen soll.

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 Betriebssystem und das BASIC gespeichert.

CPU Central Processing Unit, der Mikroprozessor

Im C16 befindet sich eine CPU vom Typ 7501. Die 7501 ist kompatibel zum Typ 6502.

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 an.

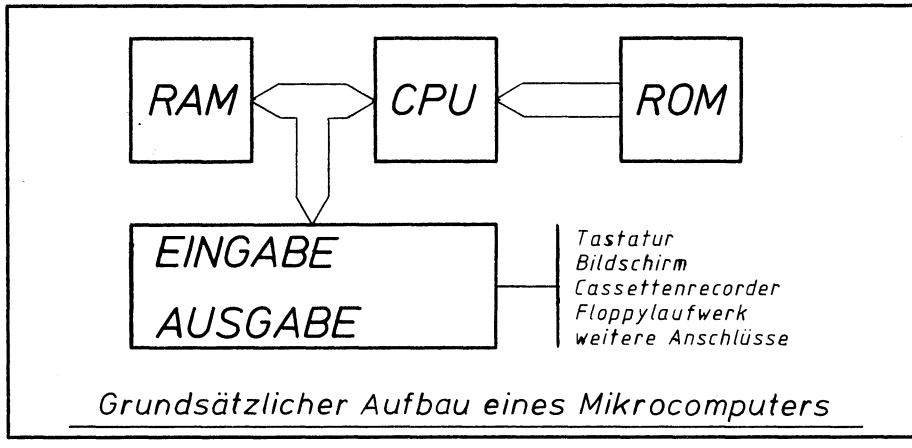


Bild 2.1

Das wichtigste Bauteil in Ihrem C16 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. Sie 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.

Die CPU arbeitet ständig, vom Einschalten bis zum Ausschalten des Computers, Programme ab. Wenn Sie Ihren C16 einschalten, liest die CPU an

einer ganz bestimmten Stelle im ROM 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 C16.

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 C16 nun auch in BASIC programmieren können, ist im ROM ein Übersetzungsprogramm vorhanden. Dieses Programm stellt sich übrigens selbst vor, wenn Sie den C16 einschalten. Auf dem Bildschirm erscheint die Meldung »COMMODORE BASIC V3.5 12277 BYTES FREE«. 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 *Speicherstelle* einzeln »angesprochen« werden. Die Speicherstellen sind durchnummeriert. Durch den Schaltzustand der Adreßleitungen wird eine ganz bestimmte Speicherstelle ausgewählt.

Dadurch weiß die CPU aber noch nicht, welcher Wert nun in dieser Speicherstelle enthalten ist. Dafür gibt es den *Datenbus*. Dieser besteht aus acht Leitungen. Man spricht daher auch vom 8-Bit-Mikroprozessor. Die angesprochene Speicherstelle 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 ver-

weisen auf weiterführende Literatur, die sich mit der Hardware oder mit Maschinensprache 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.

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, die 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 dargestellt.

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

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.

Sie können sich mit Ihrem C16 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 nun 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 eine 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 $2^{16} = 65535$ und auf dem Datenbus Werte bis $2^8 = 256$ darzustellen. Das heißt, in unserem Computer können maximal 65535 verschiedene Speicherstellen angesprochen werden. Beim C16, das sei am Rande erwähnt, sieht das etwas anders aus, hier können durch raffinierte elektronische Schaltungen mit den 16 Adreßleitungen noch mehr Speicherstellen erreicht werden.

Jede Speicherstelle kann Werte von 0 bis 255 enthalten. 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 eine 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:

```
%0010 1101   =   $2D   =   45
%1111 0001   =   $F1   =  241
```

Im C16 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 Bytes (ein Kilo-Byte) handelt. Das »K« steht aber nicht für »Kilo«, sondern für 2^{10} Bytes, das sind 1024 Bytes. Außerdem ist die Bezeichnung für »Kilo« ein kleines »k«. Zur Unterscheidung schreibt man also KByte mit großem »K«.

Jetzt können Sie auch genau ausrechnen, wieviele RAM-Speicherzellen Ihr C16 hat, nämlich 16 KByte, also 16x1024 Byte. Das sind genau 16384 Byte. Das eingebaute ROM hat 32 KByte, also 32768 Byte.

Damit ist dieses Kapitel beendet. Da Sie Ihren Computer nicht gekauft haben, um ein Buch lesend, vor dem Gerät zu sitzen, 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, evtl. 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 bemerkbar gemacht hat. Doch dazu in einem späteren Kapitel 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 wir 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 einmal größere Zahlen berechnen. Allerdings werden Zahlen mit mehr als neun Ziffern nicht mehr genau berechnet.

Nun 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 spingt 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.

Und 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.

So, Sie haben nun Ihr erstes Programm im Computer, das war doch ganz einfach, nicht wahr? Achten Sie auch einmal darauf, wie schnell der Computer die Zahlen berechnet. Mit dem Taschenrechner oder mit Bleistift und Papier hätten Sie sicher länger gerechnet. Nun noch ein kleiner Tip: Sie möchten sicher dieses Programm nicht jedesmal neu eintippen, wenn Sie einmal 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 einmal 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. dann, wenn der Computer gerade die Binärzahl berechnet. Geben Sie also eine Zahl ein, oder drücken Sie nur die **RETURN**-Taste, und drücken Sie dann unmittelbar darauf die **RUN/STOP**-Taste. Das Programm stoppt

evtl. 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).

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

SAVE "DEZIMAL/HEX"

Oder Sie speichern auf eine Diskette mit:

DSAVE "DEZIMAL/HEX"

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

3.3 HEX$ und DEC

Der C16 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 einmal

PRINT HEX$ (241)

ein. Auf dem Bildschirm erscheint:

```
00F1
```

Das ist ein sehr nützlicher und starker Befehl, den viele andere Computer nicht kennen. Das C16-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 auch einmal aus. Geben Sie

PRINT DEC ("00F1")

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

```
241
```

Soviel zu den Zahlensystemen. Wir kommen nun zu den logischen Verknüpfungen und den Variablen. Diese beiden Themen müssen noch durchgearbeitet werden, dann kann es an das Programmieren gehen.

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 kann unser C16 selbstverständlich die Grundrechenarten

- Addition
- Subtraktion
- Multiplikation
- Division
- Potenzieren

Nun wissen Sie 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 nun 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.

Nun folgt die Beschreibung einiger Funktionen.

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

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

Schreibweise:

`SGN (x)`

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

-1 wenn der Wert bzw. das Ergebnis von x negativ ist.
0 wenn das Ergebnis 0 ist.
1 wenn sich ein positives Ergebnis ergibt.

Um den absoluten Wert einer Zahl zu bilden, dient die Funktion ABS. Dabei wird ein evtl. vorhandenes negatives Vorzeichen in ein positives Vorzeichen umgewandelt.

Schreibweise:

ABS (x)

Auch hier kann für x entweder ein einzelner Wert oder eine Variable oder eine komplette Formel eingesetzt werden.

Das Ergebnis von A ist immer eine positive Zahl oder Null.

Der C16 kann auch Quadratwurzeln bilden. Dafür gibt es den BASIC-Befehl SQR.

Schreibweise:

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.

Schreibweise:

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.

Schreibweise:

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 C16 leider nicht, es ist aber dennoch möglich, sie zu berechnen. Dazu gibt es folgende Formeln:

ln = Bezeichnung für Logarithmus zur Basis e
lg = Bezeichnung der Zehnerlogarithmen

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

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

4.2 Winkelfunktionen

Der C16 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. Nun kann es 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.

Schreibweise:

```
ATN(y)
```

Und damit beenden wir unsere Reise in die Welt der Mathematik. Bedenken Sie, daß der C16 nicht so genau rechnet wie ein Taschenrechner, denn der BASIC-Interpreter benutzt Näherungsformeln, die zwangsläufig zu Ungenauigkeiten führen.

4.3 Definition von Funktionen - 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 also so 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.

Schreibweise:

```
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 innerhalb eines Programms verwendet werden. Im Direktmodus würde eine Fehlermeldung erscheinen.

Sie können nun 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 aber auch statt W den Winkelwert direkt angeben:

FN WINKEL (45)

Es wird hier also 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 dem 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 Würfelspieles. Ebenso können Zufallszahlen für statistische Berechnungen verwendet werden. Eine weitere Anwendung wäre evtl. das Verschlüsselnvon Daten.

Schreibweise:

RND (x)

x beliebige Zahl oder Variable.

Zu beachten sind 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 nun lange Zahlenreihen über den Bildschirm. Auf den ersten Blick scheinen 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 genau, die »Zufallszahlen« werden nach genau vorgegebenen Formeln berechnet. Außerdem wiederholen sich die Zufallszahlen nach einer gewissen Zeit.

Zum Beweis schalten Sie bitte Ihren C16 aus. Schalten Sie ihn nun wieder ein, und geben Sie ein:

```
PRINT RND(1)
```

Auf dem Bildschirm erscheint die Zahl

```
1.07870447E-03
```

RND (positive Zahl) beginnt also immer mit dem gleichen Wert. Auch die folgenden Werte haben immer die gleiche Folge. Das liegt daran, daß die Zufallszahlen nach genau vorgegebenen Formeln berechnet werden.

Sehen wir uns nun einmal den Befehl RND mit negativem Klammernargument an. Geben Sie bitte ein:

```
PRINT RND(-1)
```

Auf dem Bildschirm erscheint

```
2.99196472E-08
```

Nun kommt etwas Besonderes: Geben Sie den gleichen Befehl noch einmal 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 C16 erzeugt. Sie bekommen also 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 dem ersten Blick erscheint der Befehl RND(0) als die ideale Art, Zufallszahlen zu erzeugen.

Wir versuchen nun einmal, 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.

Diese 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 gebrauchen 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 nun 0,5 dazu und bekommt als Ergebis 1,9. Nun wird der Nachkommaanteil 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 enthalten, auf eine beliebige Anzahl 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 nun mit 100 multipliziert, dann durch INT abgerundet und anschließend durch 100 dividiert. Die ursprüngliche Zufallszahl wird nun links und die gerundete Zahl rechts auf dem Bildschirm ausgedruckt.

Die Multiplikation mit 100 und anschließende Division durch 100 ergeben also 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, wieviele Stellen die Zahl haben soll. Sie können die Frage mit 2 bis 6 beantworten. Anschließend geben Sie bitte ein, welche Ziffern 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 ja die Anzahl der Versuche verringern. Dazu müssen Sie die Programmzeile 55 ändern.

Wir wünschen Ihnen nun viel Spaß beim 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$(147);"***SUPERHIRN***":PRINT
330 PRINT"10 VERSUCHE, ZIFFERN VON 1 BIS ";ZH:PRINT
340 PRINT"VERSUCH        ERGEBNIS"
350 PRINT
360 FORI=6TO15
370 PRINT"NR.";I-5;TAB(7);":      :"
380 NEXTI
390 CHAR,1,VM+8,"":PRINT"IHRE ZAHL, BITTE"
400 RETURN
495 :
500 REM EINGABE
505 :
510 DOUNTILX>ZM
520 Z$="":GETZ$:IFZ$=""THEN520
525 IFZ$=CHR$(20)THENZ$="":ZA$(X)=" ":X=X-2:GOTO560
540 IFZ$=CHR$(13)ANDX=ZMTHEN560
545 IFZ$=CHR$(13)ANDX<ZMTHEN520
550 IFZ$<"1"ORZ$>CHR$(48+ZH)THEN520
555 IFX=ZMTHEN520
560 X=X+1:IFX<1THENX=0
570 ZA$(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 nun 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. Lesen Sie bitte dieses Kapitel durch, auch wenn Sie sich nicht vorstellen können, wozu Sie später einmal so etwas gebrauchen können.

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
-------------------------------------
Ergebnis %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 einmal 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

Nun unser Beispiel:

```
                ODER
      A  %0110 1011      107      $6B
      B  %1101 0010      210      $D2
-----------------------------------
Ergebnis  %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.

A	NOT A
0	1
1	0

Allerdings funktioniert das mit unserem BASIC-Befehl NOT nicht. 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 C16 einen Grafik-Befehl, der eine Exclusiv-Oder-Verknüpfung durchführt. Es handelt sich dabei um den Befehl GSHAPE, den wir im Kapitel über die Grafik noch genau erklären werden.

Das Ergebnis einer Exclusiv-Oder-Verknüpfung, kurz EXOR genannt, ist dann 1, wenn entweder die erste Aussage 1 ist oder die zweite Aussage 1 ist. Das Ergebnis ist aber nicht 1 wenn beide Aussagen 1 oder beide Aussagen 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
-----------------------------------
Ergebnis %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 Grafikbildschirmes mit einem anderen Bitmuster verknüpft.

Ein anderes Beispiel: In Ihrem C16 gibt es Steuerregister, in denen jedes Bit eine andere Bedeutung hat. Solche Register gibt es zum Beispiel im TED-Chip. Wir sehen uns einmal das Register mit der Adresse 65286 an. In diesem Register zeigt das Bit 3 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 ja 25 Zeilen, also können wir ihn jetzt einmal auf 24 Zeilen umschalten. Dazu muß also Bit 3 den Wert 0 bekommen. Das Register 65286 hat aber noch 7 andere Bits, die für verschiedene Funktionen gebraucht werden, diese dürfen wir also 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.

Also geben wir folgendes ein:

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

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

Der Befehl PEEK kann den Inhalt einer Speicherstelle auslesen.

Im Computer läuft nun folgendes ab: 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? Das ist auch ganz einfach, wenn wir uns das einmal an einem Beispiel ansehen:

Wir haben eine Binärzahl mit dem Wert % 0001 1011

Die Bits werden von rechts nach links bezeichnet, und zwar 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. Also muß Bit 3 wieder den Wert 1 bekommen. Dazu gebrauchen wir die ODER-Verknüpfung.

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

%0000 1000 hat den Wert 8 (dezimal). Geben wir also 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 ins »Eingemachte« gehen müßten. Für die Programmierung in BASIC reicht das aus, was wir Ihnen in diesem Kapitel über die logischen Verknüpfungen gesagt haben.

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)x10)+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.

Zeile 20 erwartet eine Eingabe.

In den Zeilen 30 und 40 wird nun 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 nun 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 nicht erfüllt, wenn beide Werte 0 sind. Vergleichen Sie das bitte einmal 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 nicht gleich 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 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, das 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)x10)+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 auch 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 muß. Sie können also 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 Speicherstellen mit einem vorgegebenen Wert logisch verknüpfen. Wenn das Ergebnis 0 ist, wird die Speicherstelle erneut überprüft. Der Computer wartet (daher »WAIT«) also, 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 Speicherstelle irgendwann 0 ergibt. Das Programm würde sonst bei dem WAIT-Befehl stehen bleiben. 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 Speicherstelle 192 wird mit 15 Exclusiv-Oder-Verknüpft. Anschließend wird das Ergebnis mit 1 UND-Verknüpft. Die EXOR- und die UND-Verknüpfung wird solange 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 also der Inhalt der Speicherstelle 192 mit dem Wert 1 UND-Verknüpft.

Bei dem WAIT-Befehl handelt es sich also auch um einen Vergleichsbefehl, nur bietet WAIT keine Alternative wie IF...THEN. Es wird einfach solange 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 (Kassettenrekorder oder Diskettenlaufwerk).

Voraussetzung dabei ist, daß man genau weiß, welche Speicherstellen wann welchen Wert annehmen.

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

```
WAIT 239,8
```

In der Speicherstelle 239 wird vom Betriebssystem die Anzahl der gedrückten Tasten abgelegt. Wir warten nun mit unserem WAIT-Befehl solange, bis acht Tasten gedrückt wurden. Probieren Sie das einmal 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 Speicherstelle 239 mit 0 erfolgt. Dadurch ist das Ergebnis praktisch genau dem Wert der Speicherstelle 239. Dieses Ergebnis wird nun 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 nun weiter warten, bis die achte Taste gedrückt wird.

Sie können mit WAIT auch die interne Uhr des C16 abfragen, bzw. warten, bis die Uhr einen bestimmten Wert angenommen hat. Speicherstellen, die die interne Zeitmessung des C16 wiedergeben, sind die Speicherstellen 163 bis 165.

```
WAIT 163,1
```

wartet solange, bis der Inhalt von 163, verknüpft mit 1, ungleich 0 ergibt. D.h., wenn die Speicherstelle 163 den Wert 1 hat, beendet das Programm.

7
Die Tastatur des C16/116

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 C16/116 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. Zu einer Schreibmaschine sind auch noch andere Besonderheiten zu beachten, da die Tastatur noch einige Sondertasten aufweist, die den Umgang mit dem Computer einfacher gestalten.

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 deses Rechners. Fangen wir also an.

7.1 Der Direktmodus

Wenn Sie Ihren C16/116 einschalten, 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.

Kommen wir nun zu den einzelnen Tasten und ihrer Bedeutung.

7.2 Die RETURN-Taste

Mit dieser Taste, die sich rechts auf der Tastatur befindet, haben Sie eben schon Ihren ersten Erfahrungen gesammelt. Diese Taste ist eine der wichtigsten Ihres C16/116. Wird diese Taste gedrückt, so wird dem Rechner das Ende einer Eingabe mitgeteilt. Der Rechner stellt nun fest, in welcher Bildschirmzeile der Cursor steht, und versucht nun 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 in der Cursorzeile, so erfolgt lediglich ein Zeilenvorschub. Drücken Sie nun einigemale die **Leertaste** und dann die **RETURN**-Taste. Der Cursor befindet sich nun in der ersten Spalte der nächsten Zeile. Wiederholen Sie nun das ganze, diesmal aber mit **SHIFT/Leertaste**. Der Rechner meldet sich mit »READY« zurück, der Cursor steht nun 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 mit einer anderen Taste zusammen 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 dem 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-Befehls.

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	
1	Zeichenfarbe	Schwarz
2		Weiß
3		Rot
4		Cyan
5		Purpur
6		Grün
7		Blau
8		Gelb
9	Reverse Modus ein	
0	Reverse Modus aus	
S	Hält ein Programm im Ablauf oder ein Listen durch LIST ohne Meldung an. Die Fortführung erfolgt durch Drücken einer zeichenausgegebenen 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 **CRSR UP**-Taste
;	wir **CRSR 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 C16/116 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
1	Zeichenfarbe Orange
2	Braun
3	Gelb-Grün
4	Pink
5	Bau-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 C16/116 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 Diskettenkaufwerkes 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 C16/116

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.

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 **Leertaste**(n) werden wiederholt
POKE 1344,64	Es wird keine Taste wiederholt
POKE 1344,128	Es werden alle Tasten wiederholt

Wir kommen jetzt zur Beschreibung der Sondertasten **F1** bis **F8** und der **ESC**-Taste.

7.7 Die Funktionstasten

Eine Besonderheit des C16/116 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$(13), wobei 13 den ASCII-Code von »Carriage Return« darstellt. Dadurch wird das Drücken der Taste **RETURN** nachgebildet. Eine Zusammenstellung aller ASCII-Codes finden Sie im Anhang Ihrer Bedienungsanleitung. So bringt z.B. PRINT CHR$(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 wrden. 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 C16/116-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 ja 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 anschauen.

So, und nun sind die Tasten durch das Experimentieren sicher nicht mehr sinnvoll belegt. Um den Einschaltzustand zu erreichen, nennt das An-

leitungsbuch nur zwei Alternativen. Die erste wäre, den Rechner auszuschalten, die zweite, einen Reset durchzuführen. Beide Arten sind nicht zu empfehlen, da ein evtl. 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.

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.

Zeile160: 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 auch 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"C-16"« einzugeben. Falsch wäre »"PRINT"C-16"«.

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 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 Direkmodus 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 falsch 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 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 erspart. Die erste Lösung ist sicher dann die bessere, wenn es um das Einfügen nur eines Buchstaben 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, das bedeutet 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 Zeile verkleinert. Durch **ESC N** oder zweimaligem 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 Verbessern 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 diese Funktion wird 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 Verbessern.

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 am unteren Bildschirmrand wird eine Leerzeile erzeugt. Dies läßst 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 Bildschirms 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 einfach **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 **NICHT** aufgehoben! Dies wird nur durch **ESC C** erreicht.

Wir zeigen Ihnen jetzt die Verwendung der ESC-Modi/Funktionen in einem Programm.

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 sooft wie möglich zu nutzen. Optimal eingesetzt kann Sie Ihnen viel Arbeit ersparen.

Um Ihnen den Umgang mit der **ESC**-Taste etwas zu erleichtern, haben wir ein kleines Maschinenprogramm für Sie geschrieben. Auf eine Erklärung der Programmzeilen wurde bewußt verzichtet, da die Erläuterung der Maschinensprachbefehle den Rahmen diese Buches sprengen würde.

Zum Programmlisting

Bei dem Programm handelt es sich um einen sogenannten sogenannten BASIC-Lader. Solche Programme poken Maschinenprogramme in den Speicher. Speichern Sie das Programm vor dem Starten ab, da es sich nach dem Start und 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 evtl. im Speicher vorhandenes Programm wurde aber nicht zerstört.

Ausschalten läßt sich das Ganze auch, und zwar durch **ESC** dann **CLEAR/HOME**.

Wird dieses Programm wieder benötigt, reicht SYS 1630.

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.

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, inden 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 soviele 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. Zum Beispiel die Variable APFELSINEN. Probieren Sie das einmal aus:

```
APFELSINEN = 55
```

Der Computer antwortet mit:

```
?SYNTAX ERROR
```

Der Fehler steht bei APFELSINEN. Da steckt also die Sinus-Funktion im Namen. Ebenso sind 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 unterschiedet 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.

Integer-Variablen 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

Stringvariablen 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 16"
```

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 einen weitere Einschränkung: *Die reservierten Variablen des C16.*

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 C16 gerade aktuelle Uhrzeit. Ja, in Ihrem Computer ist auch eine Uhr eingebaut. Sie können diese auch einstellen:

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

(Stunden, Minuten, Sekunden der momentanen Uhrzeit)

Die Uhr ist jetzt eingestellt, und Sie können jederzeit die Uhrzeit wieder abfragen. Das können Sie natürlich auch in Programmen machen. Leider geht die Uhr nicht ganz genau.

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 außerdem beim Neueinstellen von TI$ ebenfalls auf 0 gesetzt.

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 »Fehlerbehandlung« und im Anhang A.

DS und DS$ werden im Kapitel über die Diskettenbefehle beschrieben.

8.4 Die Dimensionierung

Nehmen wir einmal an, Sie gebrauchen 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 Adresse eingegeben wurden. Dann nämlich 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 solange um 1 erhöht, bis der Wert von (X-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, sooft ausgeführt, wie Adresse 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.

Zugegeben, dieses Programm ist recht simpel, und ein Sinn ist eigentlich kaum zu erkennen. Es müßte zumindest möglich sein, die Adressen auf Kassetten oder Disketten zu speichern. Geben Sie dieses Programm also gar nicht erst in Ihren Computer ein. Es handelt sich hierbei nur um ein Beispiel, das zur ernsthaften Adressenverwaltung nicht geeignet ist. Damit Sie aber nicht glauben, die Autoren könnten keine besseren Programme schreiben, zeigen wir Ihnen im Kapitel 16 »Disketten-Programmierung« ein Programm, das Adressen auf Kassetten oder Disketten abspeichert.

Das Programm in diesem Kapitel 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 in einen Zusammenhang bringen, z.B. Nachname 1, Vorname 1, Wohnort 1 und Straße 1.

Bei diesen 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.

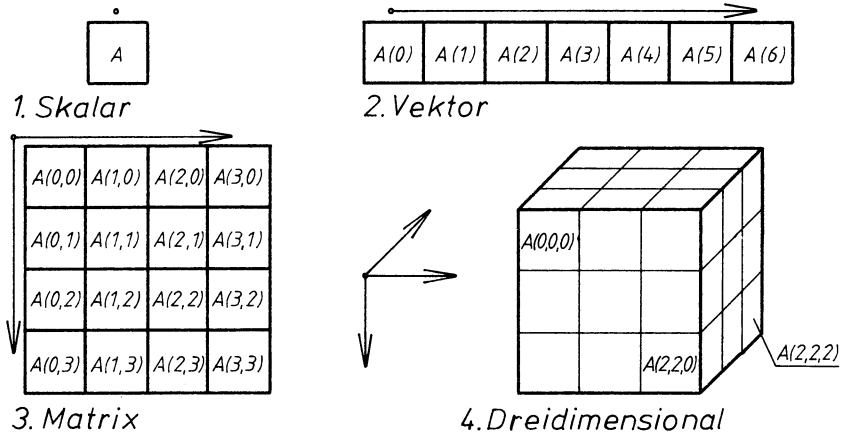


Bild 8.1

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 weiter führen. So gibt es auch vierdimensionale Felder. Leider ist uns keine 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.

Der Befehl DIM

Für die Dimensionierung gibt es den BASIC-Befehl DIM.

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 gebrauchen, 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. So könnte z.B. in einem Programm abgefragt werden, wieviele 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 gewüschte Arraygröße nicht in den zur Verfügung stehenden Speicherplatz paßt, gibt der Computer die Meldung:

```
OUT OF MEMORY ERROR
```

9
Programmerstellung auf dem C16/116

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, je nach Programm, vielleicht noch einigemale 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 finden, 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

Besprechen wir erst einmal das Anhalten des Programmablaufs durch die Tastatur.

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.

Der 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 Programmes erreicht man ebenfalls mit dem CONT-Befehl. Das Programm fährt dann mit der Abarbeitung des Befehls, der dem STOP-Befehl folgt, fort.

ACHTUNG: Sie können mit CONT kein Programm fortsetzen, wenn es in irgendeiner Weise verändert wurde. Es erfolgt sonst die Fehlermeldung »CAN'T CONTINUE 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 C16/116 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.

Der END-Befehl

Syntax:

END

Dieser Befehl wird immer an der (den) Stelle(n) eingesetzt, an der ein Programm beendet werden soll. Er darf sich an jeder Stelle eines Programmes befinden. Oft befinden sich 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 C16 wird die eingegebene Zeile nicht auf evtl. vorhandene Fehler überprüft, das geschieht erst während des Programmablaufes.

Der C16 verfügt über einen sogenannten bildschirmorientierten Editor, was nichts anderes bedeutet, daß Sie den Cursor auch beim Eingeben eines Programms frei auf dem Bildschirm bewegen können. Diese Tatsache macht den C16 besonders bedienfreundlich, da auch nachträglich sehr schnell eine vorhandene Zeile verändert werden kann. Danach kann man 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 C16 stellt Ihnen viele komfortablen Progrmmierhilfen 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 gebrauchen. Mit ihm können Sie sich jederzeit das gesammte Programm ansehen. Nun einige Erklärungen zur Syntax :

`LIST`	Es wird das gesammte Programm aufgelistet.
`LIST Zeilennr`	Zeigt die Programmzeile, die in Zeilennr. angegeben wurde.
`LIST Zeilennr.-`	Alle Zeilen ab Zeilennr. werden gelistet.
`LIST - Zeilennr.`	Alle Zeilen bis einschließlich Zeilennr. werden gelistet.

`LIST Zeilennr. - Zeilennr.` 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 dann 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, so 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« schalten 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"
```

Nach **RETURN** erscheint nun am linken Rand des Bildschirms die Zahl 20. Wenn Sie jetzt 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 nämlich, ob sich im Speicher bereits eine Zeile mit derselben Nummer befindet. Sollte dies der Fall sein, so 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 machen wir 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.

Sie sehen, es klappt.

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 erfogt die Neunumerierung der Zeilen von der ersten bis zur letzten Zeile im 10er Abstand, und auch 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 Neunummerierung

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 gesammte 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 dies 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. Aber es kommt noch besser.

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. So können Sie zum Beispiel statt PRINT auch das Fragezeichen (?) eingeben. Durch diese Abkürzungen läßt sich beim Programmieren eine Menge Arbeit ersparen. 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 so 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 solange ?: ein, bis zwei Bildschirmzeilen vollgeschrieben sind. Nun folgt List. Sie sehen, mehrere Bildschirmzeilen sind mit »PRINT:« vollgeschrieben. Auch wenn Sie mit RUN starten, werden diese PRINT-Befehle klaglos abgearbeitet. Der C16/116 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, so erfolgt die Fehlermeldung »STRING TOO LONG ERROR«.

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

9.4 Verbessern von Programmen

Über den bildschirmorientierten Editor lassen sich sehr schnell irgendwelche Änderungen und Berichtigungen eines Programmes 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 Leezeile 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 somit 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 dann 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 C16/116 läßt uns auch hier nicht im Stich.

9.4.1 Der DELETE-Befehl

Syntax:

`DELETE Anfangszeile - Endzeile`

Dieser Befehl versetzt Sie in die Lage, bestimmte Programmbereiche zu 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, so werden alle Zeichen bis einschließlich der Zeile 200 gelöscht. DELETE 200- schließlich löscht alle Zeichen 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 diesen 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.

9.5 Programmiervorbereitungen

Jedes Programm, das erstellt wird, 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 und 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 unüber-

sichtlich. 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, der im Programm aber nichts bewirkt.

9.5.1 Der REM-Befehl

Syntax:

REM

Dieser Befehl bewirkt nichts. 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. Wir, die Autoren sprechen aus eigener Erfahrung. Bei längeren Programmen sollten die ersten Zeilen mit REM beginnen, um dahinter die im Programm vorkommenden Variablen zu dokumentieren. Das könnte wir folgt aussehen :

```
10 REM NN=NACHNAME VN=VORNAME ST=STADT
20 REM II=LAUFVARIABLE, KANN IN 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 C16-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 Falle vor dem ersten Start abgespeichert werden, insbesondere wenn POKE's vorhanden sind. Ist der Rechner erst einmal abgestürtzt, dann hilft nämlich meist nur der Druck auf die **RESET**-Taste. Das Programm ist dann aber völlig zerstört.

Wie man Programme effektiv und übersichtlich erstellt, erfahren Sie im nächsten Abschnitt.

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, so einfach ist das leider nicht. Sicher, die Programme, die man einfach Zeile für Zeile schreibt, laufen evtl. fehlerfrei und erfüllen ihre Aufgabe, sodaß 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 inkraft treten, und die Berechnung daher geändert werden muß? Sie müssen also Ihr Programm entweder neu schreiben, oder die Änderungen am schon bestehendem 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 durchfühen 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 (2 KByte beim C16 mit eingeschalteter Grafik).

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

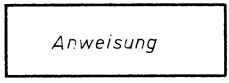


Bild 9.1

Operation, allgemein

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

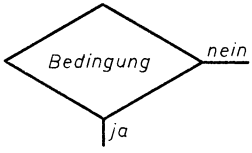


Bild 9.2

Verzweigung

Die Bedingung für die Verzweigung sollte im Symbol eingetragen werden.

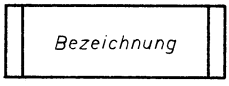


Bild 9.3

Unterprogramm

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

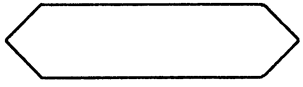


Bild 9.4

Programm-Modifikation

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

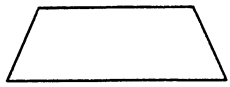


Bild 9.5

Operation von Hand

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

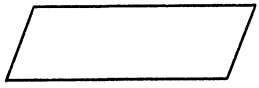


Bild 9.6

Eingabe, Ausgabe

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

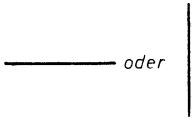


Bild 9.7

Ablauflinie

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

Bild 9.8

Zusammenführung

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

Bild 9.9

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.

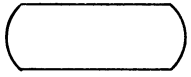


Bild 9.10

Grenzstelle

Programmstart oder Programmende.

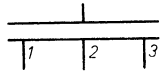


Bild 9.11

Aufspaltung

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

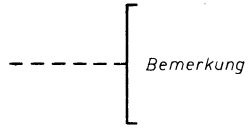


Bild 9.12

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.

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 C16 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**

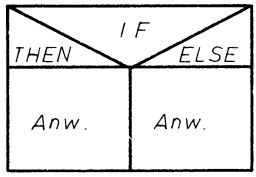
Sequenz

Mehrere Operationen.

Bild 9.13

Verzweigung

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



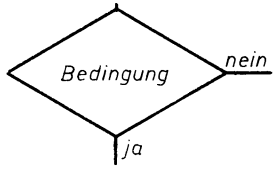
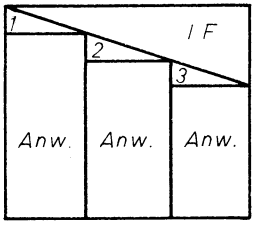


Bild 9.14

Mehrfache Verzweigung

Z.B. ON..GOSUB..



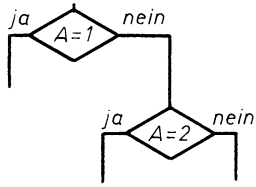


Bild 9.15

Wiederholung

Ein Programm wird solange wiederholt, bis eine Bedingung erfüllt ist.

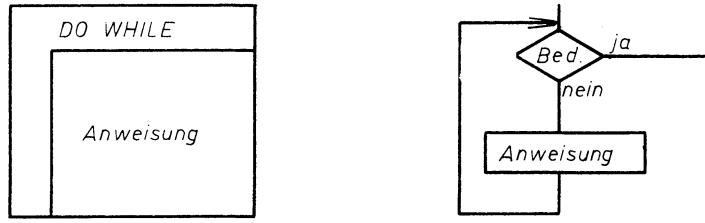


Bild 9.16

Unterprogramm

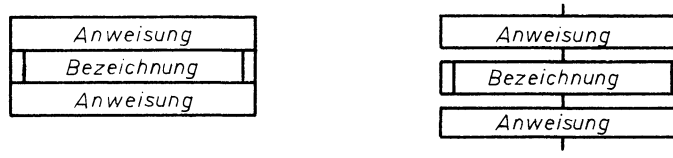


Bild 9.17

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 Eingegebenes Datum ausgeben
3.3 Wochentag ausgeben
3.4 Frage, ob weitere Berechnungen durchgeführt werden sollen

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

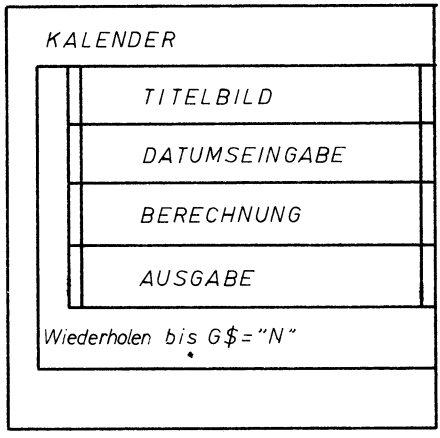


Bild 9.18: *Struktogramm des Programms KALENDER*

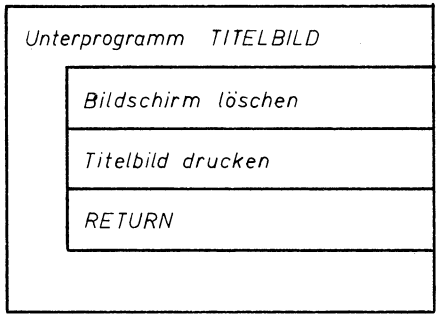


Bild 9.19: *Struktogramm des Unterprogramms TITELBILD*

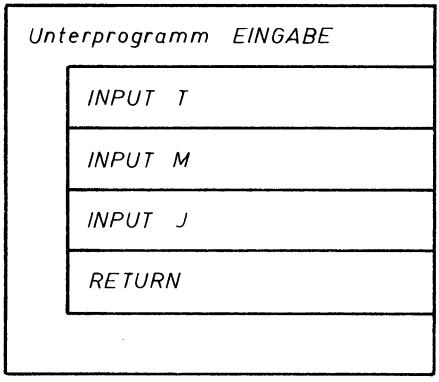


Bild 9.20: *Struktogramm des Unterprogramms EINGABE*

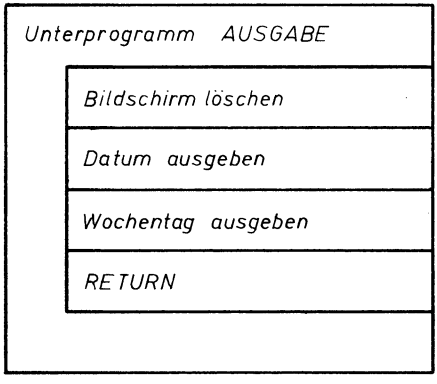


Bild 9.21: *Struktogramm des Unterprogramms AUSGABE*

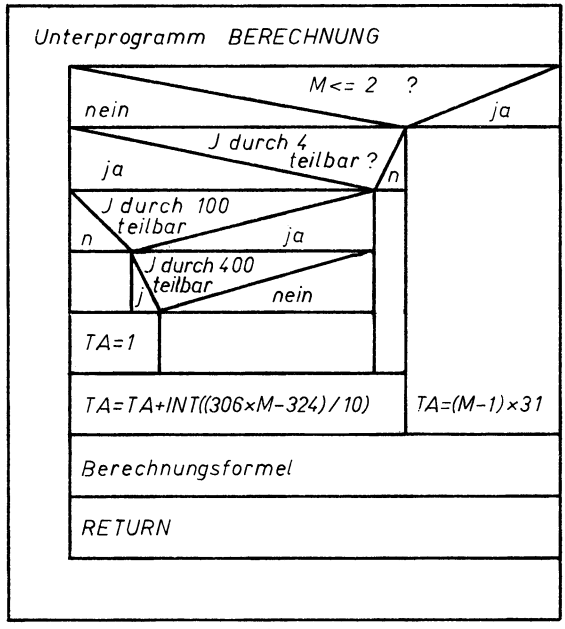


Bild 9.22: *Struktogramm des Unterprogramms BERECHNUNG*

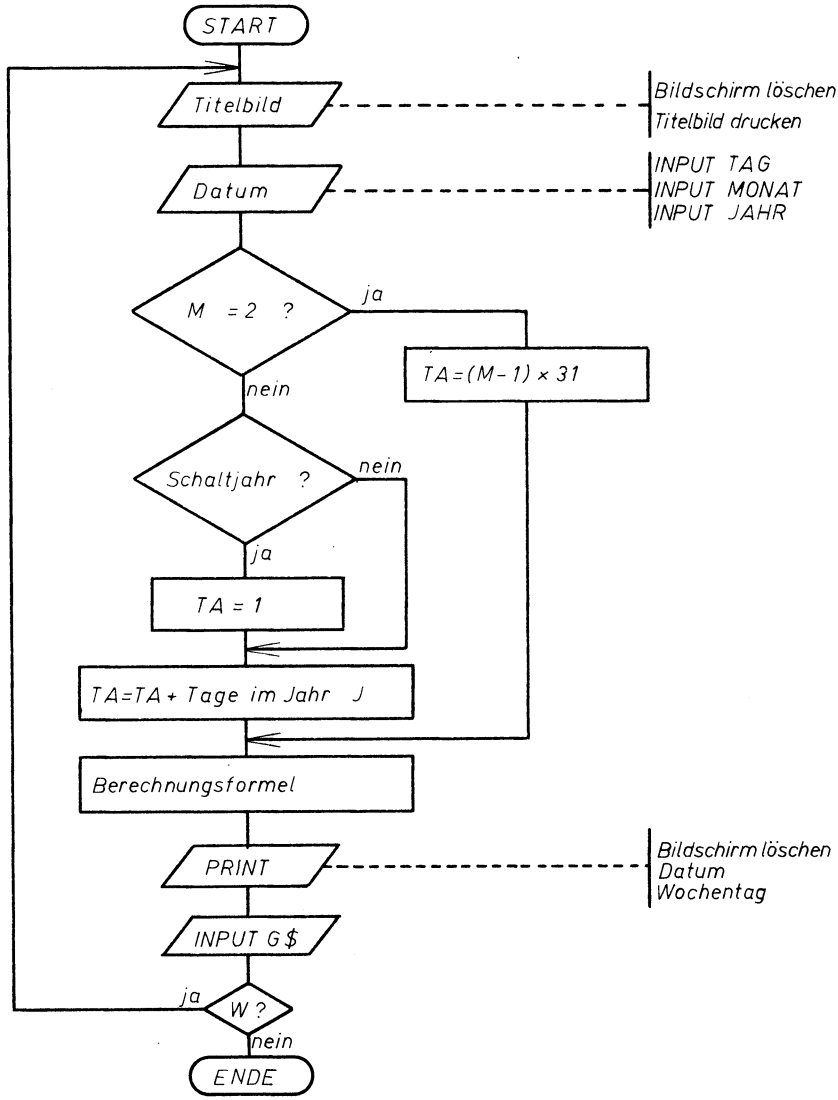


Bild 9.23: *Programmablaufplan des Programms KALENDER*

Sie sehen, beim Struktogramm sind wir einen anderen Weg gegangen. Die Programmblöcke werden hier als Unterprogramme eingesetzt. Diese Unterprogramme werden solange 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 Unterprogramm 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 vorzunehmem.

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 PRNT
260 PRINT" GEBEN SIE EIN BELIEBIGES DATUM EIN."
270 PRINT
280 PRINT" ICH WERDEN 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 fuktionieren. Die Berechnung im Unterprogramm ab Zeile 500 ist genauso, 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 Biorhytmus 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-Rhytmus, 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 C16 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. Notieren Sie auch die genaue Bedeutung der Variablen. Verwenden Sie sinnvolle Namen für Ihre Variablen.

10
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 Programmes 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 5. 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 Aufwandes 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 4 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 ums Beste bestellt. Wie soll ein Fremder oder auch man selbst noch nach ein paar Wochen wissen, welche Eingaben er 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 so auf Ihr Versäumnis aufmerksam machen. Sie werden so 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 in $ (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 stattdessen Buchstaben eingeben, erscheint die Fehlermeldung »?REDO FROM START«. Das Programm wird nicht abgebrochen, vielmehr erscheint der Kommentar des INPUT-Befehls von neuem, 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 deises Befehles 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 deisem Grund erhielt der C16/116 noch weitere Eingabebefehle.

10.2 Der GETKEY-Befehl

Syntax:

GETKEY Variable,Variable,...

Die Syntax dieses Befehles gestattet einem nicht, einen Kommentar mit auszugeben. Wird ein Kommentar erwünscht, so muß dieser vorher mit dem PRINT-Befehl auf dem Bildschirm gebracht werden.

Beim Antreffen diesen Befehles 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 werden Sie im Kapitel über die Zeichenausgabe mehr erfahren.

Sobald Sie nun das Programm starten, so 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 Falle, daß das Programm von neuem abläuft.

Wenn Sie nun einmal 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
```

Na, wie gefällt Ihnen das Ergebnis?

Nun können Sie auch den Unterschied zum INPUT-Befehl feststellen. Mit dem GETKEY-Befehl ist das »Ausschalten« einzelner Tasten möglich geworden. Natürlich wurden die Taste **CLEAR/HOME** nicht wirklich abgeschaltet. Das Programm bekommt nun 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 nun die Lösung diese 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 nun 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 also.

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 sicher auch sehr schnell auffallen:

Das Betätigen der **F**-Tasten verursacht keinen Programmabsturz mehr.

Nun erst einmal die Programmbeschreibung:

Zeile 10 Hier wird die Variable I auf 0 gesetzt. Diese Zeile ist in diesem Programm wenn mit RUN zwar umsonst gestartet wird, 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 entsprechend 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 einmal einige Tasten schnell hintereinander, lehnen Sie sich dann zurück und achten dann einmal 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 denn nun die Zeichen zwischengespeichert werden.

Der C16/116 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 Bildschirmes sich der Cursor befindet? Ein Teil dieses Bereiches ist der sogenannte Tastaturpuffer. Er hat eine Länge von zehn Speicherzellen. Daneben existiert noch eine Speicherstelle, die dem Rechner mitteilt, wieviele 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 Tastenpuffer übernommen und dann die Zählerspeicherstelle 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 Programms 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 abgearbeitet wird (kein Programmstop), daß es Ihnen wohl nur sehr selten gelingen wird, ein Zeichen an Ihr Programm zu übergeben.

Ihnen wird aufgefallen sein, daß ich in den Beispielen über den GETKEY- und den GET-Befehl nur Stringvariablen benutze. Dies hat seinen Grund darin, daß ansonsten beim Übergeben von Buchstaben an eine nummerische Variable die Programme mit der Fehlermeldung »TYPE MISMATCH

ERROR IN...« ausgestiegen wären. Dem kann man sicher 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 es zum weiteren Programmablauf notwendig, daß die Eingabe numerisch ist, so ist dies mit dem C16/116 kein Problem, bietet er einem doch die Möglichkeit, Strings in numerische Variablen umzuformen. Hierzu, und 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, 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 Programmes 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 nun 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 dann die Frage nach der Zahl und dann kommen die Ergebnisse.

Finden Sie dies optimal? Wir auch nicht. Fangen wir also damit an, etwas Ordung 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 also 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 "C 16"
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 Bildschirmes, so wird beim nächsten PRINT der Bildschirminhalt um eine Zeile nach oben gescrollt und der Inhalt der ersten Zeile geht verloren. Der C16/116 bietet einem die Möglichkeit, das Scollen 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 Programmablaufs nicht sichtbar ist, gesteuert. Der Bildschirm des C16/116 hat 40 Spalten und 25 Zeilen. Stellen Sie sich den Bildschirm in vier gleich große Teile aufgeteilt vor. Das ergibt dann 4*10 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 nun 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:

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

Trifft der Rechner auf diese Programmzeile, so nimmt er an, das 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 nun auf dem Schirm ausgegeben. Danach folgen wieder Kommata. Nach dem ersten Komma wird der Cursor in die Spalte 30 gesetzt. Da der C16/116 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 nun folgende Komma setzt den Cursor noch einmal um zehn Spalten nach rechts, und nun 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 nummerische 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 evtl. 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 "C 16";
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 »C 16« ein Leerzeichen (Space) anzuhängen. Nach getaner Arbeit erscheint der Satz leich 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 voran gestellt und ein Leerzeichen angehängt wird. Ändern Sie bitte die Zeilen 5 und 20 wie folgt :

```
5  A=16
20 PRINT "C";A;
```

Die Ausgabe erfolgt, obwohl dem Wort »C« kein Leerzeichen folgt und dem Wort »Computer« kein Leerzeichen voran gestellt ist, mit Leerzeichen zwischen den einzelnen Worten. Dies gilt aber wie schon gesagt nur für numerische Variablen. Stringvariablen werden wieder unmittelbar an den Text angehängt. Der Beweis :

```
5  A$="16"
20 PRINT "C";A$;
```

Wir müssen also der Übersicht zuliebe wieder Leerzeichen in den Text einfügen. Das Semikolon zwischen »C« und A$ braucht nicht mit eingegeben zu werden. Die Textausgabe funktioniert auch in diesem Falle einwandfrei. Probieren Sie's!

Es folgt nun ein etwas längeres Programm, welches Ihnen auch später sicher noch gute Dienste leisten wird.

Möchte man ein Programm mit einem längeren Einleitungstext versehen oder sollen Texte während eines Programmablaufs 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 Wörter in andere Programmzeilen übertragen werden, um den Text auf dem Bildschirm übersichtlich zu gestalten. Ein späteres Einfügen oder Löschen von Wörtern und Sätzen ist auch nur sehr schwer zu bewerkstelligen.

Dieses Programm beendet diese Misere durch eine spezielle Eingabe- und Ausgabetechnik. Die Texte müssen in DATA-Zeilen abgelegt werden. Das Programm liest nun jeweils eine DATA-Zeile und untersucht diese nach Space (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 dieses der Fall, so wird diese Position gespeichert und die Position des nächsten Leerzeichens gesucht. Dieses wird solange 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 evtl. zwischengespeicherte Text ausgegeben und das Programm beendet. Es können auch sehr einfach Absätze oder Leerzeilen eingegeben werden.

Zur Bedienung :

Wird dieses Programm als Unterprogramm verwendet, 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 erfogt 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 Textteile muß mit »-1« beendet werden.

Sie müssen auch darauf achten, 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 im 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. Sollte Ihnen, was höchstwahrscheinlich ist, irgendetwas im Programm nicht klar sein, so möchte ich Sie auf die vorangegangenen und folgenden Kapitel verweisen. Alle Befehle, die dieses Programm erhält, werden dort ausführlich beschrieben. Doch nun das Programm :

```
 1 REM  **** KOMFORTABLE AUSGABE VON TEXTEN ****
 2 :
 3 REM  ***** DIE VERWENDETEN VARIABLEN : ******
 4 REM A$= HIERIN BEFINDET SICH DER EINGELESESENE STRING
 5 REM B$= 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 "DIESES 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. Diese Anweisung wird in Anführungszeichen eingeschlossen. Es muß ein Semikolon folgen, welches dann von den auszugebenden Variablen (String- oder nummerische 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 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 10
```

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

Starten Sie nun das Programm. Beantworten Sie die Frage nach dem Format mit #### (vier Rauten) und die Frage Zahl mit 1234. Es erscheint nun die Zahl 1234 linksbündig in der nächsten Zeile. Die Formatfrage beantworten Sie nun 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 untereinander usw..

Behalten Sie das Format bei und geben Sie nun als Zahl 12345 ein. Es erscheinen vier Sternchen auf dem Bildschirm. Jedes Sternchen repräsentiert eine # im Format. Sie müssen also für jede, in einer Zahl enthaltenen Ziffer, ein # in das Formatierungsfeld schreiben.

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 nun einmal 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 wir nun 123.4 ein. Das Ergebnis wird mancher nicht erwartet haben. In diesem Falle hat der Rechner abgerundet und die Zahl dann angezeigt. Diese Tatsache macht diesen Befehl sehr komfortabel, wird einem doch zusätzliche Programmierarbeit abgenommen.

ACHTUNG: Der C16/116 verarbeitet lediglich Zahlen mit nicht mehr als neun Ziffern. Geben Sie zum Beispiel 11 Rauten und dann eine 11-stellige 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 nun einmal willkürliche Zahlen ein. Sie werden feststellen, daß die Dezimalpunkte immer untereinander stehen. Was passiert nun aber bei der Eingabe einer negativen Zahl? Probieren wir es doch 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; wie, wird jetzt erklärt.

11.2.3 Plus- und Minuszeichen bei PRINT USING

Geben wir nun das Format mit -###.## (Minuszeichen, dreimal Raute, Dezimalpunkt, zweimal Raute) an, und geben dann die Zahl 123.45 ein.

In der nächsten Zeile erscheint nun 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 also jeder positiven Zahl ein Pluszeichen voangestellt. Ein bei der Eingabe der Zahl ebenfalls eingegebenes Pluszeichen wird nicht mit angezeigt. +123 ergibt also ebenfalls »nur« +123.00.

Auch negative Zahlen können in diesem Format dargestellt werden:

-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 solch ein 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 nun folgende Zahl im wissenschaftlichen Format an. Für all diejenigen, denen dieses Format nicht geläufig ist, folgt nun erst einmal 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 einmal 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 nun 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 nun zum 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 C16/116 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 C16/116 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 also mit dem PRINT USING-Befehl weiter.

Als Format geben wir #^^^^ ein. Die Zahl soll 1000.22 sein. Das Ergebnis lautet: 1E+03.

Sie können also feststellen, daß der Rechner von sich aus gerundet hat, da die 0,33 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 dann aber ungenau. Es können Zahlen von 1E+38 bis 1E-38 mit dem C16/116-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 auch sehr einfach die formatierte Ausgabe von Hexadezimalzahlen auf dem Bildschirm.

Wählen wir nun das 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 dann 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 mit 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 Bespiel 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.

Wir kommen nun zur Textausgabe mit dem PRINT USING-Befehl.

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 folgendermaßen ändern:

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

So, nun sind die Vorarbeiten 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: `C16`

Den Unterschied zur Zahlenformatierung werden Sie sicher sofort erkannt haben. Der Text wird linksbündig ausgegeben. Geben wir nun als Text ein: C16/116

Na, da sind Unterschiede, nicht wahr? Es erscheinen also keine Sternchen, sollte der Text auch mehr Zeichen umfassen als Rauten bei der Formatierung angegeben sind. Der Text wird also um die rechten Zeichen gekürzt, die nicht mehr in das definierte Format passen. Machen Sie nun auch noch einmal folgendes:

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

Text: `DER COMPUTER`

Kommen wir nun gleich zur nächsten Formatangabe für Texte.

11.2.10 Das Gleichheitszeichen bei PRINT USING

Wir haben oben festgestellt, daß die Textausgabe in der ersten Spalte beginnt und soviel Zeichen umfaßt wie Rauten bei dem Formatstring vorhanden sind.

Geben Sie nun bitte als Format ein: = und dann 39 Rauten.

Als Text nehmen wir: `DER C16 COMPUTER`

Ja, so einfach ist es, Texte in die Mitte des Bildschirmes zu bringen. Das Gleichheitszeichen (=) bewirkt nämlich, daß der Text innerhalb der Formatzeichen zentriert wird.

Nun ein anderer Fall mit dem Format =#### (Gleichheitszeichen, vier Rauten) und dem Text C-16:

Der Text erscheint rechtsbündig auf dem Bildschirm. Auch dies ist also mit diesem Formatbefehl, allerdings nur bedingt und auch ewas umständlich, möglich. Wäre in diesem Falle der Text länger als fünf Zeichen, so würden die rechten Zeichen nach dem Fünften abgeschnitten, d. h. nicht mit angezeigt.

An welcher Stelle des Formatstrings das Gleichheitszeichen steht, spielt genauso wenig 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.

Und nun zeigen wir Ihnen noch mit dem letzten Formatierungszeichen für Texte, wie man diese rechtsbündig auf dem Bildschirm ausgeben kann.

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

Wir fangen gleich mit dem Format an:

>#### (größer als Zeichen, viermal Raute)

Der Text: `C16`

Sie sehen, der Text erscheint in einem Abstand von zwei Bildschirmspalten vom linken Rand des Schirmes. Belassen wir es einmal bei diesem Format und geben als neuen Text

Sie sehen, der Text erscheint in einem Abstand von zwei Bildschirmspalten vom linken Rand des Schirmes. Belassen wir es einmal bei diesem Format und geben als neuen Text

```
DER C16
```

ein. Es wird nur DER C 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 siebente Leerzeichen wird durch Dezimalpunkt bewirkt. Trifft der Befehl nämlich 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
```

Sie sehen, 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 dem 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 "###";"C16";: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 einem noch mehr.

Sie können nämlich 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 und noch einiges mehr, bietet das BASIC des C16/116 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? Nun, der String wurde mit PUDEF umgeformt. Als erstes Zeichen wurde statt eines Leerzeichens die Raute aufgenommen. Soll der Rechner nun bei der Ausgabe von Zahlen Leerzeichen setzen, nimmt er stattdessen 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, welches ja ursprünglich in diesem String enthalten war, definiert. Dann wurde noch 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 nun 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 C16/116 wird das Dollarzeichen nur dann gegen ein anderes definiertes Zeichen ersetzt, wenn ihm im Formatstring von PRINT USING mindestens eine Raute vorangestellt wird. Wird dies mißachtet, erscheint es weiterhin auf dem Bildschirm.

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

```
PUDEF "*"
```

Alle anderen Zeichen, auch die evtl. 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. Vielleicht stoßen Sie auch auf Zusammenhänge, die in dieser Beschreibung fehlen, aber man kann im Laufe eines Kapitels nicht auf jeden speziellen Fall eingehen. Versuchen und probieren Sie daher ruhig noch ein wenig und lassen Sie sich dabei durch evtl. auftretende Fehlermeldungen nicht entmutigen. Auch wir, die Autoren, hatten anfänglich Probleme. Sie wissen ja: es ist noch kein Meister vom Himmel gefallen.

Es geht nun mit weiteren Befehlen zur Ausgabe von Zeichen weiter.

11.5 Die Positionierung des Cursors

Das C16/116-BASIC 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 C16/116 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 nun 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 C16"
```

gibt den Text, beginnend in der sechsten Spalte der fünften Zeile, aus. Im Gegensatz zum Grafikmodus werden auch alle Bildschirmsteuerzeichen, die ein String enthalten kann, 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 hinder 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 6
30 PRINT SPC(5) "C16"
40 NEXT I
```

Sie sehen, das Wort C16 erscheint nach dem Programmstart fünf Spalten von Rand entfernt; das zweite Wort fünf Zeichen vom ersten Wort usw.. Dieser Cursorsprung wurde durch SPC(5) verursacht. Möchten Sie also 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.

11.5.2 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 ja schon wissen, können in jeder Zeile des Bildschirms 40 Zeichen angezeigt werden. Mit TAB ist es nun 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 (28) "C16"
40 NEXT I
```

Nach dem Ablauf dieses Programmes sehen Sie den Unterschied zu PRINT SPC(X) sehr deutlich. Die erste Ausgabe erfolgte in der 28. Spalte der ersten Bildschirmzeile, die nächsten drei Ausgaben unmittelbar dahinter. Erst die fünfte Ausgabe erfolgte wieder in der 28. 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 also 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 C16/116 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"
```

Zu 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. Nun folgen noch Anführungszeichen und dann ein Semikolon.

Sie sehen, zuerst erscheint das Wort »DER« und dann, acht Bildschirmzeilen tiefer, das Wort »COMPUTER«. Der Rechner behandelt in An-

führungszeichen eingeschlossene Steuerzeichen also so, als ob die direkt über die Tastatur eingegeben worden seien.

Das reverse Q erhält man auch, wenn man den Reversemodus mit **CTRL** und **RVS ON** einschaltet und dann das **Q** eingibt. Das Ergebnis ist das Gleiche, nur ist das Eingeben umständlicher. Der C16/116 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 " "
        |
        hier Taste CLEAR/HOME drücken
```

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

Die Steuerung des Cursors mit den Sonderzeichen trägt vor allem dann, wenn mehrere Steuerzeichen hintereinander stehen, nicht zur Übersichtlichkeit eines Programms bei. 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 C16/116 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 Steuerkodes auftreten. Nicht jeder Rechner verfügt über so vielfältige Möglichkeiten der Bildschirmgestalltung wie unser C16/116.

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$ (Kodezahl)« ausgegeben werden. Ein weiteres Beispiel:

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

Der Kode 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 sinnvollerwiese mit:

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

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

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

Versuchen Sie jetzt einmal, mit **SHIFT/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 Kode in X steht, wird als ASCII-Zeichen an A$ übergeben. Dies soll an dieser Stelle über CHR$ reichen. Wir kommen nun zu der Umkehrfunktion.

11.5.7 Der ASC-Befehl

Syntax:

```
PRINT ASC ("Zeichen")
```

oder

```
X = ASC ("Zeichen")
```

Obwohl dieser Befehl nicht sehr 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-Kode des Buchstabens A dar. Das Zeichen, dessen ASCII-Kode 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 Kode des ersten Zeichens ausgegeben, die anderen Zeichen werden ignoriert.

Auch folgendes ist möglich:

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

Beachten Sie bitte, daß Stringvariablen nicht in Anführungszeichen stehen dürfen, sobst wird nämlich der Kode 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 in einem eigenen Kapitel 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 unverändert. 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 C16/116 besteht aus 25 Zeilen mal 40 Spalten. Es können also insgesamt 1000 Zeichen auf einmal angezeigt werden. Jedes dieser Zeichen würde einen Platz des Bildschirms belegen. Wie Sie wissen, findet man in jedem Rechner, der programmierbar sein soll, RAM-Speicher (RAM=Schreib-Lesespeicher). Einen Teil dieses RAMs nennt man beim C16/116 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 von Bildschirmspeicher und 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 Zeie fort.

Ein Beispiel für die Zählung:

Welche Adresse hat die Speicherzelle der dritten Zeile und 25. Bildschirmspalte? Rechnen wir einmal:

Der Ursprung in der linken oberen Ecke hat die Adresse 3072. Addieren wir nun hierzu 40 (jede Bldschirmzeile besteht ja aus 40 Zeichen), so haben wir den Anfang des Speichers der zweiten Zeile. Zum Ergebnis werden nun

noch einmal 40 addiert. Wir erhalten 3152. Diese Zahl repräsentiert nun die Anfangsadresse des Bildschirmspeichers für die dritte Zeile. Hierzu muß nun noch 24 für die 25. Bildschirmspalte addert werden. Wir erhalten nun als Ergebnis die Zahl 3176. (Da 3152 schon die Adresse für die erste Bildschirmspalte der dritten Zeile ist, dürfen nur noch 24 addiert werden.)

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

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

Rechnen Sie mit dieser Formel einmal unser Beispiel nach.

In jeder dieser 1000 Speicherzellen steht nun ein Wert. Jeder dieser Werte repräsentiert das Zeichen, welches an der entsprechenden Stelle abgebildet wird. In jeder Speicherstelle können Werte von 0 bis 255 stehen.

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

```
POKE 3176,1
```

Es erscheint nun an der vorberechneten Stelle auf dem Bildschirm ein großes A. Sie sehen, die Werte für den Bildschirmspeicher sind **nicht** gleich dem ASCII-Kode. Auch die reverse Darstellung läßt sich mit POKE erreichen. Sie brauchen nur zu dem Wert, in diesem Falle eins, die Zahl 128 addieren. Probieren Sie es aus.

Mit dieser Methode können Sie direkt an jeder Stelle des Bildschirms alle Zeichen, die der C16/116 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 haben. 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 C16/116 hat 16 Farben, die auch noch in der Leuchtkraft verändert werden können. Soll in den Farbspeicher gepokt werden, muß man darauf achten, das dort die Farben mit Null beginnen. Null ist hier 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 Luminazstufen 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 nun einmal den Multiplikator in 11. Die Farbe hat sich nicht verändert aber der Buchstabe blinkt. Werden in den Farbspeicher Werte größer als 127 gepokt, 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.

Sie können Zeichen 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 Sie 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 sowieso 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 solange zu, wie der Rechner nicht zum Verlassen dieses Weges angewiesen wird. Die folgenden Befehle dienen dazu, den Programmablauf an einer anderen Stelle des Programmes fortzusetzen.

Sie unterscheiden dich nicht nur in der Schreibweise, sondern auch in ihren Eigenarten. Lesen Sie sich 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, in der es auftritt, ein, als noch 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.

Es folgt 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 näher zu bringen.

Man kann sehr 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.

Kommen wir zu einer Abwandlung des Befehls.

12.2 Der ON ... GOTO-Befehl

Syntax:

`ON Variable GOTO Zeilennr.,Zeilennr.,.....Zeilennr.`

Nehmen wir einmal an, Sie möchten ein Programm schreiben, in dem es möglich sein soll, mit Hilfe eines Auswahlmenüs zwischen drei verschiede-

nen Programmpunkten zu wählen. Man könnte des Menü derart gestalten, daß die Wahl durch die Eingabe einer Zahl erfolgt. Im Programm könnte man dann durch dementsprechende IF...THEN-Zeilen auf die Eingabe reagieren.

Es gibt aber eine bessere Lösung. Geben Sie erst einmal 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. Sie sehen, das Programm verzweigt an die jeweils richtige Stelle. Hat die Variable I den Wert eins, so wird zur ersten Zeilennummer, hat sie den Wert zwei zur zweiten Zeilennummer und beim Wert drei zur dritten Zeilennummer, die dem GOTO Befehl folgt, verzweigt. Geben Sie einmal vier ein. Das Programm wird nun in der Zeile 100 weiter 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 den Wert Null hat, 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 Programmes an anderer Stelle. Soll ein Programm aber nach der Abarbeitung einiger Programmzeilen wieder zurückkehren, dann ist dies nur als 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 einmal 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 sollten, 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 Programms er die Arbeit abgebrochen hat. Er verzweigt dann zur angegebenen Zeilennummer und fährt mit der Abarbeitung der dortigen Anweisungen solange 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
```

Hier nun noch eine kurze Erklärung:

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 aufrufbarten Unterprogrammen. 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 dann nämlich 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 C16/116 Befehle zur Programmierung von Schleifen. Fangen wir gleich mit dem ersten an.

13.1 Der FOR..NEXT-Befehl

Syntax:

`FOR Variable = Startwert TO Zahl, Variable STEP Zahl, Variable`

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 Variablenname folgen.
- TO kann ein Variablenname oDer eine Zahl folgen.
- STEP braucht nicht mitangegeben zu werden, der Rechner setzt dann als Schrittweite Eins fest.

Nun 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.

Programmerklärung:

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 nun wiederum mit dem Endwert verglichen. Sollte es größer sein als der Endwert, so macht das Programm mit dem Befehl weiter, der NEXT folgt. Ist es 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 nun 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 einmal 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 1*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 düren Sie nicht durch Sprungbefehle verlassen, da sie aktiviert bleiben, bis die oben genannten Bedingungen erfüllt sind. Es gibt im C16/116-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
```

Programmerklärung:

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 nun 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, wenn das, was nach UNTIL definiert wurde, eintritt.

Kommen wir nun zu WHILE.

```
1 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 verwendet, 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 nämlich 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 nun ein kleines Programm, mit dem wir EXIT näher beschreiben werden.

```
10 X = O
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 Programmlaufs die Taste **S** gedrückt wird. Dies wird durch die Zeile 40 ermöglicht. Drückt man nämlich 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 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 C16/116 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 nun 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
Stringbefehle

In diesem Kapitel werden Sie sicher auch später einige Male nachlesen. Das BASIC 3.5 bietet nämlich starke Befehle, wenn es um die Bearbeitung von Strings geht. Um einzelne Zusammenhänge nicht trennen zu müssen, erfolgt die Erklärung der Befehle nicht in alphabethischer 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 nun einmal einer solchen Variablen einen Inhalt zu:

```
A$="COMMODORE C/16"
```

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$:

Auch verknüpfen lassen sich Stringvariablen:

```
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, wieviele 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. So gibt es Fälle, in denen einem String an seinem Ende Zeichen hinzugefügt werden sollen. Dies ist aber nur möglich, wenn seine genaue Länge bekannt ist. Einige dieser Fälle werden wir Ihnen im Laufe dieses Kapitels aufzeigen.

Wir kommen nun zu einigen Befehlen, mit denen man Strings »zerteilen« kann.

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 nun 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$ wurde 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 auch einmal 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.

Es gibt auch einen Befehl, mit dem man Zeichen, von rechts beginnend, übergeben kann.

14.3 Der RIGHT$-Befehl

Syntax:

```
B$ = RIGHT$  (A$, Variable oder Zahl)
PRINT RIGHT$  (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 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 stärkenen.

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 V-ten 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 nun 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, 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.

Der MID$-Befehl kann aber noch mehr.

14.5 Die Änderung eines Strings mit MID$

Mit MID$ können Strings sehr einfach verändert werden. MID$ darf nämlich auch links des Gleichheitszeichens (=) 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$-Befehl festgelegt, bei dem fünften Zeichen. Die Angabe der Länge braucht nicht zu erfolgen, da sie von der Länge des Strings in B$ 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 A$, 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=O 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. Dieses Zeichen und seine Position im String werden angezeigt.

Mit diesem kleinen Programm ist es also möglich, auf einfache Art und Weise Zeichen 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 nun 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 also 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 Stringbefehlen weiter verarbeitet werden. Dieser Befehl ist die

Umkehrung von VAL. Im String sind nach diesem Befehl die Zahlen als Zeichen vorhanden. Mit diesem String können keine Rechenoperationen durchgeführt 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 C16/116 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 C16

Eine besondere Stärke des Commodore 16 ist zweifellos seine Grafikfähigkeit. Er ist in der Lage, 320x200, also 64000 Bildpunkte auf dem Bildschirm einzeln anzusteuern. Der C16 zaubert 16 Farben auf den Bildschirm und diese auch noch in acht Helligkeitsstufen.

Das ist aber 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 Darstellmöglichkeit von »Shapes«. Diese werden an anderer Stelle eingehend beschrieben. Für einen Rechner dieser Preisklasse sind diese Fähigkeiten ganz enorm!

15.1 Bildschirmaufbau

Nachdem Sie Ihren C16 bzw. C116 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 25x40 = 1000 Zeichen. Jedes Zeichen besteht aus 8x8 Bildpunkten. Diese Punkte können auch einzeln angesteuert werden.

Diese Möglichkeit besteht in der hochauflösenden Grafik. Dann ist der Bildschirm in 40x8 = 320 Punkte in der Waagerechten (X-Richtung) und 25x8 = 200 Punkte in der Senkrechten (Y-Richtung) eingeteilt. Dies ergibt 320 x

200 Bildpunkte, 64000 einzeln ansteuerbare Punkte. Damit lassen sich ganz ausgezeichnete Grafiken darstellen. Auf der nächsten Seite zeigen wir Ihnen ein Beispiel so einer hochauflösenden Grafik.

Manchmal ist es wünschenswert, Grafik und Text gleichzeitig darzustellen. Auch das ist mit dem C16 möglich. Der dritte Modus: Hochauflösende Grafik und Text. Dabei wird der Bildschirm aufgeteilt, in einen oberen Bereich von 320x160 Bildpunkten und einen unteren Bereich für fünf Zeilen Text.

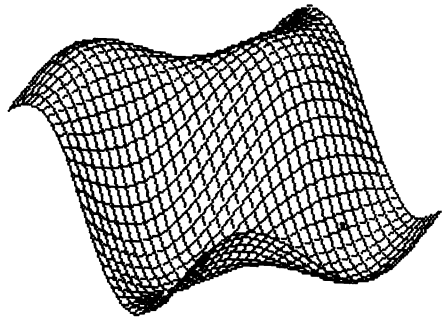


Bild 15.1

Die hochauflösende Grafik hat leider eine Schwäche: Es sind pro 8x8 Punktematrix (vergl. Textmodus) nur zwei Farben darstellbar. Auch der C16 hat seine Grenzen. Der Grund liegt im zu kleinen Speicher des C16. Wir können mit unserem Computer aber trotzdem großartige farbige Grafiken erzeugen.

Außer der hochauflösenden Grafik mit nur zwei Farben pro 8x8 Punktematrix kennt der C16 die sogenannte Mehrfarbengrafik. Dabei sind pro 8x8 Punktematrix vier Farben darstellbar. Die zusätzlichen Farben müssen aber durch eine geringere Auflösung des Bildschirmes »bezahlt« werden. Die Auflösung im Mehrfarbenmodus beträgt 160x200 Bildpunkte. Diesen Mehrfarbenmodus kann man ebenfalls mit Text kombinieren, der fünfte Modus: Mehrfarbengrafik und Text.

15.2 Die Grafikmodi des C16

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

Genau genommen befindet sich der C16 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 nun 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 fuktionieren 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` (evtl. Parameter für das Löschen)

Die Grafik wird wieder ausgeschaltet durch das Umschalten in den Textmodus:

`GRAPHIC 0` (evtl. Parameter für das Löschen)

Beachten Sie bitte, daß der C16 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. Ihnen stehen also im Grafikmodus nur noch ca. 2000 Bytes Speicherplatz für Ihr Programm zur Verfügung. Das ist recht wenig. Wenn Sie erst ein wenig Übung in der Programmierung haben, werden Sie problemlos Programme schreiben, die wesentlich länger als 2 KByte sind, und dann beginnen die Probleme. Dann heißt es, entweder eine Speichererweiterung kaufen oder mit allen Mitteln Speicherplatz sparen. Aus eigener Erfahrung können wir, die Autoren, allerdings sagen, daß knapp bemessener Speicherplatz die Programme besser werden läßt. Der Programmierer wird also gezwungen, effektiver zu programmieren. Es erscheinen weniger überflüssige Programmteile. Allerdings leidet dann die Übersichtlichkeit eines Programmes. Eine Computerzeitschrift, die sich mit Commodore-Rechnern beschäftigt, hat vor einiger Zeit einen Wettbewerb gestartet, in dem die Leser aufgefordert wurden, sogenannte »Einzeiler«, das sind BASIC-Programme, die nur eine Programmzeile lang sind, zu schreiben. Die Ergebnisse waren erstaunlich, wenn man bedenkt, daß die Programmzeilen bei den Commodore-Rechnern maximal 80 Zeichen lang sein dürfen.

80 Zeichen reichen also schon aus, um effektive Programme zu schreiben. Nun haben Sie, lieber C16-Anwender, etwa 2000 Zeichen zur Verfügung - welch ein Unterschied!

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 KByte Speicher »zurückholen«. Und das machen Sie mit dem Befehl:

`GRAPHIC CLR`

Jetzt steht Ihnen auch wieder der ganze C16-Speicher zur Verfügung. Tatsächlich ist das Grafikbild jetzt immer noch nicht gelöscht. Erst wenn Sie

ein Programm in den Speicherbereich laden oder eintippen, in dem sich der Grafikspeicher befand, ist das Bild verfälscht oder gelöscht. Das gleiche bewirken Variablen, die Sie eventuell in Ihrem Programm gebrauchen.

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, kann die hochauflösende Grafik doch nur von einem Programm erstellt werden. Die Grafik wird ja nicht direkt von der Tastatur eingegeben. Es gibt dagegen Anwendungen für Ihren Computer, denken Sie an ein Malprogramm, 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 wer-

den. 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 durch

GRAPHIC 0

wieder aus. Ändern Sie nun die Zeilen 30 und 40 wie folgt:

```
30 DRAW,TO30,10
40 DRAW,TO40,20
```

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
```

Nun werden also nur die Endpositionen angegeben.

Kommen wir noch einmal zurück zum Pixel-Cursor.

Wir können den Pixel-Cursor auch (unsichtbar) an jede beliebige Stelle des Bildschirmes 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 DRAW1,TO20,20TO30,10TO40,20
```

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 nun ein kleines Beispielprogramm, das die großartigen Möglichkeiten des Commodore 16 ausnutzt. Sie kennen sicher die Uhren, die im Fersehprogramm vor den Nachrichten gezeigt werden. Das wäre doch einmal eine interssante Programmieraufgabe für uns. Für den C16 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 werden, wenn Ihr Fernsehgerät ständig die Uhrzeit anzeigt - ein herrlicher Gag!

15.4 Beispielprogramm FERNSEHUHR

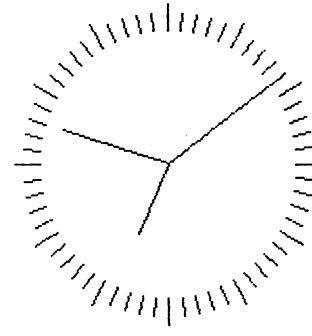


Bild 15.2

Das Problem erscheint auf dem ersten Blick einfacher, als es ist: Das Programm soll in BASIC realisiert werden. Als Uhrzeit wird die eingebaute Uhr des C16 verwendet, die natürlich vorher eingestellt werden muß. Und da ist auch schon das erste Problem: Unser 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 C16 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 C16 kennt zwar sehr starke Grafikbefehle, es ist aber nicht ganz einfach, die 60 Minutenstriche und die 12 Stundenstriche zu zeichen.

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,Y
    TOMX%(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*π/180)*RA)
```

```
340 X2=X+INT(COS(W*π/180)*RE)
350 Y1=Y-INT(SIN(W*π/180)*RA)
360 Y2=Y-INT(SIN(W*π/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. Drücken Sie dann bitte die **F1**-Taste und geben dann eine »0« ein. Nach der **RETURN**-Taste erscheint wieder der Textbildschirm (in F1 ist der BASIC-Befehl GRAPHIC gepeichert - mit GRAPHIC 0 wird in den Textmodus umgeschaltet).

Nun die Programmbeschreibung:

Wir haben gesagt, daß unser Programm schnell genug sein muß, um rechtzeitig den Sekundenzeiger zu zeichnen. Aber damit ist es ja noch nicht getan, der Minuten- und Stundenzeiger muß ja 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 C16 nicht ganz einfach ist, von einem Mittelpunkt gleichlange Linien im Abstand von 6 Grad zu zeichnen. Am einfachsten ist es, 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 gebrauchen wir Integervariablen (gekennzeichnet mit dem Prozentzeichen). Die Integervariablen haben neben der etwas schnelleren Verarbeitung den Vorteil, daß sie weniger Speicherplatz verbrauchen. In der hochauflösenden Grafik haben wir ja nur ca. 2000 Byte Speicher zur Verfügung. Unser Uhrenprogramm »verbraucht« diesen Speicherbereich auch bis auf etwa 200 Bytes.

In der Zeile 20 werden die Integervariablen für die Zeigerpositionen dimensioniert. Für den Stundenzeiger gelten 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 Bildschirmes.

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 Program aufgenommen.

Wir gebrauchen nun 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 irretieren 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 der Variablen RE minus RA. RA hat ja 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 nämlich 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«, also oben, steht. Dann muß der Minutenzeiger weitergesetzt werden.

So, das wäre die Programmbeschreibung, die etwas länger ausgefallen ist. So einfach die Uhr auch aussehen mag, so erreicht dieses Programm doch die Grenzen des in BASIC auf dem C16 Möglichen.

Nun noch eine Anmerkung:

Wenn Sie später den BASIC-Befehl CIRCLE kennenlernen werden, werden Sie sich evtl. fragen, wieso in diesem Uhrenprogramm so »umständlich« mit den Winkelfunktionen gearbeitet wird. Nun, 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*π/180)*RA*2)
340 X2=X+INT(COS(W*π/180)*RE*2)
```

oder

```
330 X1=X+INT(COS(W*π/180)*RA*2)
340 X2=X+INT(COS(W*π/180)*RE*1.5)
```

oder

```
330 X1=X+INT(COS(W*π/90)*RA)
340 X2=X+INT(COS(W*π/90)*RE)
350 Y1=Y-INT(SIN(W*π/90)*RA)
360 Y1=Y-INT(SIN(W*π/90)*RE)
```

oder

```
340 X2=X+INT(COS(W*π/180)*RE*2)
```

oder

```
330 X1=X+INT(COS(W*π/180*2)*RA)
340 X2=X+INT(COS(W*π/180*2)*RE)
350 Y1=Y-INT(SIN(W*π/180/2)*RA)
360 Y2=Y-INT(SIN(W*π/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 C16 mit dem Befehl SCALE. Durch diesen Befehl wird der Bildschirm feiner eingeteilt, nämlich in 1024 Punkte horizontal und vertikal. Natürlich ist das nur rein rechnerisch möglich, die Bildschirmauflösung beträgt weiterhin »nur« 320 x 200 Punkte.

Einschalten der Skalierung durch:

```
SCALE 1
```

Ausgeschaltet wird die Skalierung 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. Drücken Sie nun 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 Skalierung eingeschaltet. Der Kreis wird dann viel kleiner dargestellt.

Um auf dem skalierten Bildschirm genau die gleichen Koordinaten zu erreichen, wie im Normalmodus, müssen wir die X- und Y-Koordinaten umrechnen:

X x 3,2 = skalierter X-Wert

Y x 5,12 = skalierter Y-Wert

X x 6,4 = skalierter 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 Skalierung einschalten, bevor die Uhr gezeichnet wird. Fügen Sie also die Programmzeile 205 ein:

```
205 SCALE 1
```

15.6 Der Grafikbefehlssatz

Einige Befehle des C16 haben wir nun schon in unseren Beispielprogrammen kennengelernt. Der gesamte Befehlssatz ist aber noch sehr viel umfangreicher. Für die Grafik gibt es viel starke Befehle. Uns ist aufgefallen, daß das Handbuch des C16 im Hinblick auf den Grafikbefehlssatz sehr unübersichtlich ist.

Wir möchten daher an dieser Stelle alle Befehle des C16 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 Infomationen 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 nun folgenden Beschreibung werden die Parameter als Buchstaben angegeben. Wir gebrauchen hier die gleichen Bezeichnungen wie in Ihrem C16-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 mit angegeben 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.

Der Befehl COLOR

Mit COLOR können wir die Farben unserer Grafik bestimmen.

Schreibweise:

`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 C16 ansehen. Gleichzeitig sehen Sie hier, 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 Reverse-Darstellung wird durch CHR$(18) eingeschaltet.

Zeile 100 schaltet wieder die normale Farbe ein.

Das C16-BASIC bietet Ihnen zwei Befehle, die die momentan eingestellte Farbe der Farbzonen feststellen können.

Die Befehle RCLR und RLUM

RCLR (Farbzone)

Die Klammern müssen mit eingegeben werden. Durch diesen Befehl erfahren Sie die Farbnummer der angegebenen Fabzone. 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

Schreibweise:

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) gezeichnet werden oder, bei fü = 1 wird das Rechteck ausgemalt.

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$!

Schreibweise:

`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. Vielleicht haben Sie schon einmal etwas von dem Befehl PRINT AT gehört. Der C16 kennt PRINT AT nicht. Aber unser CHAR-Befehl ist fast genauso gut. 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 einfach 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.

Hierzu haben wir einen kleinen Trick, der dieses Problem beseitigt:

Positionieren Sie doch mit dem CHAR-Befehl einfach den Cursor an die gewünschte Stelle, und arbeiten Sie 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"
```

Der Befehl CIRCLE

Mit CIRCLE können Kreise, Ellipsen, Kreissegmente oder Vielecke gezeichnet werden.

Schreibweise:

```
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 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 herm 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 mit angegeben 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.

So, genug der Beispiele. Sie sehen auf jedem 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 Vordergrund- und Hintergrundfarben 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, aber nur mit erhöhtem Programmieraufwand.

Der Befehl DRAW

Diesen Befehl haben wir schon einigemale gesehen. Mit DRAW lassen sich Linien in der hochauflösenden Grafik zeichnen.

Schreibweise:

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 gleich mehrere Linien zeichnen.

Das folgende Programmbeispiel schreibt in großen Buchstaben »C16« auf den Bildschirm.

```
10 GRAPHIC1,1
20 DRAW,80,10TO10,50TO10,150TO80,199
30 DRAW,100,50TO160,0TO160,199
40 DRAW,300,0TO250,100TO250,199TO300,199TO300,100TO250,100
```

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«.

Schreibweise:

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, sodaß sie sich farblich vom Hintergrund abheben.

Beispiel:

```
10 GRAPHIC1,1
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 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 bei dabei um Rechteckformen. Ein Shape ist eine rechteckige Fläche des Grafikbildschirmes, die als Stringvariable im Computerspeicher abgespeichert wird. Diese Stringvariable kann nun jederzeit wieder ausgelesen werden, sooft 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:

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 den Zeilen 10, 20 und 30 setzen wir die Farben und schalten die Grafik ein.

In den Zeilen 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 nun 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 Bildschirmes, so oft wir möchten, 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

Schreibweise

`SSHAPE Stringvariable,x1,y1(,x2,y2)`

Stringvariable	kann jeden Variablennamen annehmen (bis auf die Systemvariablen).
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 Grafikbildschirmes 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 Bytes 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 - theoretisch, wohlgemerkt. 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 die einzelnen Punkte abgespeichert sind, 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 gebrauchen 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 C16 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 SSHAPE:

```
10 YE=20
20 GRAPHIC1
30 TRAP60
40 SSHAPE 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 aber 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. 2000/30, also etwa 66 Punkte lang sein. Das probieren wir nun 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 nun alles richtig läuft, müßte der Computer jetzt einen

```
?STRING TOO LONG ERROR
```

ausgeben. Unser Shape ist also zu lang. Wir benutzen nun die Cursortasten , gehen auf die Befehlszeile (SSHAPE...) und ändern die letzte Zahl. Ändern wir die Zahl zunächst in 63. Drücken Sie nun 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.

Schreibweise:

```
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	reverse
Modus 2	Wiedergabe ODER-Verknüpft
Modus 3	UND-Verknüpft
Modus 4	EXCUSIV-ODER-Verknüpft

Erinnern Sie sich bitte an die logischen Verknüpfungen im vierten 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 nun 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 unverändertem 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.

Nun, 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 C16, 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 Speicherstellen von einer Grafik »belegt« werden.

Der C16 hat bekanntlich leider nur 16 KByte RAM. Wenn nun eine Grafikseite 320 x 200 Bildpunkte hat, so werden dafür 64000 Bit RAM benötigt. Jeweils acht Bit ergeben ein Byte, also wären das genau 8000 Bytes. Zusätzlich werden noch 2048 Byte für die Farbinformationen benötigt. Zusammen sind also über 10000 Byte für die Grafik »verbraucht«. Für das BASIC bleibt lediglich noch ein Speicherbereich von 2048 Byte übrig.

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 gebrauchen wir das eingebaute Monitorprogramm.

Wir starten dieses Programm mit dem BASIC-Befehl:

```
MONITOR
```

In dem 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 als des zu speichernden Bereichs plus 1.

Um den gesamten Grafikspeicher abzuspeichern (z.B. auf Disk), 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 Kassettenrekorder 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 unbedungt 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 C16

Wir werden Ihnen nun anhand eines Programms die ausgezeichneten Grafikfähigkeiten des C16 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.

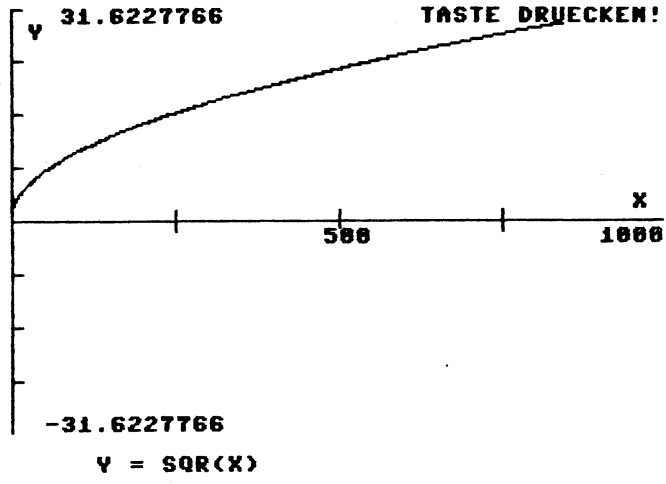


Bild 15.3

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 evtl. 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 nun den Maximalwert für Y und teilt daraufhin den Bildschirm entsprechend ein, sodaß das Bildschirmformat (200 Punkte in Y-Richtung) immer optimal genutzt wird.

```
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
235 TRAP 520
240 FORF=0TOXMSTEPXM/320
250 X=F/180*π
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
335 TRAP 525
340 FORF=0TOXMSTEPXM/320
350 REM UMRECHNUNG IN DAS BOGENMASS
360 X=F/180*π
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 280
525 RESUME 390
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 Bewandnis: 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 den 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 Speicherstelle 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 ausgelistet. 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 entweder Zeile 500 oder Zeile 510 weglassen.

Zeile 510 beendet das Programm (kann weggelassen werden).

Durch RESUME, 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

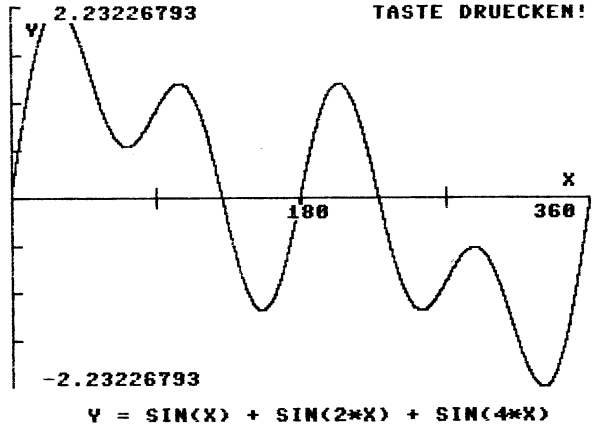


Bild 15.3

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.

Und nun viel Spaß mit dem Programm.

16
Disketten-Programmierung

Sind Sie glücklicher Besitzer eines Diskettenlaufwerkes? Wenn ja, dann freuen Sie sich, daß Sie die langen Wartezeiten beim Laden und Speichern mit dem Kassettenrekorder nicht mehr haben. Lesen Sie aber trotzdem dieses Kapitel, damit Sie auch richtig mit Ihrem Gerät umgehen. Sollten Sie dagegen kein Diskettenlaufwerk (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 näher bringen.

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 Kunststofffolie, die mit einem magnetischem 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.

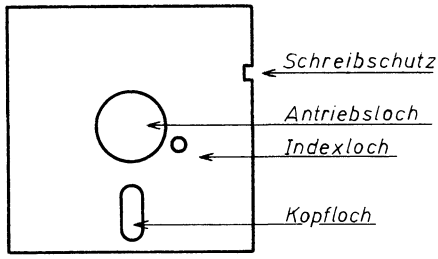


Bild 16.1

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. Das macht ein Motor im Diskettenlaufwerk. 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 Diskettenlaufwerke gedacht. Unser Commodore 1541-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-Diskettenlaufwerk 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 Führungschienen, 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 Diskettenlaufwerk. 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-Laufwerk ist außerdem noch eine umfangreiche Elektronik untergebrach. Es handelt sich dabei um einen kompletten Computer, der die Motoren und die Verarbeitung der Daten steuert.

Der Computer im 1541-Laufwerk hat, genau wie der C16, ein ROM. In diesem Speicher ist das sogenannte DOS (Disk Operating System) gespeichert, ein Programm, das das Laufwerk erst zum Leben erweckt.

16.2 Diskettenlaufwerk kontra Kassettenrekorder

Kassettenrekorder haben einen großen Vorteil gegenüber Diskettenlaufwerke, sie sind nämlich sehr viel billiger. Das wars 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,

wenn Sie z.B. die letzten Teile dieser Daten lesen möchten. 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. Dies 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 die gewünschte 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 Diskettenlaufwerk immer, wo die Daten zu finden sind.

Ein weiterer Vorteil der Disketten gegenüber Kassetten ist die höhere Datenübertragunggeschwindigkeit bei 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 30 Sekunden. Man kann sagen, daß die Verarbeitungsgeschwindigkeit der Diskette etwa 10 mal schneller ist als bei Kassetten.

Und noch ein Vorteil der Disketten:

Mit dem Diskettenlaufwerk 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 Diskettenlaufwerk 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 Kassettenrekorder unmöglich.

Stellen Sie sich einmal vor, Sie haben ein 50 KByte langes Programm auf Kassette und möchten dieses einladen (der C16 braucht dafür natürlich eine Speichererweiterung!) - Sie können während der Ladezeit ruhig eine Tasse Kaffee trinken gehen, denn Sie haben sehr viel Zeit!

Zugegeben, ein so langes Programm von der Diskette einzulesen dauert auch etwa 150 Sekunden, ist aber um den Faktor 10 schneller.

Wenn Sie ernsthaft mit Ihrem Computer arbeiten möchten, kommen Sie nicht umhin, ein Diskettenlaufwerk zu kaufen.

Für den Anfänger dagegen reicht ein Kassettenrekorder 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 C16 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. Mit dem Diskettenlaufwerk können Sie außerdem auch »relative 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 davon gelesen wird, »geöffnet« werden. Das Diskettenlaufwerk 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 Befehlskanäle Ihres Diskettenlaufwerks oder anderer Ein- und Ausgabegeräte.

Schreibweise:

```
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 Kassettenrekorder oder Diskettenlaufwerk geöffnet werden, sondern auch zum Beispiel auf dem Bildschirm. Um das zu unterscheiden, gibt es diese Gerätenummer.

k Sekundäradresse. Im Zusammenhang mit dem Diskettenlaufwerk wird hier von der Kanalnummer gesprochen.

16.4.1 Die Gerätenummer

0 = Tastatur
1 = Kassettenrekorder
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 die 3 und einmal die 4 als Gerätenummer eingesetzt wird.

16.4.2 Die Sekundäradresse

Die Sekundäradresse wird z.B. bei dem Kassettenrekorder 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 C16 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 Datei verlängert (A = append, anfügen).

Wir haben nun gesehen, wie eine Datei geöffnet wird. Nach dem Arbeiten mit der Datei muß diese wieder geschlossen weeden. 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

Wir haben unsere Datei nun geöffnet. Wie bekommen wir denn nun unsere Daten in die Datei hinein?

Genauso, wie wir Daten auf dem Bildschirm mit PRINT ausgeben können, können Daten auch mit dem PRINT-Befehl zur 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<>64THENGOTO20
50 CLOSE1
60 END
```

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 einzelne Daten ja evtl. auch aus mehreren Zeichen bestehen können. Wenn also GET# verwendet wird, muß erkannt werden, wann der Datensatz zuende 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 OPEN1,8,2,"ADRESSEN,S,R"
20 GET#1,X$
30 IFX$=CHR$(13)THENPRINTA$:A$="":GOTO20
40 A$=A$+X$
50 IFST<>64THENGOTO20
60 CLOSE1
70 END
```

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.

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.

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 SPIELEINS 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. Wenn Sie auf Ihrer Diskette z.B. ein Programm mit dem Namen »PLOTTEN« haben. Nehmen wir nun an, Sie hätten das Programm 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 nun mit dem neuen überschrieben. Wenn Sie sequentielle Files überschreiben möchten, so gebrauchen 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 C16-Handbuch das komplette Diskettenlaufwerk

ausführlich zu beschreiben. Wir beschränken uns daher hier auf das Wesentliche.

16.8 Die Diskettenbefehle

Der C16 erleichtert das Arbeiten mit dem Diskettenlaufwerk durch einen Reihe von BASIC-Befehlen:

```
HEADER                    DIRECTORY
DSAVE                     DLOAD
BACKUP                    COLLECT
COPY                      SCRATCH
VERIFY
```

Zusätzlich zu den speziellen Diskettenbefehlen gebrauchen wir noch weitere Befehle zum Umgang mit dem Diskettenlaufwerk:

```
OPEN                      PRINT#
GET#                      INPUT#
CMD                       CLOSE
```

Diese Befehle haben wir schon kennengelernt.

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 gebrauchen wir den BASIC-Befehl OPEN.

Der Befehl OPEN

Schreibweise:

```
OPEN f,g,15,"Befehl:Filename..."
```

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 evtl. nötig, wenn Sie zwei Laufwerke angeschlossen haben. Bei einem normalen 1541-Laufwerk ist diese Nummer eine »8«.
15	Der sogenannte Befehlskanal des Diskettenlaufwerks.
"Befehl:Filename..."	Der gewünschte Systembefehl kann voll ausgeschrieben 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 Filename, angegeben.

Wir werden bei den einzelnen Befehlen die Schreibweise des OPEN-Befehls jeweils mitangeben.

Der Befehl HEADER

Wenn Sie eine neue Diskette kaufen (nicht neue Disketten, auf denen bereits Programme abgespeichert sind!), 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 Ug = 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 Ug kann auch nur Ug 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 Informationen, 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 Inhaltverzeichnis, das Direktory, der Diskette angelegt.

Außerdem wird der momentane Inhalt der Dikette 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.

Nun 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 Diskettenlaufwerk falsch.

Der Befehl DIRECTORY

Mit diesem Befehl können Sie sich das Inhaltsverzeichnis der Diskette ansehen. Sie brauchen nur DIRECTORY einzugeben, ohne Gerätenummer oder sonstige zusätzliche Informationen.

Schreibweise:

```
DIRECTORY (D0) (,U8) (,"Filename")
```

D0 und U8 können 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-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 C16-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 C16 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.

Schreibweise:

```
DSAVE "Filename"(,D1)(,U9)
```

D1 und U9 können Sie wieder weglassen, wenn Sie nur ein Laufwerk angeschlossen haben.

Der Filename, also der Programmname 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 Diskettenlaufwerk übertragen.

Der Befehl DLOAD

Schreibweise:

```
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 Speicherstellen 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 dem 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 müßte erneut das Maschinenprogramm geladen werden.

Nun, zum Glück ist das nicht so, denn die Inhalte von Variablen werden in diesem Fall nich 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 der Wert 1 zum Inhalt von X 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-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 evtl. vorhandenen Daten auf dieser Diskette 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 längeren Gebrauch einer Diskette kann es passieren, daß die Summe der belegten und freien Blöcke im Directory nicht mehr die mögliche Gesamtzahl von 664 ergeben. 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.

Schreibweise:

```
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«. Die Gerätenummer ist 8.

```
COPY "GRAFIK" TO "MALPROGRAMM"
```

Kopiert das File »GRAFIK« auf derselben Diskette 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-Laufwerk angeschlossen haben, können Sie diese Angabe weglassen.

g Geräteadresse, normalerweise 8. Wenn Sie nur ein 1541-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 gar nicht »gelöscht«. Es wird nur im Directory der Diskette ein Byte geändert, und im BAM der Diskette werden die durch das File belegen 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« wird, ist es nicht mehr zu retten, obwohl die Daten dieses Files noch auf der Diskette verblieben sind.

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 agbespeichert 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, mäglichst 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.

Schreibweise:

```
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-Floppy normalerweise 8.

id Die Identifizierungsnummer der Diskette.

Wenn Sie die ID angeben, wird die Diskette föllig 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.

Schreibweise:

```
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 Filename 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.

Schreibweise:

```
OPEN f,g,15,"R:neuer Name=alter Name"
CLOSE f
```

Statt R darf RENAME selbstverständlich voll ausgeschrieben werden. Der neue Filename muß zuerst angegeben werden, dann folgt, durch ein Gleichheitszeichen getrennt, der alte Filename.

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.

Schreibweise:

```
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:neuerName1=alterName1,alterName2,...
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 Diskettenlaufwerk, die BAM einer Diskette in den RAM-Speicher des Laufwerks zu laden. Lesen Sie bitte dazu den Abschnitt über die ID (Identifizierungsnummer der Disketten).

Schreibweise:

```
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 evtl. in fremden Programmen einmal auf VALIDATE stoßen könnten.

Schreibweise:

```
OPEN f,g,15,"V" (oder "VALIDATE").
CLOSE f
```

oder

```
OPEN f,g,15
PRINT#f,"V"
CLOSE f
```

16.10 System-Variablen DS und DS$

Diese zwei reservierten Variablen geben den Zustand des Diskettenbefehlskanals an. Die Variable DS gibt dabei die Fehlernummer an. DS$ beinhaltet die Fehlernummer, den Fehlertext, Spur und Sektor.

Sie können die Variablen sowohl im Direktmodus als auch im Programmodus abfragen. Das eignet sich hervorragend für die Fehlerbehandlung mittels TRAP. Sie können so z.B. feststellen, ob eine Diskette eingelegt werden muß, ob eine Diskette voll ist oder ein File eventuell nicht vorhanden ist. Das Programm würde dann nicht abbrechen.

Diese reservierten Variablen ersetzen das umständliche Öffnen und Auslesen des Fehlerkanals der Floppy, wie es bei C64 und VC-20 nötig war.

Auf eine detaillierte Fehlerbeschreibung der einzelnen Fehlernummern möchten wir hier verzichten, da eine dementsprechende und sehr ausführliche Tabelle im Bedienungshandbuch zum C16/116 und Plus/4 enthalten ist.

17
Direkter Speicherzugriff mit PEEK, POKE, WAIT

Mit diesen Befehlen haben Sie die Möglichkeit, den Speicherinhalt Ihres C16 anzusehen oder zu ändern. Dabei dient der Befehl PEEK zum Auslesen des Speicherinhalts.

Der Befehl PEEK

Schreibweise:

```
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 C16 für wichtige Daten reserviert ist.

Nun werden wir den Inhalt einer Speicherzelle ändern. Dazu gebrauchen 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.

Nun probieren Sie aber einmal folgendes aus (ein kleiner Tip, speichern Sie vorher das Programm ab, das Sie evtl. 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 weiteres übrig, als den Computer auszuschalten und wieder einzuschalten.

Dieses Beispiel soll Ihnen zeigen, was Sie mit dem POKE-Befehl »anrichten« können. Es ist eine Kleinigkeit, den den Computer lahmzulegen. Seien Sie daher sehr vorsichtig mit diesem Befehl!

Ein weiteres Beispiel:

Geben Sie bitte folgendes Miniprogramm ein und starten Sie es mit RUN.

```
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 1000*5, sondern 1000/5 (1000 dividiert durch 5).

Unser POKE-Befehl hat also das Programm geändert. Nun ist das 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, vorausgesetzt, Sie gebrauchen eine Speicherzelle, die nicht vom Betriebssystem benötigt wird, dagegen unverändert. Für Ihre Daten können Sie z.B. die Speicheradressen 1630 bis 1771 benutzen, weil dieser Speicherbereich in der Grundversion Ihres C16 frei ist.

PEEK und POKE können auch kombiniert werden. Das ist notwendig, wenn Sie in einer Speicherzelle nur ganz bestimmte Bits verändern möchten. Dabei müssen Sie die Speicherzelle erst lesen, dann den Wert ändern und wieder in die Speicherzelle zurückschreiben.

Wir werden nun einmal den Bildschirm ausschalten:

```
POKE65286,PEEK(65286)AND239
```

Zack, der Bildschirm ist ausgeschaltet. 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 bestimmt, 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 C16 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.

Der Befehl WAIT

Ein weiterer Befehl, mit dem Sie einen direkten Zugriff auf den Speicher haben, ist der WAIT-Befehl. Diesen Befehl haben wir uns schon im vorherigen Kapitel angesehen, daher hier nur kurz die

Schreibweise:

```
WAIT Adresse,Wert1(,Wert2)
```

Mit Wait ist es möglich, ein Programm solange anzuhalten, bis eine bestimmte Speicheradresse einen bestimmten Wert annimmt.

Lesen Sie zum Befehl WAIT auch das vorherige Kapitel. 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 einem Beispiel an:

```
10 FOR I=1 TO 10
20 READ T(I)
30 NEXT I
60 FOR I=1 TO 10
70 PRINT T(I)
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, zugeordnet. 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 Speicherstellen, 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 Programmes 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. Das Leerzeichen würde überlesen.

18.3 Der Befehl RESTORE

Syntax:

`RESTORE`

`RESTORE Zeilennr.` oder **`Variable`**

Sollen DATA-Elemente mehrmals gelesen werden, kann mit RESTORE der DATA-Zeiger wieder im Programm aufwärts bewegt weden. 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 Maschinensprache 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 offenbart. Wer ein übersichtliches Programm mit wenigen Programmverzeigungen und vielen REM-Zeilen geschrieben hat, wird man 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 C16/116 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.

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 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 und evtl. Sprungfehler vorliegen. 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 eine 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.

So, nun sollte es aber wirklich möglich sein, dasProgramm fehlerfrei zum Laufen zu bringen. Lesen Sie sich sonst ruhig noch einmal die Kapitel über die Befehle, die Ihnen noch nicht so ganz geheuer sind, durch.

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 evtl. 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 und der Rücksprungbefehl selbst müssen fehlerfrei sein. Der Rücksprungbefehl heißt hier RESUME.

Syntax:

RESUME

oder

RESUME NEXT

oder

RESUME Zeilennummer

RESUME bewirkt den Rücksprung zu dem den Fehler verursachten 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 nun 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.

Nun 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 nun 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 »SYNTAXERROR IN Zeile«. In Zeile 120 wird das Programm beendet.

Mit TRAP....RESUME läßt sich sogar die **STOP**-Taste ausschalten.

Verlängern wir eimal das Programm:

```
105 IF ER = 30 THEN RESUME
```

Wird jetzt die **STOP**-Taste gedrückt, so verweigt das Programm zur Zeile 100. Die Bedingung in Zeile 105 wird erfüllt, also erfolgt mit RESUME der Rücksprung an die Stelle des Programmes, an der es unterbrochen wurde. So, nun haben Sie einen Dauerläufer, den Sie nicht mehr unterbrechen können; oder doch?

Drücken Sie die **RESET**-Taste und halten sie gedrückt. Nun müssen Sie noch gleichzeitig die **COMMODORE**- und die **RUN/STOP**-Taste drücken. Lassen Sie jetzt die **RESET**-Taste los und dann erst die anderen beiden Tasten. 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 nun 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 nummerischer 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 C16

Da wir uns in diesem Buch hauptsächlich mit der Programmierung in BASIC beschäftigen, soll die Maschinensprache auch nur kurz angesprochen werden. Im C16 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 so viele, 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 veranlaßt. 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 jeweils 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-Pogrammen, 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 Maschinensprachbefehle gebrauchen 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 Interprter 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 gebrauchen wir den Maschinensprachmonitor, im C16 TEDMON genannt.

20.1 Der TEDMON

Dieses Programm ermöglicht es, Maschinensprachprogramme 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	= Disassemblieren
F	= Füllen
G	= Starten
H	= Suchen
L	= Laden
M	= Speicherinhalt listen
R	= Register anzeigen
S	= Speichern
T	= Verschieben
V	= Vergleichen eines Programms mit Programm auf Kass./Diskette
X	= Programm beenden
>	= Speicherinhalte ändern
;	= Prozessorregister ändern

TEDMON-Befehl A

Mit diesem Befehl können Sie Maschinensprachprogramme in der Assemblerschreibweise eingeben.

Schreibweise für A:

```
A Adresse Befehl (Operand)
```

Adresse	Speicheradresse (hexadezimal), an der der Befehl gespeichert werden soll.
Befehl	Assemblerschreibweise des Befehls.
Operand	Es gibt drei Arten Maschinensprachbefehle: 1-Byte-, 2-Byte- und 3-Byte-Befehle. Bei 1-Byte-Befehlen darf kein Operand eingegeben werden. Bei 2-Byte-Befehlen muß ein Operand 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 (.) esetzt 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 das »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 Bildschim 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 Maschinensprachprogramme 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 gebrauchen können, in einem Speicherbereich nach bestimmten Daten zu suchen.

Schreibweise:

```
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. 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 Bytefolge $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 Kassettenrekorder und 8 für das Diskettenlaufwerk. Wenn vom Kassettenrekorder 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 Bytes ausgegeben. Dann folgen acht Bytes und dann die entsprechenden CHR$-Codes der acht Bytes (in reverser Schrift).

Schreibweise:

```
M (Anfangsadresse) (Endadresse)
```

Anfangsadresse — Wird nur diese Adresse eingegeben, werden 96 Bytes 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 doch einmal 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 Bytes ausgegeben. Und tatsächlich sind die ersten drei Bytes 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 >

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 Bytes)

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 Statusregister SR, der Accu AC, das X-Register XR, das Y-Register YR und der Stapelzeiger SP.

Ändern der Prozessorregister mit ;

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 Cursortasten 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 1 zu addieren, sonst würde das letzte Byte fehlen.

Der TEDMON-Befehl T

Mit diesem Befehl kann der Inhalt eines Speicherbereiches ineinen 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 C16 verglichen. Wird dabei ein Unterschied festgestellt, wird die Meldung »ERROR« ausgegeben.

Schreibweise:

```
V "Filenamen",g
```

Filenamen Unter diesem Namen ist das Programm abgespeichert worden.

g Gerätenummer, also die 1 für Kassette und die 8 für Diskette.

Rückkehr aus TEDMON

Mit dem Befehl X können Sie wieder in das BASIC Ihres C16 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 Maschinensprachprogramme von BASIC aus starten. Das funktioniert sowohl im Direktmodus, wie auch im Programmmodus. Wir haben dadurch also die Möglichkeit, Maschinensprachroutinen 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 evtl. aufhängen könnte.

Wir können dem Maschinenprogramm keine Werte übergeben. Es ist also nicht möglich, evtl. 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 also ebenfalls möglich, Maschinensprachprogramme zu starten. Diesen Programmen können auch Werte übergeben werden, damit weitere Berechnungen durchgeführt werden können.

Schreibweise:

`USR (Variable)`

Sie sehen, hier wird keine Staradresse eingegeben. Die Adresse ist aber unbedingt nötig. Deshalb muß sie schon vor diesem Befehl festgelegt werden. Die Startdresse wird hierbei in die Speicheradressen 1281 und 1282 abgespeichert. Das geschieht von BASIC aus mit dem Befehl POKE. Da eine Startadresse 16 Bit hat, in die Speicherstellen aber nur je acht Bit gespeichert werden kann, 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-Accu 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-Accu 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 die Literatur über Maschinensprache.

Hiermit ist unser Kapitel »Maschinensprache mit dem C16« beendet. Wir verweisen in diesem Zusammenhang auch auf die Speicherbelegung des C16, die wir im Anhang angeben.

Anhang A
Die BASIC-Fehlermeldungen

Der C16/116 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 C16 können max 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 zweitenmal öffnen. Sollte ein neues File geöffnet werden, dann geben Sie ihm eine andere Filenummer.
3	FILE NOT OPEN	Wenn Sie versuchen, aus einer nicht geöffneten Datei zu lesen oder in eine solche zu schreiben, erfolgt diese Meldung. Abhilfe siehe OPEN-Befehl.

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 übermitteln 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 verwendet 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 vorhan-

	den 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 max. 255) oder »POKE 3400,-1« (erlaubt wäre min. 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 zuviele Variablen oder zu große Variablenfeldern definiert. Tritt auch auf, wenn zuviele Schleifen geöffnet, Funktionen definiert oder zuviele 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 verwendet 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 verwendet werden darf, zum Beispiel DELETE.
35 NO GRAPHICS MODE	Es wurde auf einen Grafikbefehl getroffen, ohne vorher mit GRAPHIC einen Grafikmodus einzuschalten.
36 BAD DISK	

Den Fehlernummern 32 und 36 folgt keine Erklärung, weil sie bei unseren Arbeiten mit dem C16/116 nie auftraten.

Anhang B
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. Den 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
abs	aB	Rechnen
and	aN	Logik
asc	aS	Strings
atn	aT	Rechnen
auto	aU	Prg.Hilfe
backup	bA	Disk
box	bO	Grafik
char	chA	Grafik/Text
chr$	cH	Strings
circle	cL	Grafik
close	clO	Disk/Kass
clr	cL	Variablen
cmd	cM	Ausgabeger.
collect	colL	Disk
color	coL	Farbe
cont	cO	Prg.Stop
copy	coP	Disk
data	dA	Programm
end	eN	Programm
def	dE	Rechnen
delete	deL	Löschen
dim	dI	Felder
direktory	diR	Disk
dload	dL	Disk
draw	dR	Grafik
dsave	dS	Disk
err$	eR	Variable
exp	eX	Rechnen
for	fO	Schleifen
fre	fR	Speicher
get	gE	Eingabe
getkey	getkE	Eingabe
get#	gE#	Disk/Kass
gosub	goS	Prg.Sprung
goto	gO	Prg.Sprung
graphic	gR	Grafik
gshape	gS	Grafik
header	heA	Disk
hex$	hE	Zahlen
input#	iN	Disk/Kass
instr	inS	Strings
joy	jO	Eingabe
key	kE	Tastatur
left$	leF	Strings
let	lE	Variablen
list	lI	Prg.Hilfe
load	lO	Disk/Kass
locate	loC	Grafik
loop	loP	Schleifen
mid$	mI	Strings

Name	Abk.	Bezug
monitor	mO	Masch.spr.
next	nE	Schleifen
not	nO	Logik
open	oP	Peripherie
paint	pA	Grafik
peek	pE	Speicher
poke	pO	Speicher
print	?	Ausgabe
print#	pR	Peripherie
printusing	?usI	Ausgabe
pudef	pU	Printusing
rclr	rC	Grafik
rdot	rD	Grafik
read	rE	Data
rename	reN	Disk
renumber	renU	Prg.Hilfe
restore	reS	Data
resume	resU	Fehler
return	reT	Prg.Sprung
rgr	rG	Grafik
right$	rI	Strings
rlum	rL	Grafik
rnd	rN	Zahlen
run	rU	Prg.Start
save	sA	Disk/Kass
scale	scA	Grafik
scnclr	sC	Grafik
scratch	scR	Disk
sgn	sG	Zahlen
sin	sI	Rechnen
sound	sO	Musik
spc(	sP	Ausgabe
sqr	sQ	Rechnen
sshape	sS	Grafik
stop	sT	Programm
str$	stR	Strings
sys	sY	Masch.Prg
tab(	tA	Ausgabe
trap	tR	Fehler
troff	troF	Fehler
tron	trO	Fehler
until	uN	Schleifen
usr	uS	Masch.Prg
val	vA	Strings
verify	vE	Disk/Kass
vol	vO	Musik
wait	wA	Speicher
while	wH	Schleifen

Anhang C
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	DIRECTORY
140	RESTORE	173	/	206	RLUM	239	DSAVE
141	GOSUB	174	^	207	JOY	240	DLOAD
142	RETURN	175	AND	208	RDT	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 D
Bildschirmaufbau

Wir zeigen Ihnen nun zwei Bilder, die Ihnen den Aufbau des Bildschirmspeicher und des Farbspeichers verdeutlichen sollen.

Die Zahlen auf der linken Seite geben jeweils die Anfangsadresse der jeweiligen Zeile an.

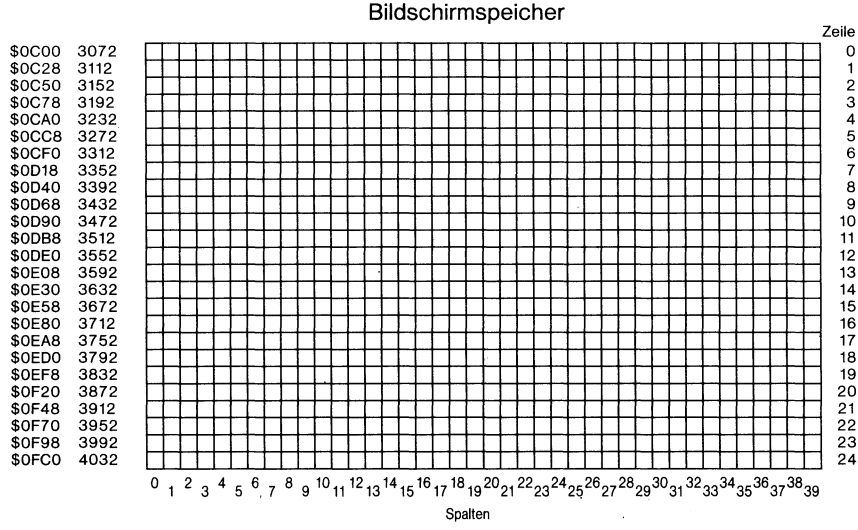


Bild D.1: *Adressbereich des Bildschirmspeichers*

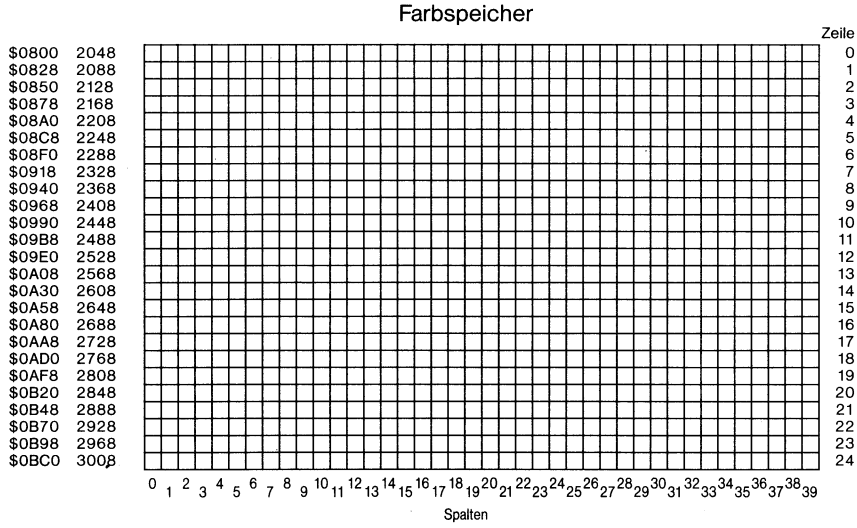


Bild D.2: *Adressbereich des Farbspeichers*

Anhang E
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 verwendet. Dabei bedeuten:

Farbnr.	Farbe	Farbnr.	Farbe	Farbnr.	Farbe
1	Schwarz	7	Blau	13	Blau/Grün
2	Weiß	8	Gelb	14	Hellblau
3	Rot	9	Orange	15	Dunkelblau
4	Cyan	10	Braun	16	Hellgrün
5	Purpur	11	Gelb/Grün		
6	Grün	12	Rosa		

Diese Farben können dann auch noch in der Helligkeit verändert werden. Dies entspricht dem dritten Parameter des COLOR-Befehls. Es dürfen Zahlen von 0 bis 7 verwendet werden.

0 entspricht dem dunkelsten Farbton, 7 dem hellsten. Die anderen Zahlen entsprechen Zwischenwerten.

Wird in das 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 :

Farbnr.	Farbe	Wert für Helligkeitsstufe							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	Schwarz	0	16	32	48	64	80	96	112
1	Weiß	1	17	33	49	65	81	97	113
2	Rot	2	18	34	50	66	82	98	114
3	Cyan	3	19	35	51	67	83	99	115
4	Purpur	4	20	36	52	68	84	100	116
5	Grün	5	21	37	53	69	85	101	117
6	Blau	6	22	38	54	70	86	102	118
7	Gelb	7	23	39	55	71	87	103	119
8	Orange	8	24	40	56	72	88	104	120
9	Braun	9	25	41	5	73	89	105	121
10	Gelb/Grün	10	26	42	58	74	90	106	122
11	Rosa	11	27	43	59	75	91	107	123
12	Blau/Grün	12	28	44	60	76	92	108	124
13	Hellblau	13	29	45	61	77	93	109	125
14	Dunkelblau	14	30	46	62	78	94	110	126
15	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 F
CHR$-Kodes

Wie zeigen Ihnen nun eine Tabelle mit allen CHR$-Kodes Ihres C16. Sie erhalten das entsprechende Zeichen, wenn Sie den Befehl CHR$(x) verwenden. Es handelt sich dabei nicht nur um Zeichen, sondern auch um die sogenannten Steuerzeichen, z.B. Carriage Return.

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	π
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 F.1

Die dargestellten Zeichen erscheinen auf Ihrem Bildschirm so, wie in der Tabelle abgebildet, im Großschrift/Grafik-Modus. Dies ist der Einschaltzustand. Wenn Sie auf Klein/Großschrift umschalten, z.B. mit »PRINT CHR$(14)« oder durch Drücken der **SHIFT/COMMODORE**-Taste, erscheinen für die CHR$-Kodes 65 bis 90 die entsprechenden Kleinbuchstaben. Für die CHR$-Kodes 97 bis 122 werden dann Großbuchstaben dargestellt. Außerdem erscheinen für die Kodes 126, 127, 169, 186, 233, 250 und 255 andere Grafikzeichen.

Für das Space (Leertaste) gibt es zwei verschiedene Kodes, 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$-Kodes der gedrückten Tasten erhalten. Das läßt sich sehr gut in Programmen verwenden, wenn einzelne Tasten ausgewertet werden sollen. Sie können in unserem Beispielprogrammen in diesem Buch sehen, daß wir davon Gebrauch machen.

Nun noch eine interessante Möglichkeit für die Tastaturabfrage: Es besteht nämlich die Möglichkeit, durch Tastenkombinationen der **CTRL**-Taste mit eine Buchstabentaste, die CHR$-Kodes von 1 bis 29 zu bekommen. Auch das läßt sich sinnvoll in Programmen einsetzen. Sie erhalten so eine Menge neuer »Funktionstasten«. Z.B ergibt **CTRL/K** den CHR$-Wert 11. In der folgenden Tabelle zeigen wir Ihnen alle Tastenkombinationen und die dazugehörigen Kodes.

Doch zuvor noch etwas: Im sogenannten Anführungszeichen-Modus ist es möglich, die CHR$-Kodes von 1 bis 29 durch reverse Buchstaben darzustellen. Sie müssen dazu nur ein Anführungszeichen eintippen und dann die entsprechende Tastenkombination mit **CTRL/Buchstabentaste**. Bei den Kodes 0, 10, 13, 19, 20 und 27 funktoniert das nicht ganz so einfach. Bei diesen Kodes müssen Sie den entsprechenden reversen Buchstaben im Reverse-Modus in Ihr Programmlisting eingeben. Hier nun die Tabelle.

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 F.2

Anhang G
Wichtige Speicheradressen des C16/116

Die ersten 4072 Bytes stehen dem Programmierer für BASIC-Programme nicht zur Verfügung. In diesen Speicherzellen werden vom Betriebssystem selbstständig 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 Standard-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
$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	Temporärer Datenspeicher
$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
$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)	
$273	627	DOS (Laufwerk 2)	
$0274-0275	628-629	2. Filename (Adresse)	
$0276	630	DOS logische Adresse	
$0277	631	DOS (Geräteadresse)	
$0278	632	DOS (Sekundäradresse)	
$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)
$031C-031D	796-797	Sprungvektor: Kanal für Egbe.öffnen	($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	
$0455-0472	1109-1138	Systemarbeitsbereich	
$0473-0478	1139-1144	Chrget Routine	

$0455-0484	1145-1156	Chrgot Routine
$0494-04DC	1172-1244	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 es 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
$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
$07E5	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:

$1000-3FFF	4096-16383	BASIC-RAM des C16/116

Belegung des RAMs nach Aufruf der Grafik:

$1000-17FF	4096-6143	BASIC-RAM des C16/116
$1800-1BFF	6144-7167	Luminanztabelle (Grafik)
$1C00-1FFF	7168-8191	Farbtabelle (Grafik)
$2000-3FFF	8192-16383	Grafikbildschirm

Anhang H
Wichtige Register des 7360 (TED)

Im C16 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 (max. 64KBytes), 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 einige hier 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 Zeilen, 1 = 25 Zeilen
		Bit 4: 0 = Bildschirm aus, 1 = Bildschirm ein
		Bit 5: 0 = HIRES aus, 1 = HIRES ein
		Bit 6: 0 = Extended-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

$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:	Rasterzeil (Bit 8 in Reg. 65290)
$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
$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
$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 I
Anschlußbelegungen

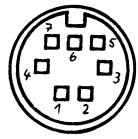
Der C16 hat eine Reihe von Anschlüssen für externe Geräte. Wir zeigen Ihnen hier nun die Steckerbelegung folgender Anschlußbuchsen:

- Datasette
- Audio/Video
- Joystick
- Seriell
- Erweiterungs-Port

Datasette

Stift	Belegung
1	Masse
2	+ 5V
3	Motor
4	Read
5	Write
6	Schalter
7	Masse

Datasette



Audio/Video

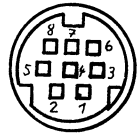
Stift	Belegung
1	Helligkeit
2	Masse
3	Audio Ausgang
4	Video Ausgang
5	Audio Eingang
6	Farbe Ausgang
7	+ 5V
8	frei

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



Joystick 2

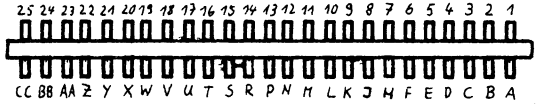
Stift	Belegung
1	Key 0
2	Key 1
3	Key 2
4	Key 3
5	+ 5V
6	Feuerknopf, Key 7
7	Masse
8	D1

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	2	+ 5V
3	+ 5V	4	$\overline{\text{IRQ}}$
5	R/$\overline{\text{W}}$	6	C1 High
7	C2 Low	8	C2 High
9	$\overline{\text{CS1}}$	10	$\overline{\text{CS0}}$
11	$\overline{\text{CAS}}$	12	MUX
13	BA	14	D7
15	D6	16	D5
17	D4	18	D3
19	D2	20	D1
21	D0	22	AEC
23	Extern Audio	24	Φ2
25	Masse		
A	Masse	B	C1 Low
C	$\overline{\text{RESET}}$	D	$\overline{\text{RAS}}$
E	Φ0	F	A15
H	A14	J	A13
K	A12	L	A11
M	A10	N	A9
P	A8	R	A7
S	A6	T	A5
U	A4	V	A3
W	A2	X	A1
Y	A0	Z	frei
AA	frei	BB	frei
CC	Masse		



Anhang K
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 äußerst rechter Stelle Bit null. Links davon steht Bit eins, Bit zwei,.....bis Bit sieben.

	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

	HEX	BINÄR	DEZ	HEX	BINÄR	DEZ	HEX	BINÄR
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
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	%01001010	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	%01110001	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

	HEX	BINÄR	DEZ	HEX	BINÄR	DEZ	HEX	BINÄR
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			

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

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

Weitere Fachbücher aus unserem Verlagsprogramm

Prof. Dr. Wolf-Jürgen Becker

CP/M-3.0-Anwender-Handbuch C128

Mai 1986, 250 Seiten

Das Buch sagt Ihnen alles über den Aufbau einer Datenverarbeitungsanlage, Mikrocomputer, Programmiersprachen und Betriebssysteme im allgemeinen und über das Betriebssystem CP/M speziell auf dem C128 PC. Ausführliche Beschreibungen der CP/M-Befehle und ihrer Funktionen fehlen ebensowenig wie die umfassende Darstellung der Struktur von CP/M-3.0 auf dem C128. Im Kapitel über das Programmieren unter CP/M erfahren Sie dann, wie man das CP/M-Betriebssystem ändert, kommerzielle Software installiert und mit ihr arbeitet.

Best.-Nr. MT 90196, ISBN 3-89090-196-4
(sFr. 47,80/öS 405,60) **DM 52,-**

H. Ponnath

Grafik-Programmierung C128

März 1986, 196 Seiten

Eine Hilfe für den Einsteiger und eine Fundgrube von Anregungen für den Profi soll dieses Buch sein. Das Themenfeld ist weit gespannt und umfaßt unter anderem:
Mehrfarbengrafik im C128-Modus; die Programmierung von Sprites und Shapes; Assembler-Makros, die die Grafikprogrammierung unterstützen; die beiden Video-Chips des C128; Erzeugung selbstmodifizierender Programme; Generierung von Fractals, ein besonders spannendes Thema im Bereich mathematischer Grafikanwendungen.
Die ungewöhnlichen Grafikfähigkeiten des C128 werden voll ausgeschöpft.

Best.-Nr. MT 90202, ISBN 3-89090-202-2
(sFr. 47,80/öS 405,60) **DM 52,-**

G. Möllmann

C128-Programmieren in Maschinensprache

August 1986, ca. 250 Seiten

Dieses Buch ist für alle diejenigen geschrieben, die die Fähigkeiten ihres Commodore 128 voll ausschöpfen wollen, um selbst erfolgreich auf dem Commodore 128 programmieren zu können. Dazu gehört außer der Beschreibung der im 128er enthaltenen Bausteingruppen auch der Umgang mit den ROM-Routinen aus Basic und Betriebssystem.

- Eine Fundgrube für jeden ernsthaften C128-Programmierer!

Best.-Nr. MT 90213, ISBN 3-89090-213-8
(sFr. 47,80/öS 405,60) **DM 52,-**

R. Schineis/M. Braun

C128-ROM-Listing: BASIC-7.0-Betriebssystem

August 1986, ca. 300 Seiten

Nach einer Einführung in die Arbeitsweise des C128 werden der interne Aufbau und die Wirkungsweise des BASIC-Interpreters erläutert. Es wird hierbei unter anderem auf die Speicher- und Variablen-Organisation sowie auf die Struktur und Ablage von BASIC-Zeilen eingegangen. Vor dem Hauptteil, einem vollständig kommentierten Assemblerlisting des C-128-BASIC-Interpreters mit Cross-Referenzliste (Verweistabelle), werden Informationen über die Struktur und Interpretation des Listings und der Verweistabelle gegeben.

Best.-Nr. MT 90220, ISBN 3-89090-220-0
(sFr. 45,10/öS 382,20) **DM 49,-**

R. Schineis/M. Braun/N. Demgensky

C128-ROM-Listing: Operating System

März 1986, 450 Seiten

Dieses Buch ist für alle Programmierer und Anwender gedacht, die mehr über ihren Commodore 128 PC wissen wollen: Eine Einführung in die Organisation und Wirkungsweise eines Mikrocomputers sowie eine detaillierte Beschreibung der Mikroprozessorfamilie 65XX bzw. 8502, Aufbau und spezielle Hardwareeigenschaften des C128 mit Beispielprogrammen. Ein umfangreiches, vollständig kommentiertes Assemblerlisting mit Cross-Referenzliste (Verweistabelle) umfaßt das komplette Betriebssystem mit dem 40/80-Zeichen-Editor sowie allen Kernel-Routinen.

Best.-Nr. MT 90221, ISBN 3-89090-221-9
(sFr. 45,10/öS 382,20) **DM 49,-**

COMMODORE AMIGA

M. Breuer

Das AMIGA-Handbuch

März 1986, 461 Seiten

Das Buch liefert übersichtlich gegliedertes Grundwissen über die neue Commodore-Maschine. Aus dem Inhalt:
Vorhang auf: Der AMIGA! · Auf der Werkbank des AMIGA · Grundlage der Bedienung des AMIGA · Grafik mit Graficraft und Delux Paint. AMIGA für Fortgeschrittene: Das CLI · Automatisierung des AMIGA · Die Spezialchips des AMIGA · Grundlagen von Sound und Grafik.

- Mit vielen Abbildungen und Übersichtstafeln für den täglichen Einsatz.

Best.-Nr. MT 90228, ISBN 3-89090-228-6
(sFr. 45,10/öS 382,20) **DM 49,-**

COMMODORE PC 10/PC 20

D. A. Lien

BASIC-Programmierung PC 10/PC 20

1985, 488 Seiten

Ein amerikanisch-lockerer BASIC-Kurs von dem kalifornischen Professor Lien. Durch seine Systematik ideal als Kursunterlage für PC 10/20 und Kompatible. Mit Einführung in das PC-10-System und Tastendarstellung im Text.

Best.-Nr. PW 80366, ISBN 3-921803-66-7
(sFr. 54,30/öS 460,20) **DM 59,-**

Die angegebenen Preise sind Ladenpreise

Wilhelm Besenthal
Jens Muus

Alles über den C16

Die Autoren:
WILHELM BESENTHAL, 27 Jahre alt, programmierte schon 1980 in der Maschinensprache des 6502. Er arbeitete damals mit einem selbstgebauten Einplatinen-Computer. Er stieg 1981 auf einen in BASIC programmierbaren Computer der Marke OHIO SCIENTIFIC um. Es folgten ein C64 und ein C16.

JENS MUUS ist 29 Jahre alt. Seine Arbeit mit Computern begann 1980 mit einem programmierbaren Taschenrechner, dem HP 41 CV. Seit 1981 programmiert er in der Programmiersprache BASIC. Er arbeitete zunächst mit einem VC 20, stieg dann aber, nachdem der Commodore 64 erschien, auf diesen Computer um. Da beide Computer über ein sehr mageres BASIC verfügen, befaßte er sich sehr bald mit der Maschinensprache des 6502. Jens Muus erarbeitete sich umfassende Kenntnisse in der Interrupttechnik dieser Computer. Heute befindet sich neben dem C 16 auch ein ATARI 520 ST+ in seinem Besitz. Beide Autoren blicken auf jahrelange Erfahrung mit dem Programmieren in BASIC und Maschinensprache des 6502 zurück.

Der C 16/C 116 ist mit seinem günstigen Preis-Leistungs-Verhältnis eines der gefragtesten Einsteigermodelle. Gerade dem noch unerfahrenen Anwender bietet dieser Homecomputer gute Voraussetzungen für einen erfolgreichen Anfang: ein leistungsfähiges BASIC, hervorragende Grafikeigenschaften und eine sehr gute Tastatur.
Das Buch enthält, übersichtlich gegliedert, alle Informationen, die für die praktische Arbeit am Computer notwendig sind.

Ausgangspunkt ist ein kompletter BASIC-Kurs, der anhand vieler instruktiver Beispiele in die Arbeit mit der am weitesten verbreiteten Programmiersprache einführt. Daran schließt sich ein Kapitel über Aufbau und Funktion der Hardware an. Viele interessante Listings regen zum direkten Nachvollziehen am Computer an.

Aus dem Inhalt:
- Bedienung des C 16/C 116
- Programmierung der Funktionstasten
- Grafik-Grundlagen mit Beispielen
- Das Arbeiten mit Datasette und Diskettenlaufwerk
- Fehlersuche
- Maschinensprachemonitor
- Register der BASIC-Befehle
- Strukturiertes Programmieren

Hardwareanforderung:
Commodore 16 oder Commodore 116, Datasette oder Diskettenlaufwerk.

ISB N 3-89090-385-1

DM 39,-
sFr. 35,90
öS 304,20