Markt&Technik

Das C64 Profihandbuch

Alle zur fortschrittlichen Programmierung notwendigen Informationen anwendergerecht dargestellt

Hans Lorenz Schneider
Werner Eberl

Tabellen und Diagramme
Allgemeine Programmroutinen
Listings mit Prüfsummen
Viele Utilities

Das C 64-Profihandbuch

Hans Lorenz Schneider
Werner Eberl

Das C 64-Profihandbuch

Alle zur fortschrittlichen Programmierung notwendigen Informationen anwendergerecht dargestellt

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Schneider, Hans Lorenz:
[Das C-vierundsechzig-Profihandbuch]
Das C64-Profihandbuch : e. Nachschlagewerk / Hans-Lorenz Schneider ; Werner Eberl. —
Haar bei München : Markt-und-Technik-Verlag, 1985.
ISBN 3-89090-110-7
NE: Eberl, Werner:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86 85

ISBN 3-89090-110-7

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

Vorwort

In diesem Buch haben wir für Sie wichtige und nützliche Informationen über den Commodore 64 zusammengetragen. Der Begriff 'Profi' ist einerseits in seinem ursprünglichen Sinn zu verstehen (professionelle Anwendungen), es ist aber auch an die Leser gerichtet, die fast alle allgemeinen Fähigkeiten an ihrem Commodore 64 ausgereizt haben. Einiges aus dem Inhalt wird sicherlich bekannt sein, aber unsere Absicht war die Zusammenstellung eines Nachschlagewerkes. Das Buch soll eine umfassende Information über Ihren Rechner auf höherer Ebene darstellen.

Das umfangreiche Stichwortverzeichnis soll Ihnen die Handhabung des vorliegenden Buches als Nachschlagewerk erleichtern, obwohl die Themata nicht lexikographisch behandelt, sondern in logische Gruppen untergliedert sind.

Abschließend sei noch zu erwähnen, daß die Programme mittels des - aus der Zeitschrift 64er bekannten - Checksummers ausgedruckt wurden, um Ihnen die Arbeit zu erleichtern. Eine Beschreibung und das Listing des Checksummers finden Sie im Anhang.

München, im Februar 1985

Hans Lorenz Schneider

Werner Eberl

Einleitung

Obwohl das Buch ein Nachschlagewerk sein soll, haben wir es in fünf große Kapitel untergliedert. Das Nachschlagen wird durch das umfangreiche Stichwortverzeichnis am Ende dieses Buches erleichtert.

Das erste Kapitel beschäftigt sich mit allgemeinen Algorithmen, die nicht nur speziell für den 64er angewendet werden können, sondern auch auf andere Rechner übertragbar sind. Die dargestellten Programmstücke sind in diesem Kapitel nicht alleine lauffähig, da es sich nur um die Realisierung einzelner Probleme handelt, die in andere Programme einzubetten sind.

Im zweiten Kapitel sind einige Utilities (getrennt nach Basic- und Maschinenprogrammen) ausgeführt. Da sich Basic-Erweiterungen im Commodore 64 ohne Hardwareänderungen sehr leicht durchführen lassen, wurden die im zweiten Teil vorgestellten Maschinenprogramme auch als Basic-Erweiterung ausgebaut. Besonders nützlich für den Profi dürften die erweiterten PEEK- und POKE-Funktionen sein.

Das dritte Kapitel widmet sich den sechs Verbindungen des Rechners nach außen hin, und im vierten Kapitel sind einige nützliche Tips und Tricks aufgezählt, die von einzelnen PEEK's und POKE's über SYS-Aufrufe bis hin zu kurzen Programmen reichen.

Das letzte Kapitel stellt die Internas des Commodore 64 in tabellarischer Form sowie einigen Abbildungen zusammen. Besonders wird hier auf den Speicher und die Registerverteilung (VIC, SID, CIA) eingegangen, aber auch die im ROM implementierten Routinen werden angesprochen. Neben einigen anderen Übersichten befindet sich hier auch eine ausführliche Fehlerbeschreibung, sowohl der Fehler des Rechners als auch der Floppy.

Neben dem Literatur- und Stichwortverzeichnis finden Sie im Anhang zwei Übersichten zur Übernahme der vorgestellten Basic-Erweiterungen in Ihren Rechner (Hex-Listing, Basic-Loader).

Das Commodore 64 Profi-Handbuch

Inhaltsverzeichnis

1

Allgemeine Routinen

1. Allgemeine Routinen - nicht nur für den Commodore 64

Da das Handbuch ein Nachschlagewerk für den Profi sein soll, haben wir im ersten Kapitel einige Programmsequenzen zusammengestellt, die nicht 64er-spezifisch sind. Es sind Programmstücke, die man im täglichen 'Programmiererleben' immer wieder braucht. Sicherlich werden Sie diesem Kapitel viele nützliche Tips entnehmen können.

1.1 Variablendefinitionen

Als erstes wollen wir uns mit einem Thema beschäftigen, dem - trotz seiner Wichtigkeit - recht wenig Beachtung geschenkt wird, vielleicht, weil eine explizite Definition von Variablen in Interpreter-Basic nicht notwendig ist. Die Variablen werden hinter dem Programmteil im Anwenderbereich des Speichers gehalten. Eine Variable erhält ihren Speicherplatz bei ihrem ersten Aufruf vom Interpreter zugewiesen. Da der Interpreter bei Verwendung einer Variablen alle Vorhandenen absucht, um den Wert zu ermitteln, werden offensichtlich Variablen, die sehr früh im Programm auftauchen, auch schneller gefunden. Daher sollte man häufiger auftauchende Variablen am Programmanfang mit einer 'leeren Zuweisung' versehen (A%=0; A=0 oder A$=""), um die Rechenzeit zu verkürzen, zumal, wenn die Variable am Programmende in einer Schleife steht, die sehr oft durchlaufen wird.

Auch wenn BASIC nur zweibuchstabige Variablennamen zuläßt, sollte man die Bezeichnung möglichst mnemotechnisch wählen (DA$-Datum/Date; NA$-Name; TE$-Telefon). Einige Bezeichnungen haben sich auch eingebürgert : I, J, K, L, M, N als Laufvariablen ; H1, H2, ..., HZ als Hilfsvariablen, die nur temporär benutzt werden. Man sollte auch darauf achten, daß Variablen , die in mehreren Programmen vorkommen, auch immer den gleichen Namen haben, was die Testphase und die Wartung der Programme sehr vereinfacht, auch wenn es der Rechner nicht zwingend vorschreibt. Die Lesbarkeit der Programmlistings wird dadurch sehr erhöht.

Die einzigen explizit zu definierenden 'Variablen' sind die Funktionsdefinitionen, und die Array's (Felder, Matrizen, Vektoren), die mehr als 11 Einträge haben. Jedoch sollte man Felder schon wegen der Dokumentation dimensio-

nieren, auch wenn es aus Speicherplatzgründen nicht notwendig ist (vorbesetzt ist die Feldlänge bei Commodore mit 11 Einträgen, aber DIM A(5) spart Speicherplatz, weil Felder aufgrund der Anzahl ihrer Elemente den Speicherplatz zugewiesen bekommen). Wenn ausreichend Platz vorhanden ist, sollte man die Felder - von denen die Größe vorher nicht genau bekannt ist, entsprechend größer wählen.

Im Folgenden soll noch ein anderer 'Typ' von Variablen beschrieben werden, nämlich Variablen, die andere Variablen beschreiben.

Um die Erstellung individueller Software weiter zu erleichtern und um damit auch die enormen Personalkosten in Grenzen zu halten, geht man dazu über, Individual-Software auf Standardbasis zu erstellen. Dies bedeutet nichts anderes, als daß die Anwender-Software ihrerseits in eine 'Hardware' und 'Software' geteilt wird. Mit 'Hardware' ist also hier der Teil der Anwender-Software gemeint, der weder zum individuellen Teil noch zum Betriebssystem gehört. Den höchsten Standard bilden hier zur Zeit die hochintegrierten Pakete, die es allerdings erst für größere Personal-Computer gibt.

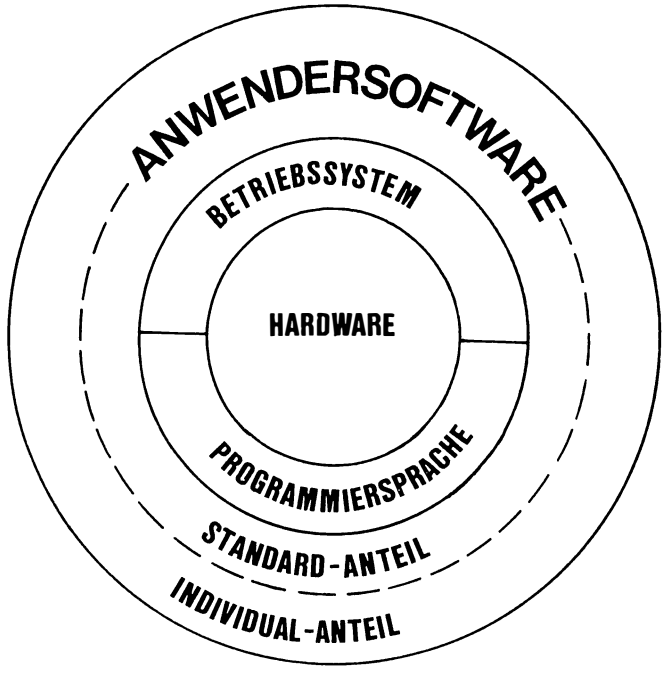


Bild 1.1-1: Schematischer Aufbau des Zusammenhanges zwischen Hardware und Software

Programmgeneratoren, die vor allem in der Groß-EDV eingesetzt werden, bieten hier die Möglichkeit, durch Eingabe von speziellen Programmkenndaten und weiteren Beschreibungen fertige Software zu erstellen. Eine andere Möglichkeit ist das Programmieren mit zwei logischen Variablenebenen.

Als Beispiel für das Programmieren mit zwei logischen Variablenebenen soll hier das Bearbeiten einer Datendatei in Auszügen dargestellt werden. Die Bearbeitung einer Datei wurde als Beispiel gewählt, weil hier ein großer Einsparungseffekt erzielt werden kann. Zur besseren Übersichtlichkeit wird ein dem Commodore-Basic ähnliches Pseudo-Basic verwendet.

Was bedeuten nun diese beiden logischen Variablenebenen? Den Daten, die durch das Programm verarbeitet werden sollen, fügen wir noch Beschreibungen dieser Daten hinzu, die ebenso durch Variablen dargestellt werden. Da Rechner im allgemeinen nur zwischen den Variablentypen INTEGER, FLOATING POINT und STRING unterscheiden, ist eine physikalische Unterscheidung der beiden Datentypen nicht mögich. Daher wollen wir sie logische Variablenebenen nennen.

Wenden wir uns gleich dem Beispiel einer 'Dateiverwaltung' für eine Schallplattensammlung zu. In herkömmlicher Programmierung könnte diese Erfassung lauten:

Beispiel (1):

```
10 INPUT"Titel       ";T$    (TI$ reserviert für Zeit)
20 INPUT"Interpret   ";IN$
30 INPUT"Ersch.Jahr  ";EJ$
40 INPUT"Fundort     ";FU$
50 GOSUB 'SPEICHERN'
```

Es werden also die vier Variablen T$,IN$,EJ$ und FU$ eingelesen. Dieses Programmstück ist starr auf eine bestimmte Dateiverwaltung eingerichtet und zu nichts anderem zu gebrauchen.

Schreiben wir nun das Programmstück einmal anders unter Zuhilfenahme der Variablen

```
AW      - Anzahl der Werte    (=4)
NA$(4) - Namen der Eingaben
IH$(4) - Inhalt der Eingaben
```

Beispiel (2):

```
10 DATA4,Titel,Interpret,Ersch.Jahr,Fundort
20 READ AW
30 FOR I=1 TO AW:READ NA$(I):NEXT

100 FORI=1 TO AW
110       PRINT NA$;:INPUT IH$(I)
120 NEXT
130 GOSUB 'SPEICHERN'
```

Für den Anwender sind beide Programme nicht voneinander zu unterscheiden. Der zeitliche Aufwand (auch für die Programmierung) ist in etwa gleich.

Nehmen wir nun einen anderen Anwendungsfall: Erfassen von Büchern einer Heimbibliothek mit dem herkömmlichen Programm:

Beispiel (3):

```
10 INPUT"Titel              ";T$
20 INPUT"Autor              ";AU$
30 INPUT"Verlag             ";VE$
40 INPUT"Rubrik             ";RU$
50 INPUT"Fundort            ";FU$
60 INPUT"ausgeliehen an:    ";AA$
```

oder vielleicht eine Lagerverwaltung

Beispiel (4):

```
10 INPUT"Artikelnr          ";AR$
20 INPUT"Bezeichnung        ";BE$
30 INPUT"Bestand            ";BS$
40 INPUT"EK-Preis           ";EK$
50 INPUT"VK-Preis           ";VK$
```

Für Beispiel (3) und (4) muß man wieder ein neues Programm schreiben, wenn man das Programm aus Beispiel (1) bereits hat. Wenn jedoch das gewünschte in Form des Programms in Beispiel (2) geschrieben wurde, genügt es, die Zeile 10 jeweils zu ändern auf:

```
10 DATA 6,Titel,Autor,Verlag,Rubrik,Fundort,ausgeliehen an
        (für Beispiel 3)
```

oder

```
10 DATA 5,Artikelnr,Bezeichnung,Bestand,EK-Preis,VK-Preis
        (für Beispiel 4)
```

Wie man schon an diesen kleinen Beispielen sieht, ergeben sich hier einige Zeitvorteile bei der Erstellung von Programmen mit gleicher Struktur, aber verschiedenem Verwendungszweck. Stellvertretend für die Ebene der zu verarbeitenden Daten steht dabei in Beispiel (2) das Variablenfeld IH$(AW) und für die Ebene der programmbeschreibenden Daten die Variable AW und das Feld NA$(AW).

Noch größer wird der Einsparungseffekt beim Programmieren des Änderungsdienstes.

Für die Daten aus (1) könnte das so aussehen:

Beispiel (5):

```
200 GOSUB'EINLESEN'
210 GOSUB'ANZEIGEN DER DATEN'
220 PRINT"1 - Titel
230 PRINT"2 - Interpret
240 PRINT"3 - Ersch.Jahr
250 PRINT"4 - Fundort
260 GET A$:IFA$=""THEN260
270 A = VAL(A$):IF A LT 1 OR A GT 4 THEN260
                        (LT für kleiner; GT für größer)
280 PRINT"Neuer Wert"
300 ONAGOTO310,320,330,340
310 INPUT T$:GOTO 350
320 INPUT IN$:GOTO 350
330 INPUT EJ$:GOTO 350
340 INPUT FU$:GOTO 350
350 GOSUB 'SPEICHERN'
```

wobei die Unterroutine zum Anzeigen der Daten so aussehen könnte:

```
10000 PRINT"Titel       :";T$
10010 PRINT"Interpret   :";IN$
10020 PRINT"Ersch.Jahr  :";EJ$
10030 PRINT"Fundort     :";FU$
```

Analog zu (2) könnte das Programm lauten:

Beispiel (6):

```
200 GOSUB 'EINLESEN'
210 GOSUB 'ANZEIGEN DER DATEN'
220 FOR I=1 TO AW
230       PRINT I;"-";NA$(1)
240 NEXT
250 GETA$ : IFA$ = "" THEN250
260 A = VAL(A$) : IF A LT 1 OR A GT AW THEN250
270 INPUT "Neuer Wert";IH$(A)
280 GOSUB 'SPEICHERN'
```

und die Routine zum Anzeigen:

```
10000 FORI = ITOAW:PRINT NA$,IH$:NEXT
```

Wollte man für die in (3) und (4) erfaßten Daten einen Änderungsdienst programmieren, so müßte man das Programm in (5) fast ganz neu schreiben, beim letzten Beispiel genügt jedoch auch wieder das Ändern der Zeile 10.

Neben den Einsparungen beim Programmieren sieht man jetzt schon, daß beim Arbeiten mit programmbeschreibenden Variablen noch etwas sehr Wichtiges eingespart wird: der kostspielige und rare Speicherplatz (Die Begriffe wie Titel, Interpret etc. benötigen nur einmal Speicherplatz). Die Programmlaufzeiten erhöhen sich jedoch nur unwesentlich (Einlesen der Daten AW und NA$(AW), Verwalten von Feldern).

Wie dem aufmerksamen Leser sicher nicht entgangen ist, werden als Variablen in den Beispielen nur Strings verarbeitet. Dies ist bei der Erstellung von kommerzieller Software eine allgemein übliche Praxis, da auf diese Art und Weise bei der Dateneingabe über eine 'Input-Routine' bereits eine Prüfung der Daten auf Plausibilität erfolgen kann (z.B. Länge der Eingabe, korrektes Datum etc.). Außerdem ist bei eindimensionalen Feldern nicht möglich, Feldelemente verschiedenen Datentypen zuzurechnen. Wenn es erforderlich sein sollte, als String eingegebene Daten zum Rechnen zu benutzen, so ermöglicht dies die Funktion VAL:

```
X = VAL(IH$(I))
```

Beim 'Rechnen mit Strings' ergeben sich zwar mitunter lange Programmzeilen, die bei näherem Hinsehen jedoch gar nicht so kompliziert sind.

Beispiel (7):

```
NA$(4) = "Bestand       "      IH$(4)="100    "
NA$(5) = "EK-Wert       "      IH$(5)="  2,50"
NA$(9) = "Lagerwert     "      IH$(9)=" ?     "
```

IH$(9) 'errechnet' sich auf folgende Weise

```
IH$(9) = STR$( VAL(IH$(4)) * VAL(IH$(5)) )
```

Obwohl man ein so allgemeines Programm lediglich mit Daten vom Typ STRING aufbauen kann, ist es für die Prüfung von Plausibilitäten jedoch unerläßlich zu wissen, ob die einzugebenden Daten Zahlen sein dürfen oder nicht. Wir ergänzen daher die Zeile 10 (für 4) durch die Zeilen

```
11 DATA A,A,I,F,F
12 DATA 6,40,4,5.2,5.2
```

und benutzen die neuen Variablen

```
AR$(AW) - Art der Daten
          A - Alphanumerische Zeichen (Buchstaben)
          I - Integer (Ganze Zahlen)
          F - Floating Point (Dezimalzahlen)

L(AW)   - Länge der Daten
          für A - Länge der Zeichenreihe
          für I - Anzahl der Stellen
          für F - Anzahl der Vor- und
                  Nachkommastellen
```

Eingelesen werden die Daten mit

```
40 FOR I = 1 TO AW : READ AR$(I):NEXT
50 FOR I = 1 TO AW : READ L(I):NEXT
```

oder kürzer:

```
FOR I = 1 TO AW
   READ NA$(I)
   READ AR$(I)
   READ L(I)
NEXT
```

wenn man die DATA-Anweisungen in den Zeilen 10 bis 12 entsprechend ändert. Die mit diesen Variablen genauer spezifizierten Daten in Zeile 10 können nun auch Plausibilitätsprüfungen auf ihre Eigenschaften unterzogen werden, ohne daß dazu das jeweilige Programm geändert werden muß. Z.B. kann die Länge der Artikelnummer geprüft werden mit der Zeile:

```
IF AR$(1)="A" THEN IF LEN(IH$(1)) GT L(I) GOTO'FEHLER'
```

oder die Höhe des Bestands auf vier Stellen mit:

```
IF AR$(3)="I" THEN IF ABS(VAL(IH$(3))) GT 10**L(I)
        GOTO'FEHLER'
```

Um ein Programm noch weiter zu abstrahieren, kann man statt der DATA-Zeilen noch sequentielle Dateien verwenden und dem Programm ein Menü der zu bearbeitenden Dateien voranstellen, wodurch eine sequentielle Datei gewählt wird, aus der die programmbeschreibenden Daten (in den Beispielen AW, NA$(AW), L(AW), AR$(AW) eingelesen werden.

1.2 Menütechniken

Wenn man sich einige kleine Unterprogramme, Funktionen oder Hilfsprogramme geschrieben hat, so ist es eine ziemlich aufwendige Sache, wenn man dauernd neue Programme nachladen muß. Man kann diese Programme aber auch als Teil eines Ganzen ansehen, d.h. alles in ein Programm laden und über ein Menü - wie in der kommerziellen Datenverarbeitung - aufrufen.

Gegeben seien neun Unterprogramme oder Funktionen (alle unabhängig voneinander), die jeweils in den Zeilen 1000-1999, 2000-2999,..., 9000-9999 stehen. Die Programmteile enden alle mit der Anweisung 'GOTO 100'.

Folgendes Programmstück liefert ein einfaches Menü:

```
100 CLR      rem löschen aller Variablen
110 PRINT "(clr screen/crsr home/5 x crsr down)"
120 PRINT "    1 - (Programmname 1)
130 PRINT "    2 - (Programmname 2)
140 PRINT "    3 - (Programmname 3)
150 PRINT "    4 - (Programmname 4)
160 PRINT "    5 - (Programmname 5)
170 PRINT "    6 - (Programmname 6)
180 PRINT "    7 - (Programmname 7)
190 PRINT "    8 - (Programmname 8)
200 PRINT "    9 - (Programmname 9)
210 PRINT : PRINT : PRINT
220 PRINT "       Bitte wählen Sie:";
230 GETA$ : IF A$ = "" THEN 230
240 A=VAL(A$) : IF A LT 1 OR A GT 9 THEN 230
250 PRINT A$
260 ON A GOTO 1000,2000,3000,...,9000
```

Die Zeilen 110 bis 220 bringen offensichtlich das Menü auf den Bildschirm. Zu beachten ist hier das Semikolon in Zeile 220, um eine saubere Ausgabe zu gewährleisten.

Zeile 230 ist eine Warteschleife für die Eingabe, in Zeile 240 wird die Plausibilität der Eingabe geprüft. Es werden nur die Zeilen 1 bis 9 zugelassen, der Rest der Tastatur ist 'blockiert'. Zeile 260 beinhaltet den normalen Programmverteiler.

Es ist natürlich auch möglich, den Menüpunkten Buchstaben voranzustellen (E-erfassen, A-ändern, L-löschen, S- sortieren). In jedem Fall empfiehlt sich jedoch die Verwendung nur eines Zeichens um die Eingabe durch die RETURN-Taste zu sparen oder das Menüprogramm nicht unnötig zu komplizieren. Anwender, die sich durch mehrere Menü-Stufen 'durchquählen' müssen, wissen die Einsparung des RETURN's zu schätzen.

Werden Buchstaben bei der Menüanzeige verwendet, so sind die Zeilen 240 bis 260 zu ersetzen durch:

```
240 IF A$ = "A" THEN 1000       : REM ändern
250 IF A$ = "E" THEN 2000       : REM erfassen
    .              .                .
    .              .                .
    .              .                .
299 GOTO 230
```

Ein Rücksprung nach Zeile 100 ist zum einen nicht erforderlich, und zum anderen würde der Bildschirm nach einer Fehleingabe flackern. Die Ausgabe einer Fehlermeldung ist auch nicht erforderlich, da es sich meist nur um einen Tippfehler handelt, kann jedoch in Zeile 240 bzw. 299 eingefügt werden:

```
299 PRINT"Falsche Eingabe, bitte wiederholen":
        FOR I=1 TO 1000 : NEXT
```

Die FOR...NEXT-Schleife veranlaßt den Rechner zu warten, da er zur Bearbeitung der Schleife Zeit benötigt. Durch Ersetzen des Wertes '1000' kann jede beliebige Zeit eingestellt werden. Durch die Anweisung PRINT"(Cursor oben) (Anzahl der Leerzeichen in der Fehlermeldung)", kann die Fehlermeldung anschließend gelöscht werden.

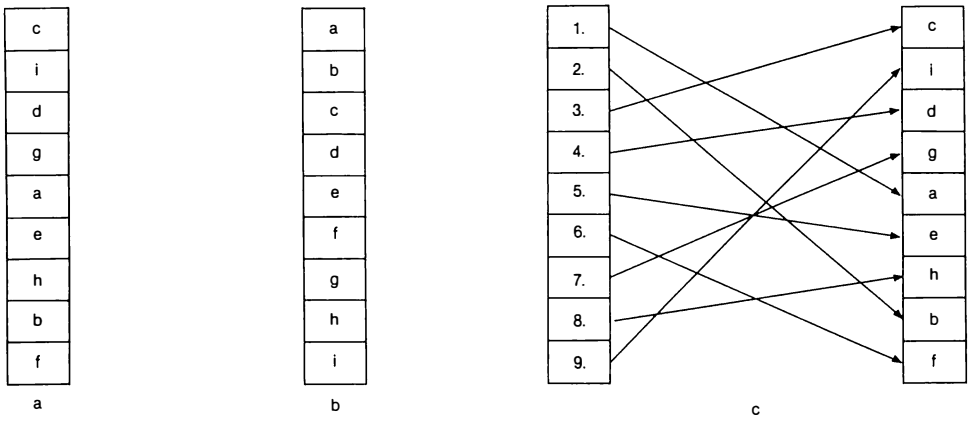
Die Verwendung eines GOSUB im ersten Beispiel Zeile 260 (ON A GOSUB ...) ist nicht sinnvoll, da hier nur der Stack belastet wird. Die Programmteile dürfen bei Verwendung von GOSUB nicht mit 'GOTO 100' beendet werden, sondern mit 'RETURN', da sonst der Stack irgendwann überläuft.

Bei mehrstufigen Menüs innerhalb eines Programmes sollten die Untermenüs in den Zeilen 1000,2000,... beginnen. Ein Zusammenfassen aller Menüs ist aus Dokumentationsgründen nicht anzuraten. Bei mehrstufigen Menüs sollte auf die Einheitlichkeit aller Menüs geachtet werden, um Bedienungsfehler zu vermeiden.

Abschließend in diesem Kapitel noch ein Wort zu den angewählten Programmen. Nicht immer ist es möglich, Arrays im Unterprogramm zu löschen. Deshalb sollte bei mehrfacher Verwendung von Programmteilen durch Menüaufruf die Dimensionierung der Arrays vor dem Hauptmenü erfolgen. Ebenso die Definition von Funktionen. Werden im Vorspann des Hauptmenüs Variablen von globaler Bedeutung vorbesetzt, so ist das CLR in Zeile 100 nicht zu verwenden. Achten sie in diesem Fall auf die Kompatibilität der Daten. Sicherheitshalber sind entsprechende Variablen immer neu in einem Programmteil vorzubesetzen. Achten sie auch darauf, daß geöffnete Dateien vor dem Sprung auf das Hauptmenü geschlossen werden.

1.3 Sortierverfahren

Neben dem Erfassen, Ändern, Löschen, Listen und selektiven Ausgeben der Daten bildet das Sortieren besonders im kommerziellen Bereich aber auch bei privaten Programmen ein sehr wichtiges Auswertungsverfahren (z.B. Hitlisten im Verkauf, Ladenhüterlisten in der Lagerverwaltung, alphanumerisch sortierte Kundenlisten, Kunden nach Umsätzen sortiert oder Schallplatten nach Titel und Interpret sortiert). Sortieren kann zum einen bedeuten, daß Zahlen oder Datensätze, die ein bestimmtes Sortierkriterium enthalten, alphanumerisch geordnet werden, es kann aber auch heißen, daß die ursprüngliche Ordnung der Daten nicht verändert wird und die alphanumerisch sortierte Reihenfolge über Zeiger durchgeführt wird.



Sortieren von Daten

a) unsortiert

b) Sortieren im Ursprung

c) Sortieren mit Hilfsbefehl

Bild 1.3-1: Prinzipielle Sortiermöglichkeiten

In diesem Kaptitel soll nur kurz auf die verschiedenen Sortierverfahren und ihre Vor- und Nachteile eingegangen werden. Eine umfangreiche Diskussion aller Sortierverfahren ist im Rahmen dieses Buches leider nicht möglich.

1.3.1 Maxsort / Minsort

Maxsort und Minsort unterscheiden sich im Algorithmus nicht, der Unterschied liegt im Endergebnis: Maxsort sortiert absteigend; Minsort sortiert aufsteigend. Maxsort und Minsort sind die einfachsten Sortierverfahren. Mit ihnen werden unter Zuhilfenahme einer Hilfsvariablen die Datensätze so umgruppiert, daß das jeweils größte (kleinste) Element herausgesucht wird und mit dem ersten noch nicht sortierten Element über die Hilfsvariable vertauscht wird. Nachfolgend ein Programmbeispiel für Minsort:

```
100 FOR I = 1 TO N-1
110     M = 0 : AM = 1 E 38
120     FOR J = I TO N
130        IF A(J) LT AM THEN AM = A(J) : M = J
140     NEXT J
150     AM = A(I) : A(I) = A(M) : A(M) = AM
160 NEXT I
```

Zum Ablauf: Zeile 100 läßt das Sortierverfahren für alle N Elemente durchlaufen. In Zeile 110 werden die Variablen M (Fundstelle des minimalen Elementes) und AM (aktuelles minimales Element) vorbesetzt. Die Daten selbst stehen in dem Vektor A(I). Dabei ist zu berücksichtigen, daß AM größer gleich dem minimalen Element im zu sortierenden Vektor sein muß, was durch Vorbesetzen mit der größtmöglichen Zahl (oder bei Strings mit dem größtmöglichen Zeichen = CHR$(255)) geschieht.

In Zeile 120 beginnt eine Schleife, die in dem noch nicht sortierten Teil des Vektors das minimale Element sucht. Falls dieses gefunden wird (Zeile 130), wird der Zeiger, wo sich dieses Element befindet (M), mit der aktuellen Stelle (J) vorbesetzt und AM mit dem gefundenen Wert besetzt. Am Ende dieser Schleife (Zeile 140) steht in AM das minimale Element und in M,wo dieses Element zu finden ist. In Zeile 150 wird dieses minimale Element und das erste unsortierte Element im Datensatz-Vektor ausgetauscht. Wird

Zeile 160 erreicht, so sind N-1 Sortiervorgänge abgelaufen, das letzte Element (A(N)) braucht offensichtlich nicht mehr sortiert zu werden.

Betrachtet man sich die Anzahl der Zugriffe bei Minsort genauer, so erhält man die Gauss'sche Formel n*(n-1)/2, was ungefähr n**2/2 ('**' als Potenzierungsoperator 'hoch') entspricht. Bei einem n von etwa 5000 ergibt sich ein n**2/2 von 12.500.000 . Alle guten Sortierverfahren haben eine Zugriffshäufigkeit von n*log n, womit ein gutes Verfahren im Vergleich nur 60.000 Zugriffe benötigen würde, und damit um den Faktor 400 schneller fertig wäre als Minsort. Dies trifft zwar nicht ganz zu, da in der Regel bei Sortierverfahren noch ein allgemeiner Aufwand auftritt, der zusätzlich zum variablen Wert noch einen konstanten Faktor bringt. Dieser allgemeine Aufwand ist bei guten Verfahren wesentlich größer als bei Minsort. Nach Knuth (Literaturverzeichnis #1#) gelten etwa folgende Formeln für die Anwendung bei Sortierverfahren:

Primitives Verfahren:	2	* n ** 2	+ 9 * n
Heap sort:	23	* ln (n)	+ 0,2 * n
Quicksort:	12 n	* ln (n)	+ 2 * n

Aufgrund dieser Formel ist der Faktor zwischen Maxsort und Quicksort immerhin noch 100.

Für das Verhalten von Sortierverfahren ist weiterhin noch der Zustand der zu sortierenden Daten abhängig. Je nachdem, ob die Daten sortiert, antisortiert oder gut gemischt sind, schwankt der Zeitaufwand für den Sortiervorgang. Bei Quicksort (siehe unten) ist dieser Faktor im ungünstigsten Fall proportional n**2 . Allerdings sind diese Fälle nicht sehr häufig. Bei zeitkritischen Sortiervorgängen verwendet man besser ein schnelleres Verfahren, wie zum Beispiel HEAP-SORT.

1.3.2 Heap-Sort

Heap-Sort ist ein Sortierverfahren, das auf Binär-Bäumen basiert. Auf Binärbäume kann an dieser Stelle aus Platzgründen nicht gesondert eingegangen werden; es sei auf die entsprechende Literatur, z.B. Basic ohne Probleme Band 3, verwiesen. Heap-Sort ist ein sehr schnelles Verfahren und

läßt den Sortieraufwand auch bei ungünstiger Anordnung der Daten nicht allzu sehr anwachsen. Bei Heap-Sort geht ein Vorsortieren des Vektors nicht in die Bearbeitungsdauer mit ein. Der Aufwand bei Heap-Sort wird im wesentlichen durch den Faktor n*log(n+1) bestimmt. Bei großen Datenmengen ist Heap-Sort besonders günstig, weil die Daten in ihrem eigenen Feld sortiert werden und kein zusätzlicher Speicherplatz benötigt wird. Heap-Sort besteht im Prinzip daraus, daß aufgrund der Daten ein Binärbaum erzeugt wird, der dann nach einem vorgegebenen Algorithmus ausgedruckt wird. Erhält man den Binärbaum auch nach dem Sortieren, so läßt sich durch geeignete Verfahren des Einfügens und Löschens immer wieder auf die sortierten Daten zurückgreifen, ohne jeweils den Baum neu anlegen zu müssen. Der Algorithmus stammt aus dem Jahre 1964 und wurde von den Amerikanern Floyd und Williams entwickelt.

1.3.3 Quicksort

Quicksort ist ein sehr raffiniertes Verfahren und trägt seinen Namen zurecht. Eine genaue Beschreibung des Verfahrens befindet sich in #2#.

Quicksort ist ein rekursives Verfahren; rekursiv heißt, daß das Programm bzw. die Prozedur sich selbst aufruft. Dies ist in Basic im Sprachumfang nicht vorgesehen, und wir müssen die Rekursion selbst herbeiführen. Bevor wir jedoch zu einer möglichen Implementierungs-Version auf BASIC kommen, wollen wir uns die prinzipielle Verfahrensweise von Quicksort einmal allgemein verdeutlichen. Gegeben sei ein Vektor A() mit n Elementen, die es zu sortieren gilt. Man wähle sich nun ein beliebiges Element aus A() als Vergleichskriterium. Bei hinreichend geschickter Auswahl des Elementes aus A() kann man bei einigermaßen vorsortierten Daten die Sortierzeit sehr stark verkürzen. Ein günstiges Vergleichselement des Vektors ist z.B. bei einem vorsortierten Vektor in der Mitte zu finden. Die Elemente des Vektors A() werden nun so geordnet, daß links von unserem Vergleichselement AV alle Elemente stehen, die kleiner sind als AV und rechts alle Elemente, die größer sind. Nachdem diese Ordnungsbeziehung erreicht wurde, kann man auf die zwei Teilvektoren (1 bis AV ; AV+1 bis N) ein beliebiges Sortierverfahren anwenden: zum Beispiel Quicksort (daher der rekursive Unterprogrammaufruf).

Wie erreicht man nun diese Ordnung? Man prüft vom linken

Rand des Vektors ausgehend (mit einer Laufvariablen I) alle Elemente ab, ob diese kleiner sind als das Vergleichselement AV. Dies geschieht solange, bis ein Element auftritt, das größer ist als AV. Dann geht man in gleicher Weise vom rechten Rand des Vektors (mit einer Laufvariablen J) aus und prüft, ob alle Elemente größer sind als das Vergleichselement AV. Dies wiederum auch so lange, bis ein Element auftritt, das kleiner ist als AV.

An dieser Stelle zeigt nun der Laufanzeiger I auf ein Element das größer ist als AV, aber auf der linken Seite des Vektors steht, und der Laufanzeiger J zeigt auf ein Element, das kleiner ist als AV, aber im rechten Teil des Vektors steht. Die von uns gewünschte Ordnung ist durch einfaches Austauschen dieser beiden Elemente wieder hergestellt, und man läßt zunächst I und dann J weiter auf die Mitte des Vektors zulaufen, bis sie sich treffen (J=I+1; I=J-1). An dieser Stelle ist dann die gewünschte Ordnung erreicht, und auf die Teilvektoren 1,...,I und J,...,N kann wieder ein beliebiges Sortierverfahren angewendet werden. An dieser Stelle ist bereits ersichtlich, daß der Sortieraufwand mit der Anzahl der Vertauschungen wächst. Hat man nun einen einigermaßen vorsortierten Vektor (z.B. bei der Kundendatenerfassung werden neue Kunden im Anschluss an die bereits sortierten erfaßt, so wird man geschickterweise vor einem erneuten Sortiervorgang ein Element, das kurz hinter der Mitte des Vektors liegt, oder das Element INT(N/2) wählen, womit die Vertauschungsvorgänge auf ein Minimum reduziert werden).

Auch in BASIC ist das einmalige Herstellen einer Ordnungsbeziehung, wie sie oben erwähnt wurde, keine Schwierigkeit. Probleme treten dann auf, wenn auf die beiden Teilvektoren jeweils wieder dasselbe Sortierverfahren angewendet werden soll, weil BASIC keine rekursiven Prozeduren und keine Parameterübergabe an Unterprogramme kennt. Unseren rekursiven Programmaufruf wollen wir an das von höheren Programmiersprachen verwendete Verfahren anlehnen.

Dazu verwenden wir einen sogenannten Stack (Kellerspeicher), der nach der Last In - First Out-Regel (LIFO / was zuletzt gespeichert wurde, wird als erstes wieder verwendet) arbeitet. Als Parameter müssen jeweils der linke und rechte Rand des zu sortierenden Teilvektors übergeben werden. Wir definieren uns also zwei Felder L(1000) und R(1000), wo jeweils die Werte für den linken und den rechten Rand des Teilvektors abgelegt werden. Da bei jedem Aufruf von Quicksort je ein Element der beiden Vektoren L(1000) und R(1000) abgearbeitet wird, und nur zwei neue Elemente hinzukommen, ist somit eine Kellertiefe von 1000 Elementen auch für relativ große Datenmengen ausreichend.

Wie Erfahrungswerte zeigen, ist der allgemeine Aufwand von Quicksort zur Verwaltung des Kellers bei weniger als 10 bis 20 Elementen sehr groß, und wir wollen bei einem Teilvektor, der weniger als 15 Elemente enthält, das Sortierverfahren Minsort anwenden. Dies entlastet gleichzeitig auch unseren Kellerspeicher, da bei Minsort dem Kellerspeicher keine weiteren Elemente zugeführt werden.

Eine mögliche Basic-Version von Quicksort:

```
100 DIM L(1000)
110 DIM R(1000)
120 Z = 1
130 L(1) = 1 : R(1) = N
140 L = L(Z) : R = R(Z)
150 Z = Z - 1
160 IF R-L LT 15 THEN GOSUB 'MINSORT' : GOTO 270
170 AV = A(INT((L+R)/2))
180 I = L : J = R
190 FOR K=1TO N
200 IF A(I) LT AV THEN I = I+1 : GOTO 200
210 IF A(J) GT AV THEN J = J+1 : GOTO 210
220 IF I GT J THEN 250
230 H = A(I) : A(I) = A(J) : A(I) = H
240 IF I LE J THEN NEXT K
250 IF L LT J THEN Z = Z+1 : L(Z) = L : R(Z) = J
260 IF I LT R THEN Z = Z+1 : L(Z) = I : R(Z) = R
270 IF Z GT 0 GOTO 140
280 END
```

Beschreibung: In den Zeilen 100 und 110 werden die provisorischen Stacks für den Vektoranfang in das Vektorende gesetzt. In Zeile 120 wird der Stackzeiger auf das erste Element gesetzt und in Zeile 130 werden auch die entsprechenden Elemente der Matrix vorbesetzt.

In Zeile 140 beginnt der eigentliche Sortiervorgang (diese Zeile wird für jeden neuen Durchlauf angesprungen). Dann wird der Zeiger auf den Stack auf das vorhergehende Element - bereits für den nächsten Durchlauf zurückgesetzt und geprüft, ob das Verfahren Minsort nicht effizienter ist. In Zeile 170 wird das Vergleichskriterium besetzt und in Zeile 180 die aktuellen Grenzen festgelegt. In Zeile 190 beginnt eine prinzipielle Endlosschleife, die sicherstellt, daß auch der ganze Bereich geprüft wird. Dann erfolgt der

Vergleich elementweise (Zeile 200 linke Seite; Zeile 210 rechte Seite). Ein Durchlauf ist beendet, wenn sich die beiden Laufvariablen 'begegnen'.

Wurden die Vergleichsschleifen verlassen, ohne daß das Endekriterium erreicht ist, so muß eine Vertauschung durchgeführt werden (Zeile 230). In den beiden nächsten Zeilen werden die Daten der beiden Teilvektoren für den Stack aufbereitet, was dem rekursiven Aufruf entspricht und Zeile 260 prüft, ob noch etwas zu sortieren ist.

Da das Sortieren auf der Diskette wegen der Zugriffszeiten sehr zeitaufwendig ist (im Gegensatz zum Sortieren im Arbeitsspeicher), sollte man den Sortiervorgang nach Möglichkeit in den Arbeitsspeicher verlagern. Da in der Regel Datensätze nur nach einem (vielleicht auch mehreren, aber nie allen) Elementen sortiert werden müssen, genügt es, die zu sortierenden Datensatzteile in den Rechner einzulesen, und im Arbeitsspeicher zu sortieren. Dabei darf man jedoch nicht vergessen, die Information, die angibt, wo sich der Datensatz befindet, mit zu sortieren. Will man z.B. eine relative Datei nach Namen sortieren, so genügt es, die Namen in ein Feld A$() einzulesen, und - korrespondierend zu der Feldnummer in A$() - ein Feld A(), das die Nummer des Datensatzes enthält. Am Anfang wird A(I)=I sein, da die Daten sequentiell eingelesen werden sollten. Steht in der Sortierroutine eine Vertauschung von Feldelementen wie

```
H$=A$(I) : A$(I)=A$(J) : A$(J)=H$
```

so ist eine Zeile

```
H=A(I) : A(I)=A(J) : A(J)=H
```

zu ergänzen. Ein nach Namen sortierter Ausdruck aller Datensätze kann dann mit einem Schleifendurchlauf durch den Vektor der Datensatzzeiger erfolgen.

Sind die einzelnen Felder des Datensatzes mit Leerzeichen aufgefüllt, d.h. wird beim Einlesen statt 'Huber' 'Huber ' eingelesen - was durchaus sinnvoll in anderen Programmteilen sein kann - so sollte man zuerst die anhängenden Leerzeichen eliminieren, um keinen Speicherplatz zu verschenken. In jedem Fall ist bei großen Datenmengen eine Speicherplatzüberschlagsrechnung durchzuführen, womit der Speicherbedarf aller einzulesenden Werte

festgestellt werden soll. Stehen nur 35 Kilo-Byte Anwenderspeicher zur Verfügung, so ist es unmöglich, 1000 Einträge mit durchschnittlich 35 Zeichen Länge zu sortieren, da einerseits das Programm sich auch im Arbeitsspeicher befinden muß und andererseits die Zusatzinformation in A(N) - wo der Datensatz steht - auch Platz benötigt. Hier könnte ein Mischen und Sortieren helfen (siehe unten).

1.3.4 Andere Sortierverfahren

Eine weitere Möglichkeit des Sortierens ergibt sich durch die Verwendung von Suchbäumen. Die schon erwähnten Binär-Bäume sind als solche Suchbäume angelegt. Die Anwendungsmöglichkeiten für Suchbäume sind im Datenbankwesen und bei der Verwaltung von Variablenlisten in Compilern. Sie erlauben ein einfaches Einfügen, Suchen, Sortieren und Auslisten. Jedoch ein äußerst wichtiges Problem bei den Suchbäumen ist die mögliche Entartung. Unter Entartung eines Baumes versteht man, daß einzelne Zweige relativ lang zur durchschnittlichen Länge des gesamten Baumes sind. Ein guter Suchbaum wird durch eine solche Entartung erheblich verschlechtert, da sich die Zugriffszeiten um ein Mehrfaches im Durchschnitt erhöhen können.

Bei sogenannten AVL-Bäumen wird die Entartung durch spezielle Programmteile vermieden. Dafür haben AVL-Bäume einen erheblich größeren Rechenzeitaufwand beim Einfügen in den Baum. Die AVL-Bäume (nach Adelson, Velski, Landis) sind sogenannte balancierte Suchbäume. Wenn die Höhe eines Baumes als der längste Weg von der Wurzel zu einem Blatt gilt, so ist ein Suchbaum genau dann ein AVL-Baum, wenn sich die Höhe zweier beliebiger Knoten eines beliebigen Teilbaumes um maximal 1 unterscheidet (zur Vollständigkeit: ein nicht vorhandener Baum hat die Höhe null).

Ähnlichen Aufwand muß man oft für die sogenannten B-Bäume betreiben. Die B-Bäume sind Suchbäume, aber keine Binär-Bäume. Entwickelt wurden sie um 1971/1972 von R. Bayer. Die Grundidee der B-Bäume ist die mehrfache Bewertung von Knoten, und die Zulassung von mehr als zwei Söhnen.

Sinnvoll ist die Anwendung von B-Bäumen, wenn der Baum auf einer Platte abgelegt wird und sehr viele Daten zu verwalten sind. Jeder Zugriff auf einen Knoten würde dann einem Plattenzugriff entsprechen. Da bei schnellen Plattenlaufwerken und hohen Datenübertragungsraten zwischen dem Com-

puter und der Peripherie die Datenlesezeiten im Verhältnis zu den Datenzugriffszeiten günstiger sind, ist es rationeller, bei einem Plattenzugriff mehr als eine Bewertung in den Arbeitsspeicher des Rechners einzulesen. Bei Großanlagen liegt die Anzahl der Bewertungen in einem Knoten zwischen 100 und 1000.

Ein Entarten eines B-Baumes wäre in diesem Fall nicht so sehr von der Anzahl der Blätter und der Höhe des Baumes abhängig, sondern eher von der Anzahl der Bewertungen in einem Knoten, im schlechtesten Falle könnte sogar nur eine Bewertung in einem Knoten stehen, der bis zu 1000 Bewertungen fassen kann. Um dies zu verhindern, werden einige Tricks angewendet. Als erstes wird eine Konstante m vorgegeben, so daß die Anzahl der Werte in einem Knoten zwischen m und 2m betragen soll (Beispiel: m = 50; jeder Knoten enthält mindestens 50 und max. 100 Bewertungen). Außerdem soll der B-Baum (wie auch der AVL-Baum) immer vollständig ausgeglichen sein. Weil die Anzahl der Werte in einem Knoten variabel ist, wird die Anzahl im Knoten zusätzlich mitgespeichert, denn bei 1000 Bewertungen in einem Knoten und ebenso vielen Zeigern auf andere Knoten, ist der Speicherplatzbedarf für diese Information vernachlässigbar. Generell werden im ungünstigsten Fall bis zu 50 Prozent des Speicherplatzes verschenkt. Näheres über B-Bäume finden Sie in #3#.

Da im Prinzip jeder Algorithmus, der geordnete Daten liefert, als Sortierverfahren bezeichnet werden kann, ist eine Aufstellung aller Verfahren nicht möglich. Die wichtigsten Verfahren wurden bereits erklärt. Was von einem Sortierverfahren zu halten ist, das bis zur Fertigstellung einer Ordnung immer alle Daten von Anfang bis Ende paarweise vergleicht und durch evtl. paarweise Vertauschung eine Ordnung herstellt, mag sich der geneigte Leser selbst überlegen: bei n ** 2 Abfragen und 2 * n ** 2 Lesevorgängen im schlechtesten Fall; Zeit bei n=5000 ?)

1.4 Selektieren

Das selektive Heraussuchen von Datensätzen aus einer Datei, die einem bestimmten Selektionskriterium genügen, ist besonders im kommerziellen Bereich von großem Vorteil. Man denke nur an die Textverarbeitung (Mailing), wenn es darum geht, Werbebriefe in bestimmte Postleitbereiche zu senden, um Portokosten zu sparen, oder alle Kunden anzuschreiben

die von einem bestimmten Vertreter betreut werden. Weiterhin kann das Selektieren einer Datei dazu dienen, Karteileichen aufzufinden, indem zum Beispiel alle Kunden einer Kundendatei herausgefiltert werden, deren letztes Rechnungsdatum vor einem bestimmten Stichtag liegt, oder bei Lagerartikeln Ladenhüter herauszufinden, wenn ein gewisser Umsatz unterschritten wurde, oder bei Handelsvertretern, die verschiedene Lieferfirmen vertreten, wenn es gilt alle Kunden herauszufinden, die Produkte von einem bestimmten Lieferanten bezogen haben. Aber auch Beispiele im privaten Bereich sind denkbar, wenn jemand seine Briefmarken per Computer katalogisiert, so ist mittels eines Selektierprogramms sehr schnell eine Fehlteilliste erstellt. Oder wenn jemand bei seiner Platten- bzw. Tonbandsammlung das Inhaltsverzeichnis mittels Rechner erstellt, ist es mit Selektieren sehr einfach, alle Titel eines Interpreten herauszufiltern. Oder.... .

Wie an diesen wenigen Beispielen zu ersehen ist, gibt es für das Selektieren vielfältige Möglichkeiten, den Anwendungen sind dabei keine Grenzen gesetzt. Was selektiert werden soll und aus welchen Datenbeständen ist vom jeweiligen Anwendungsfall abhängig. Aber eine generelle Vorgehensweise, wie man selektieren kann, wollen wir hier kurz beschreiben.

Wir wollen uns auf eine Dateiverwaltung beziehen, die dateibeschreibende Variablen enthält. Was mit den selektierten Daten anschließend geschieht, ob sie auf dem Drucker oder Bildschirm ausgelistet werden und in welcher Form, ist jeweiis von dem Listprogramm abhängig. Auch kann man über ein Selektierkriterium herausgefilterten Datensatz einer Datei in eine andere Datei ablegen, so daß man eine zweite Datei erhält, die vom Aufbau identisch ist mit der originalen Datei, jedoch nur Datensätze enthält, die dem gewünschten Selektierkriterium entsprechen.

Um selektieren zu können, muß man zum Einen wissen, in welchem Feld der Datei das Selektierkriterium enthalten ist und zum anderen, wie das Selektierkriterium für dieses Feld aussehen soll. Hier ein Kurzprogramm zum Erfassen des Selektierkriteriums:

Selektierkriterium für Zeichenreihen erfassen:

```
100 INPUT"Welches Feld enthält das Selektierkriterium";SK
110 IF SK LT 1 OR SK GT AW THEN 100
120 U1$="          11111111112222222223333333334"
130 U2$="12345678901234567890123456789012345678901"
```

```
140 PRINT LEFT$(U1$,LA(SK))
150 PRINT LEFT$(U2$,LA(SK))
160 GOSUB 'Cursor unter erstes Zeichen von U2$'
170 GOSUB 'Input-Routine'
180 FOR I = 1 TO LEN(IH$)
190    IF MID$(IH$,I,1)=" "THEN 210
200    VO=I: GOTO 230
210 NEXT I
220 GOSUB 'Fehler'
230 FOR J = LEN(IH$) TO I STEP -1
240    IF MID$(IH$,J,1)=" "THEN 260
250    LS=J-I+1: GOTO 270
260 NEXT J
270 SK$ = MID$(IH$,VO,LS)
```

Variablen:

SK$	=	Selektierkriterium
SK	=	zu selektierendes Element
VO	=	ab Zeichen VO in Element SK des Feldes suchen
LS	=	Anzahl der zu vergleichenden Zeichen ab Zeichen VO
AW	=	Anzahl der Elemente des Feldes
LA(I)	=	Länge eines Feldes

Beschreibung: Gegeben sei ein Feld von Zeichenreihen A$(), wobei die entsprechenden LA() die maximale Länge einer Zeichenreiche angeben. Zunächst wird die Nummer des Feldes erfaßt, in dem das Selektierkriterium enthalten ist, gefolgt von einer Plausibilitätsprüfung. Die Zeilen 120 bis 160 dienen zur Unterstützung der Manipulation am Bildschirm. Die in Zeile 170 angesprochene Input-Routine liefert eine Zeichenreihe, die das Selektierkriterium an der richtigen Stelle (ggf. mit führenden und nachfolgenden Leerzeichen) enthält.

In den Zeilen 180 bis 220 wird die Stelle des ersten Zeichens des Selektierkriteriums festgestellt, ggf. eine Fehlermeldung ausgegeben. Im weiteren wird die Länge des Selektierkriteriums festgestellt und das Selektierkriterium extrahiert.

In ähnlicher Weise kann das Erfassen für ein Selektieren nach Zahlen behandelt werden:

Selektierkriterium für Zahlen erfassen:

```
100 INPUT"Welches Feld enthält das Selektierkriterium";SK
110 IF SK LT 1 OR SK GT AW THEN 100
120 INPUT"Untergrenze  ( M = Minimum ) ";UG$
130 IF UG$="M" THEN UG = -10**LA(SK)
140 UG = VAL(UG$):IF UG = 0 THEN 120
150 INPUT"Obergrenze   ( M = Maximum ) ";OG$
160 IF OG$="M" THEN OG =  10**LA(SK)
170 OG = VAL(OG$):IF OG = 0 THEN 150
```

Variablen: SK = zu selektierendes Element
OG = Obergrenze
UG = Untergrenze
AW = Anzahl der Elemente des Feldes

Umfeld: Die Einträge der Datei sei - je Datensatz - in dem Feld IH$() abgelegt. Beschrieben werden die Daten durch ein Feld AR$() für jedes Element des Datensatzes (F-Floating Point, I-Integer) und LA(), das die maximale Größe der Zahl angibt.

Beschreibung: Nach Abfrage der Nummer des zu selektierenden Feldelementes und der Plausibilitätsprüfung werden die Unter- und Obergrenze des Selektierkriteriums erfaßt. Um auch das Minimum und Maximum (Eingabe 'M') zuzulassen, werden die Daten zunächst als Zeichenreihe erfaßt und mittels der VAL-Funktion umgewandelt.

Für Selektieren von Kalenderdaten sei hier auf Kapitel 1.6 hingewiesen.

Anhand der erfaßten Selektierkriterien läßt sich nun sehr einfach die eigentliche Selektion durchführen. Betrachten wir zunächst wieder die Zeichenreihen.

Selektieren von Zeichenreihen:

```
100 GOSUB 'Datei öffnen'
110 GOSUB 'Zeiger auf ersten Datensatz positionieren'
```

```
120 'lies AR aus erstem Datensatz der Datei'
130 FOR I = 2 TO AR
140    GOSUB 'Zeiger auf i-ten Datensatz'
150    GOSUB 'Datensatz in Feld IH$(AW) einlesen'
160    IF MID$(IH$(SK),VO,LS)=SK$ THEN GOSUB 'Auswerten'
170 NEXT I
180 GOSUB 'Datei schließen'
```

Beschreibung: Da es uns hier hauptsächlich um Selektieren geht, haben wir die 'Rahmenhandlung' in entsprechende Unterprogramme verwiesen. Wenn wir annehmen, daß die Anzahl zu prüfender Datensätze einer Datei im ersten Datensatz steht, so muß dieser zunächst ausgelesen werden. Die Anzahl der Records soll sich dann in der Variablen AR befinden.

In einer Schleife bis zu diesem Wert wird dann jeweils der Zeiger auf den Datensatz gesetzt, der Datensatz selbst eingelesen und dann der Vergleich durchgeführt. Was in dem Unterprogramm 'Auswerten' geschieht, bestimmen Sie selbst.

Analog erfolgt die Behandlung von Zahlen.

Selektieren nach Zahlbereichen

```
100 GOSUB 'Datei öffnen'
110 GOSUB 'Zeiger auf ersten Datensatz positionieren'
120 'lies AR aus erstem Datensatz der Datei'
130 FOR I = 2 TO AR
140    GOSUB 'Zeiger auf i-ten Datensatz'
150    GOSUB 'Datensatz in Feld IH$(AW) einlesen'
160    IF VAL(IH$(SK)) LT UG THEN 180
170    IF VAL(IH$(SK)) GT OG THEN 180
180    GOSUB 'Auswerten'
190 NEXT I
200 GOSUB 'Datei schließen'
```

Etwas schwieriger wird die Sache, wenn man zwei oder mehr Selektierkriterien für einen Datensatz hat. Hier ein Erfassungsprogramm für die Selektierkriterien und dann das zugehörige Selektiermodul, sofern die Selektierkriterien mit UND verknüpft sind. Zuerst auch wieder das Erfassen der Kriterien für Zeichenreihen:

Selektierkriterium für mehrfachen Stringvergleich erfassen:

```
100 INPUT"Anzahl der Selektierkriterien";AS
110 DIM SK(AS) : DIM SK$(AS) : DIM VO(AS) : DIM LS(AS)
120 FOR I = 1 TO AS
130    PRINT"Welches Feld enthält das ";I;"-te ";
140    PRINT"Selektierkriterium";
150    INPUT SK(I)
160    IF SK(I) LT 1 OR SK(I) GT AW THEN 130
170    U1$="         1111111111222222222233333333334"
180    U2$="12345678901234567890123456789012345678901234567890"
190    PRINT LEFT$(U1$,LA(SK(I)))
200    PRINT LEFT$(U2$,LA(SK(I)))
210    GOSUB'Cursor unter erstes Zeichen von U2$'
220    GOSUB 'Input-Routine'
230    FOR J = 1 TO LEN(IH$)
240       IF MID$(IH$,J,1)=" "THEN 260
250       VO(I)=J: GOTO 280
260    NEXT J
270    GOSUB 'Fehler' :REM Kein Selektierkriterium erfasst
280    FOR K = LEN(IH$)TO J STEP -1
290       IF MID$(IH$,K,1)=" "THEN 310
300       LS(I)=K-J+1: GOTO 320
310    NEXT K
320    SK$(I) = MID$(IH$,VO(I),LS(I))
330 NEXT I
```

Variablen:

AS	=	Anzahl der Kriterien
SK$(AS)	=	Selektierkriterien
SK(AS)	=	zu selektierende Elemente
VO(AS)	=	ab Zeichen VO in Element SK(AS) des Feldes suchen
LS(AS)	=	zu vergleichenden Zeichen ab Zeichen VO(AS)

Beschreibung: Im Prinzip wird hier nichts anderes gemacht, als die für das Selektieren wichtigen Variablen als Feld anzulegen und das Erfassen durch eine Schleife laufen zu lassen. Die Rahmenbedingungen sollen den vorher genannten gleich sein.

Selektierkriterium für mehrfachen Zahlenvergleich erfassen:

```
100 INPUT"Anzahl der Selektierkriterien";AS
110 DIM SK(AS) : DIM UG(AS) : DIM OG(AS)
120 FOR I = 1 TO AS
130    PRINT"Welches Feld enthält das ";I;"-te ";
140    PRINT"Selektierkriterium";
150    INPUT SK(I)
160    IF SK(I) LT 1 OR SK(I) GT AW THEN 130
170    INPUT"Untergrenze  ( M = Minimum ) ";UG$
180    IF UG$="M" THEN UG(I) = -10**LA(SK(I))
190    UG(I) = VAL(UG$):IF UG(I) = 0 THEN 170
200    INPUT"Obergrenze   ( M = Maximum ) ";OG$
210    IF OG$="M" THEN OG(I) =  10**LA(SK(I))
220    OG(I) = VAL(OG$):IF OG(I) = 0 THEN 200
230 NEXT I
```

```
Variablen:    AS      = Anzahl der Kriterien
              SK(AS)  = zu selektierende Elemente
              OG(AS)  = Obergrenzen
              UG(AS)  = Untergrenzen
```

Hier nun das Selektieren nach bekanntem Schema:

Stringvergleich (mehrfach)

```
100 GOSUB 'Datei öffnen'
110 GOSUB 'Zeiger auf ersten Datensatz positionieren'
120 'lies AR aus erstem Datensatz'
130 FOR I = 2 TO AR
140   GOSUB 'Zeiger auf i-ten Datensatz'
150   GOSUB 'Datensatz in Feld IH$(AW) einlesen'
160   FOR J = 1 TO AS            : REM Kriterienschleife
170     IF MID$(IH$(SK(J)),VO(J),LS(J))LTGTSK$(J) THEN 190
180   NEXT J
190   GOSUB 'Auswerten'
200 NEXT I
210 GOSUB 'Datei schließen'
```

Zahlenvergleich (mehrfach)

```
100 GOSUB 'Datei öffnen'
110 GOSUB 'Zeiger auf ersten Datensatz positionieren'
120 'lies AR aus erstem Datensatz'
130 FOR I = 2 TO AR
140    GOSUB 'Zeiger auf i-ten Datensatz'
150    GOSUB 'Datensatz in Feld IH$(AW) einlesen'
160    FOR J = 1 TO AS
170       IF VAL(IH$(SK(J))) LT UG(J) THEN 200
180       IF VAL(IH$(SK(J))) GT OG(J) THEN 200
190    NEXT J
200    GOSUB 'Auswerten'
210 NEXT I
220 GOSUB 'Datei schließen'
```

Gemischte Verfahren (gleichzeitig Zahlen und Zeichenreihen) sollen an dieser Stelle nicht besprochen werden, da Sie aus obigem Beispiel sehr einfach zusammenzusetzen sind.

Eine Verknüpfung von Selektierkriterien mit UND und ODER, und einer Verwendung von Klammern bei der Vergabe von Selektierkriterien läßt sich nur sehr aufwendig in allgemeiner Form programmieren. Da diese Fälle meist Spezialfälle sind, empfiehlt sich hier eine individuelle Programmierung.

Im Gegensatz zu den Möglichkeiten, die das Selektieren bei der Datenverwaltung bringt, ist der Aufwand des Programmierens relativ gering. Werden die selektierten Daten in einer extra Datei gespeichert, so steht auch einem anschließenden Sortiervorgang der selektierten Daten nichts im Wege.

1.5 Mischverfahren

Ein beim Mikro-Computer seltener auftretendes Problem, ist das Sortieren von Daten durch Mischen, da im allgemeinen die Datenbestände nicht so groß sind und Magnetbandgeräte

nicht verwendet werden. Arbeitet man jedoch mit Kassettenrecordern und hat man Daten zu sortieren, die nicht direkt im Rechner sortiert werden können, so können die nachfolgenden Verfahren dazu herangezogen werden. Entstanden sind sie zu Anfang der Datenverarbeitung als eine Dialogbearbeitung noch nicht gebräuchlich war. Die nachfolgenden Verfahren sind historisch auf der Stapelverarbeitung (BATCH-Prozessing) begründet.

Bei dem Mischverfahren wird davon ausgegangen, daß nur ein kleinerer Teil der Daten im Arbeitsspeicher bearbeitet werden kann, als auf dem externen Speicher, Kassettenrecorder bzw. Bandgeräte zur Verfügung stehen. Aber die Verfahren können auch angewendet werden, wenn maximale Dateigrößen überschritten werden, bzw. die Kapazität einer Diskette für die Daten nicht ausreicht. Trotzdem kann man annehmen, daß die Daten als kontinuierlicher Fluß (ggf. vorsortiert) zur Verfügung stehen, was durch Programme bzw. das Betriebssystem gesteuert werden kann.

Die Grundeinheit dieses Verfahrens wird **Lauf** genannt, und einen Lauf wollen wir als maximale Sortierfolge von Daten aus einer Datenfolge betrachten. Als Ausgangspunkt nehmen wir an, daß auf n-Bändern A(n)-Läufe vorhanden sind, wobei die Anzahl der Elemente in jedem Lauf auch verschieden sein darf. Festgestellt wird weiterhin, daß mehrere Ausgabe- bzw. Eingabemedien vorhanden sind (z.B. mindestens zwei Diskettenlaufwerke oder zwei Kassettenrecorder). Ist dies nicht der Fall, so muß man diese verschiedenen Ausgabemedien im Arbeitsspeicher simulieren, was natürlich die maximale Größe der einzelnen Läufe beeinträchtigt.

Der Ablauf stellt sich dann wie folgt dar: Man liest von allen Eingabemedien jeweils den ersten Datensatz und vergleicht diese Datensätze untereinander. Dann wählt man den Kleinsten aus und schreibt diesen auf das Ausgabeband. Von dem Band, wo das ausgegebene Element herrührte, liest man nun den nächsten Datensatz ein. Dann verfährt man mit den eingelesenen Datensätze wie vorher und sucht sich das kleinste Element heraus. Ist ein Lauf zu Ende, so wird dieses Band übergangen, bis alle Läufe zu Ende sind. Auf diese Weise entsteht ein Ausgabeband mit einem wesentlich längeren Lauf, der gegebenenfalls wieder gestückelt werden muß. Dieses Verfahren wird sooft wiederholt, bis alle einzulesenden Bänder verarbeitet sind.

Sind alle Eingabebänder abgearbeitet, so vertauscht man die Ausgabebänder mit den Eingabebändern. Auf den neuen Eingabebändern stehen dann Läufe mit größerer Länge. Mit jedem Tausch Ausgabebänder gegen Eingabebänder werden die Läufe größer und die Anzahl der Läufe kleiner. Dies wird

solange fortgesetzt, bis nur noch ein Lauf existiert, der dann die komplette sortierte Datenfolge beinhaltet. Bild 1.5 veranschaulicht das Verfahren.

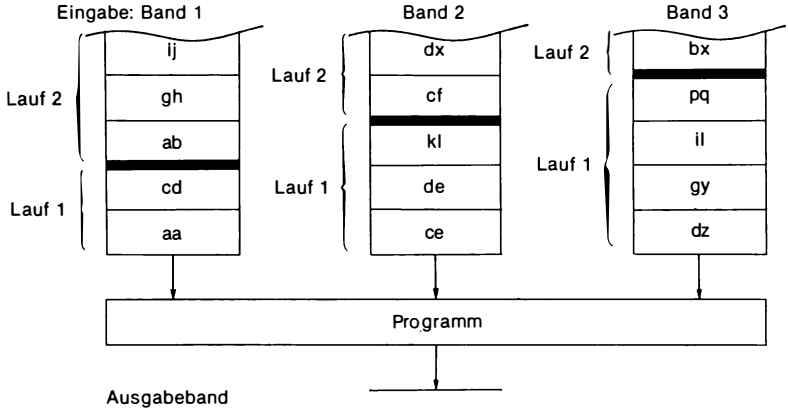


Bild 1.5-1 : Datenbestand vor Mischvorgang

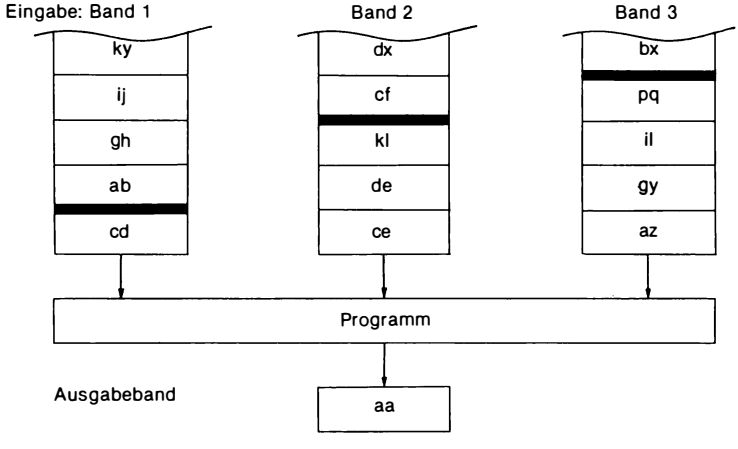


Bild 1.5-2 : Datenbestand nach erstem Mischvorgang

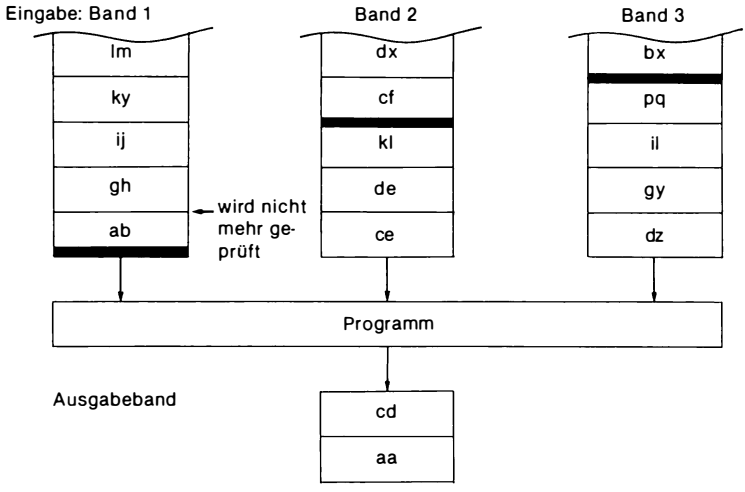


Bild 1.5-3 : Datenbestand nach zweitem Mischvorgang

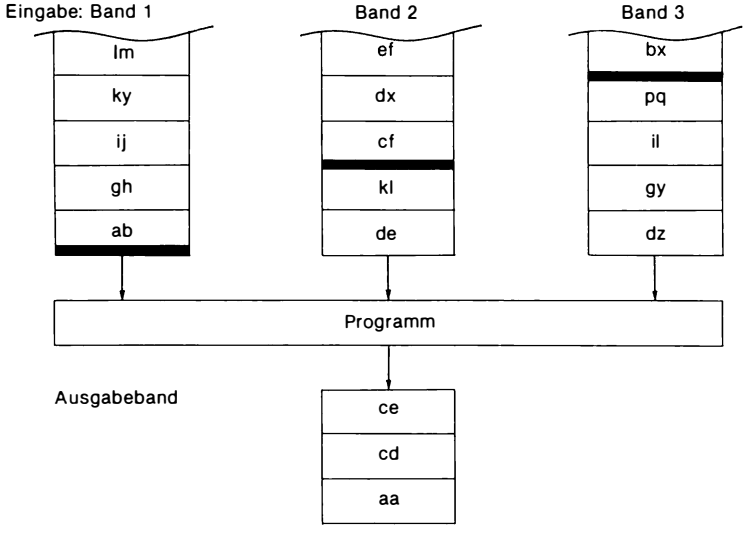


Bild 1.5-4 : Datenbestand nach drittem Mischvorgang

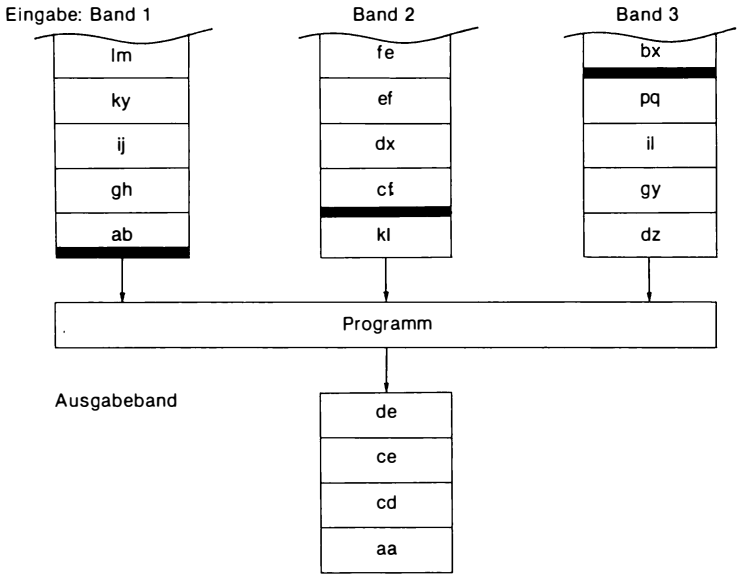


Bild 1.5-5 : Datenbestand nach viertem Mischvorgang

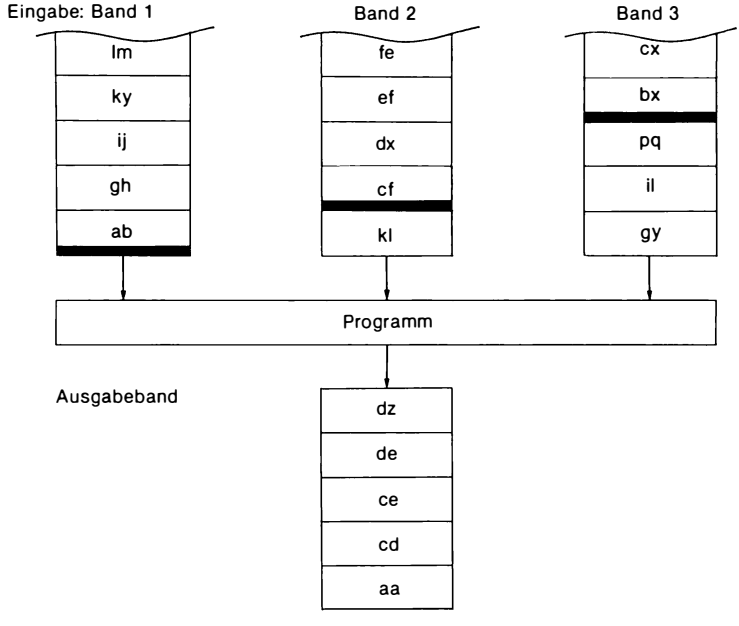


Bild 1.5-6 : Datenbestand nach fünftem Mischvorgang

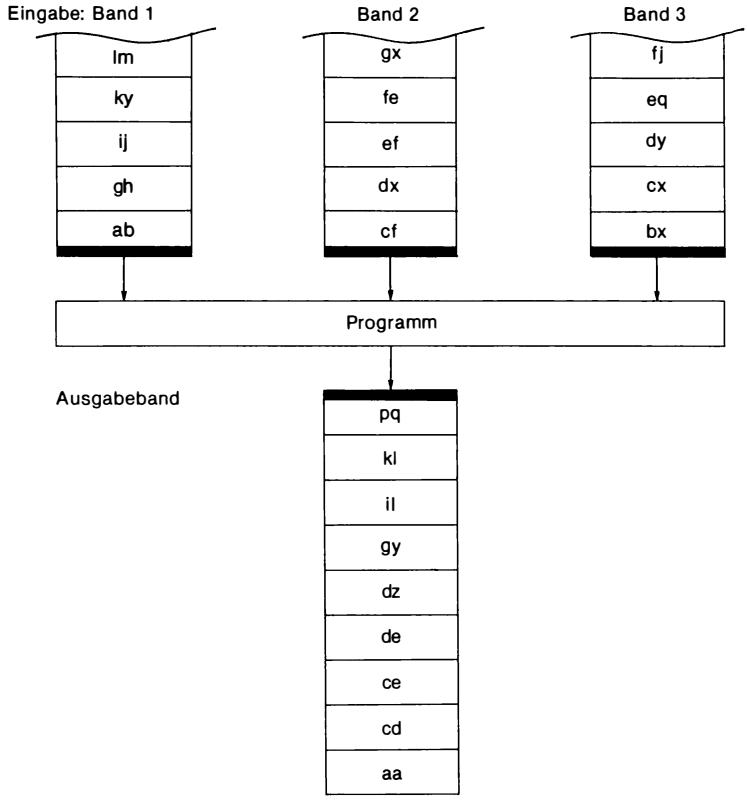


Bild 1.5-7 : Datenbestand nach dem neunten Mischvorgang. Auf dem Ausgabeband befindet sich nun ein kompletter Lauf, und alle Eingabebänder beginnen mit einem neuen Lauf. Das Programm wird jetzt erneut aufgerufen, sofern auf dem Ausgabeband noch genügend Platz ist.

Ein Beispielprogramm zum Mischen mit drei Eingabemedien und einem Ausgabemedium könnte in Pseudo-Basic wie folgt aussehen:

```
100 GOSUB 'Dateien öffnen'
110 N=3
120 DIM VK$(N)
```

```
130 DIM LE(N)
140 FOR I=1 TO N
150    INPUT#I,VK$(I)
160 NEXT
170 VK$="ZZZZZZ"
180 LF=0
190 FOR I=1 TO N
200    IF LE(I) GT 0 THEN 230
210    LF=1
220    IF VK$(I) LT VK$ THEN VK$=VK$(I) : WO=I
230 NEXT
240 PRINT#A,VK$
250 IF LF=0 THEN RUN
260 AL$=VK$
270 INPUT#WO,VK$(WO)
280 IF VK$(WO) LT AL$ THEN LE(WO)=1
290 GOTO 170
```

Beschreibung: Nachdem alle benötigten Dateien geöffnet sind, wird die Anzahl der Eingabestationen erfaßt. Für jede Eingabestation wird dann jeweils ein Feldelement zur Aufnahme des aktuellen Vergleichelementes und zur Kennzeichnung des Laufendes definiert.

Dann wird von jeder Eingabestation das erste Element gelesen. Weitere Vorarbeiten sind das Besetzen des Vergleichskriteriums mit einem maximalen Wert und das Definieren des allgemeinen Endes (aller Läufe).

Ab Zeile 190 werden die aktuellen Elemente aller Läufe geprüft. Ist ein Lauf bereits beendet, so wird er übergangen. Sind alle Läufe beendet, so ist LF beim Verlassen der Schleife '0', sonst '1'. In Zeile 240 wird das kleinste gefundene Element auf das Ausgabeband übertragen, anschließend geprüft, ob alle Läufe beendet sind und gegebenenfalls ein neues Element eingelesen. Dies muß natürlich aus dem Lauf genommen werden, wo das eben ausgegebene herstammt.

Da vielleicht auf einem Eingabeband mehrere Läufe hintereinander liegen, wird in Zeile 280 geprüft, ob der aktuelle Lauf zu Ende ist und entsprechend der Merker LF(WO) gesetzt.

Das Erzeugen von n Bändern mit A(n)-Läufen kann auf Grund der in Kapitel 1.3 beschriebenen Sortierverfahren durchgeführt werden, indem man zum Beispiel 500 Datensätze einliest, diese mit einem internen Sortierverfahren sortiert und dann auf ein Band schreibt. Mit dem internen Sortieren sollte man möglichst die gesamte Arbeitsspeicherkapazität des Rechners ausnutzen, um die Anzahl der Läufe klein zu halten.

Dieses Verfahren kann auch mit Disketten angewendet werden, da z.B. sequentielle Dateien auch als 'Eingabebänder' verstanden werden können. Dies trifft außerdem für die sequentielle Bearbeitung von Direktzugriffsdateien zu. Haben Sie z.B. - wegen der geringen Diskettenkapazität - mehrere Disketten mit gleichartigen Daten, so wurde hier ein Weg aufgezeigt, auch diese Daten zu sortieren.

1.6 Rund ums Datum

Als 'vierten Variablentyp' kann man die Kalenderdaten (neben Fließkommazahlen, Integer und Zeichenreihen) bezeichnen, obwohl sie in der Regel als Zeichenreihen gespeichert werden. Aber gerade im kommerziellen Bereich kommen sie sehr häufig vor, und erfordern immer eine Spezialbehandlung, dies gilt sowohl für die Plausibilitätsprüfung, als auch für das Sortieren, da meist die Daten nach Tag/Monat/Jahr bearbeitet werden, aber nach Jahr/Monat/Tag sortiert werden soll. In den folgenden Kapiteln soll als computergerechte Eingabe immer ein Kalenderdatum in der Form TT.MM.JJ vorliegen, wobei TT den aktuellen Tag (ggf. mit führender '0'), MM den Monat (auch ggf. mit führender '0') und JJ die letzten beiden Ziffern des Jahres darstellen sollen.

1.6.1 Plausibilitätsprüfung

Bei der Verarbeitung von Kalenderdaten kommt der Plausibilitätsprüfung eine entscheidende Bedeutung zu. Die Richtigkeit der Eingabe eines Kalenderdatums läßt sich jedoch nur innerhalb der zulässigen Tage prüfen. Innerhalb dieser Grenzen ist eine Plausibilitätsprüfung nur in Spezialfäl-

len möglich; deshalb wollen wir uns hier auf die allgemeine Prüfung beschränken.

Geprüft werden die Eintragungen für Tag, Monat und Jahr jeweils einzeln. Zur Prüfung der gültigen Tageseingabe anhand des Monates soll ein Hilfsfeld mit den maximalen Tagen eines Monates definiert werden.

```
100 DATA 31,28,31,30,31,30,31,31,30,31,30,31
110 DIM MM%(12) : FOR I = 1 TO 12 : READ MM%(I) : NEXT
```

Die Prüfung für einen gültigen Monat kann nur auf 'größer 12' und 'kleiner 1' erfolgen, und eine Prüfung des Jahres muß nicht immer sinnvoll sein und kann in der Regel auch nur gewünschte Jahresbereiche ausklammern. Wir wollen davon ausgehen, daß kein früheres Jahr, als das des aktuellen Tagesdatums (DA$ in der Form: TT.MM.JJ) eingegeben werden darf. Soll gleiches auch für Monat und Tag gelten, sind nur entsprechende zusätzliche Abfragen einzubauen.

Die einzige Schwierigkeit wäre noch der 29.Februar eines Schaltjahres, der als Tag zulässig ist, bei allgemeiner Vorgehensweise jedoch nicht akzeptiert würde. Das zu prüfende Datum soll in der Form TT.MM.JJ in der Variablen DP$ abgespeichert sein. Die Prüfroutine soll als Unterprogramm angelegt sein, und in der Variablen WF (Wahr/Falsch) eine '0' enthalten, wenn das zu prüfende Datum korrekt ist, und eine '1', wenn ein Fehler festgestellt wurde (LT less than/ kleiner; GT greater than/größer; LE less or equal/ kleiner gleich; GE greater or equal/größer gleich):

```
1000 SJ = 0
1010 T = VAL(LEFT$(DP$,2))  :REM Tageszahl festlegen
1020 M = VAL(MID$(DP$,4,2)) :REM Monatszahl festlegen
1030 J = VAL(RIGHT$(DP$,2)) :REM Jahreszahl festlegen
1040 JD= VAL(RIGHT$(DA$,2)) :REM Jahreszahl d. Tagesdatums
1050 IF J LT JD THEN 1120
1060 IF J/4 - INT(J/4)=0 THEN SJ = 1
                             :REM Schaltjahr liegt vor
1070 IF M = 2 AND T = 29 AND SJ = 1 THEN 1110
1080 IF M = 2 AND T = 29 AND SJ = 0 THEN 1120
1090 IF M LT 1 OR M GT 12 THEN 1120
1100 IF T LT 1 OR T GT MM%(M) THEN 1120
1110 WF = 0 : RETURN
1120 WF = 1 : RETURN
```

Darf das Prüfdatum nicht vor dem Tagesdatum liegen, so ist einzufügen:

```
1041 MD = VAL(MID$(DA$,4,2))
```

```
1042 TD = VAL(LEFT$(DA$,2))
1101 IF J = JD AND M LT MD THEN 1120
1102 IF J = JD AND M = MD AND T LT TD THEN 1120
```

Gegebenenfalls sind die Variablen im Unterprogramm durch andere zu ersetzen, wenn sie im Hauptprogramm an anderer Stelle vorkommen (z.B. J als Laufvariable in einer FOR... NEXT-Schleife).

Die Prüfung auf Schaltjahre wurde in der einfachsten Form vorgenommen, was im Normalfall ausreicht. Ist eine genauere Berechnung der Schaltjahre erforderlich, so sollte man diese in ein eigenes Unterprogramm bringen, falls die maximal zulässige Unterprogramm-Hierarchie nicht überschritten wird. In den Jahren 1700,1800,1900 führt obige Berechnung auf ein Schaltjahr, obwohl diese keine Schaltjahre sind. Von den vollen Jahrhunderten sind nur diese Schaltjahre, deren **zwei ersten** Ziffern durch vier teilbar sind (was im Jahr 4900 auch nicht mehr stimmt).

Weiterhin ist bei Datums-Berechnungen, die mehr als 400 Jahre zurückliegen, darauf zu achten, daß auf den 4.10. 1582 der 15.10.1582 folgt, wenn man vom Gregorianischen Kalender ausgeht.

1.6.2 Datum platzsparend speichern

Bei Mikrocomputern (und Home-Computern im besonderen) ist in der Regel der externe Speicherplatz (Floppy-Disk) ein großes Problem. Wenn viele Kalenderdaten gespeichert werden müssen, so kann man mit meinem kleinen Trick den Platzbedarf pro Kalenderdatum auf ca. 37 % verringern.

Der Rechner speichert sich alle Zeichen im ASCII-Code, d.h. als Binärzahl. Ein 'a' wird z.B. als '65' (01000001) gespeichert. Da auch Steuerzeichen im ASCII-Code gespeichert werden (z.B. Bildschirm löschen), ist hier jedoch Vorsicht geboten. Wir wollen den Buchstabenbereich 65 (a) bis 90 (z) und die angrenzenden Sonderzeichen benutzen, die auch vollkommen ausreichen.

Gegeben sei ein Datum der Form TT.MM.JJ in der Variablen DA$. Weiter bedeuten:

```
T = VAL(MID$(DA$,1,2)) : Tageszahl
M = VAL(MID$(DA$,4,2)) : Monatszahl
```

```
J = VAL(MID$(DA$,7,2)) : Jahreszahl (ohne '19')
```

T kann einen Wert zwischen 1 und 31 annehmen. Wenn wir nun zu T 65 addieren, liegt T im o.a. Bereich. M kann einen Wert zwischen 1 und 12 annehmen, und bei J kann man davon ausgehen, daß es im Bereich 70 bis 95 liegt.

Das gekürzte Datum wollen wir in DK$ ablegen und erhalten hierfür:

```
DK$ = CHR$(T+65)+CHR$(M+65)+CHR$(J)
```

Beispiel: Der 15.05.84 wird gespeichert als 'pft' mit

```
ASC ("p") = 80 (65+15)
ASC ("f") = 70 (65+5)
ASC ("t") = 84
```

Entschlüsselt wird das Ganze mit:

```
T = ASC (LEFT$(DK$,1)) - 65
M = ASC (MID$(DK$,2,1)) - 65
J = ASC (RIGHT$(DK$,1)

DA$ = RIGHT$(STR$(T),2)+"."+RIGHT$(STR$(M),2)+"."
      +RIGHT$(STR$(J),2)
```

In einer Auftragsbearbeitung, in der Bestelldatum, Lieferdatum, Rechnungsdatum und Zahlungsdatum gespeichert werden, reduziert sich der Platzbedarf von 32 Byte je Auftrag auf 12 Byte je Auftrag. Bei 1000 zu speichernden Aufträgen sind dies 20 KByte.

1.6.3 Nach Datum sortieren

Eine weitere Besonderheit des Kalender-Datums ist der Sortiervorgang. Zeichenreihen werden alphanumerisch sortiert, und Dezimalzahlen sowie ganze Zahlen (Integer) werden ihren Werten nach sortiert. Beides ist mit den üblichen BASIC-Befehlen und den Vergleichsoperatoren 'Größer', 'Kleiner' und 'Gleich' leicht zu bewerkstelligen. Sollen jedoch Kalenderdaten sortiert werden, so ist eine Sortie-

rung nach Jahr/Monat/Tag erforderlich - wenn man von Ausnahmefällen absieht. Die gleichen Schwierigkeiten treten beim Selektieren von Kalenderdaten auf, wenn nicht nach bestimmten Daten gesucht wird (Operator: =) sondern nach Zeiträumen (Operatoren 'Größer' und 'Kleiner').

Normalerweise muß dazu jedes Kalenderdatum umgruppiert werden wie in folgenden Beispielen:

```
(1) DN$ = RIGHT$(DA$,2) + MID$(DA$,3,4) + LEFT$(DA$,2)

   oder

(2) DN$ = RIGHT$(DA$,2) + MID$(DA$,4,2) + LEFT$(DA$,2)
```

Das Ergebnis z.B für den 10.09.84 wäre nach (1) '84.09.10' und nach (2) '840910', wobei das zweite Ergebnis - für den Sortiervorgang sehr wichtig - platzsparender ist, und auch als Zahl sortiert werden kann.

Nützt man die vorher beschriebene platzsparende Speicherung des Datums in 3 Byte aus, und ersetzt man das Zusammenstellen der Kurzspeicherform durch

```
DK$ = CHR$(J) + CHR$(M+65) + CHR$(T+65)
```

und das Dekodieren durch

```
T = ASC(RIGHT$(DK$,1))-65
M = ASC(MID$(DK$,2,1))-65
J = ASC(LEFT$(DK$,1))
```

so wird das Datum sozusagen rückwärts gespeichert, und kann in seiner Kurzform wie eine normale Zeichenreihe sortiert werden. Der 15.05.82 - um das alte Beispiel aufzugreifen - würde gespeichert als "rfp".

1.6.4 Selektieren durch Vergleich

Sehr häufig werden auch bestimmte Datensätze gesucht, die ein Kalenderdatum erhalten, das in einem gewissen Zeitraum liegen soll (z.B. Rechnungsdatum zwischen 01.12.83 und 31.12.83). Dazu wollen wir in einem Beipielprogrammstück zwei Hilfsvariablen V1$ und V2$ verwenden, wobei in V1$ das ältere und in V2$ das neuere Datum in der gleichen Form wie bei DA$ gespeichert sein sollen.

Das beschriebene Unterprogramm muß beim Durchsuchen einer Datei mit jedem Datensatz neu aufgerufen werden und liefert als Ergebnis in 'JN' den Wert '1', wenn das Datum im gesuchten Bereich liegt, andernfalls '0'. Wird nur nach einem Zeitraum vor oder nach einem Datum gefragt, so sind die entsprechenden Anweisungen für den anderen Fall wegzulassen, oder man kann das Vergleichsdatum auf 00.00.00 bzw. 31.12.99 setzen.

```
100 JN = 0
110 T = VAL(LEFT$(DA$,2))     : REM Tagesdatum auseinander-
120 M = VAL(MID$(DA$,4,2))    : REM ziehen, man kann aber
130 J = VAL(RIGHT$(DA$°2))    : REM auch mit den Strings
140 T1 = VAL(LEFT$(V1$,2))    : REM arbeiten
150 M1 = VAL(MID$(V1$,4,2))   : REM älteres Vergleichsdatum
160 J1 = VAL(RIGHT$(V1$,2))
170 T2 = VAL(LEFT$(V2$,2))    : REM neueres Vergleichsdatum
180 M2 = VAL(MID$(V2$,4,2))
190 J2 = VAL(RIGHT$(V2$,2))
200 IF J LT J1 THEN RETURN    : REM Jahr zu alt
210 IF J GT J2 THEN RETURN    : REM Jahr zu jung
220 IF J = J1 AND M LT M1 THEN RETURN
230 IF J = J2 AND M LT M2 THEN RETURN
240 IF J = J1 AND M = M1 AND T LT T1 THEN RETURN
250 IF J = J2 AND M = M2 AND T GT T2 THEN RETURN
260 JN = 1
270 RETURN
```

1.7 Boolsche Operatoren

Neben den Vergleichsoperatoren 'gleich', 'kleiner', 'größer', 'kleiner gleich', 'größer gleich' und 'ungleich' sind in den verschiedenen Programmiersprachen und deren Dialekten noch mindestens zwei weitere Operatoren, sogenannte boolsche Operatoren, implementiert : AND und OR um logische UND oder ODER-Verknüpfungen vornehmen zu können. Häufig (auch beim Commodore 64) findet sich auch noch die Negation: NOT.

Sehr oft werden diese Operatoren in Verbindung mit den Vergleichsoperatoren bei Plausibilitätsprüfungen oder sonstigen bedingten Abfragen verwendet. Die häufigste Plausibilitätsprüfung ist wohl die Abfrage einer Eingabe, ob ein JA ('j' oder 'J') oder ein NEIN ('n' oder 'N') vorliegt. Ohne boolsche Operatoren sähe eine solche Plausibilitätsprüfung wie folgt aus:

```
100 IF A$ = "j" THEN GOTO 'Ja - Teil'
110 IF A$ = "J" THEN GOTO 'Ja - Teil'
120 IF A$ = "n" THEN GOTO 'Nein - Teil'
130 IF A$ = "N" THEN GOTO 'Nein - Teil'
140 ' Eingabefehler '
```

Bei Verwendung von boolschen Operatoren könnte man schreiben:

```
100 IF A$ = "j" OR A$ = "J" THEN GOTO 'Ja - Teil'
110 IF A$ = "n" OR A$ = "N" THEN GOTO 'Nein - Teil'
120 ' Eingabefehler '
```

Wenn man die Plausibilitätsprüfung ohne Verzweigung alleine betrachtet, kann man schreiben (NE für ungleich/not equal):

```
100 IF A$ NE "j" AND A$ NE "J" AND A$ NE "n" AND A$ NE "N"
       THEN GOTO 'Fehler'
```

Schon aus diesen kleinen Beispielen wird ersichtlich, welche Funktion die boolschen Operatoren haben.

Ein weiteres Einsatzgebiet der boolschen Operatoren ist die bitweise Bearbeitung von Daten (z.B. das Ausblenden verschiedener Bits, wie es für Grafik und Sprites benötigt wird). Dazu muß man jedes Zeichen oder jede Zahl in ihr Bitmuster zerlegen, dem eine Zahl zwischen 0 und 255 entsprechend dem Bitmuster zugeordnet werden kann. Dazu werden die Stellenwerte jedes einzelnen Bits wie folgt addiert, wie auf der nächsten Seite dargestellt.

Die Zahlen werden den Bits von hinten nach vorne zugeordnet, d.h. die letzte Stelle entspricht 2 hoch 0 und dementsprechend die erste Stelle 2 hoch 7.

Beispiel:

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

entspricht dem Wert 128+32+4 = 164.

		für 16-Bit-Darstellung:		
2^0	= 1		2^8	= 256
2^1	= 2		2^9	= 512
2^2	= 4		2^{10}	= 1024
2^3	= 8		2^{11}	= 2048
2^4	= 16		2^{12}	= 4096
2^5	= 32		2^{13}	= 8192
2^6	= 64		2^{14}	= 16384
2^7	=128		2^{15}	= 32768

Summe 255 (wenn alle 8 Bits gesetzt sind)

Zwei mit AND verknüpfte Werte ergeben den Zahlwert der Summe aller Bits, die in beiden Werten vorkommen:

Beispiel:

A = 164	(10100100)
B = 36 (32 + 4)	(00100100)
A AND B = 36	(00100100)
A = 164	(10100100)
B = 72 (64 + 8)	(01001000)
A AND B = 0	(00000000)

Zwei mit OR verknüpfte Werte ergeben den Zahlwert der

Summe aller Bits , die in dem einen oder dem anderen Wert vorkommen:

Beispiel:

```
A = 164                    ( 10100100 )
B = 36                     ( 00100100 )

A OR B = 164               ( 10100100 )

A = 164                    ( 10100100 )
B = 72                     ( 01001000 )

A OR B = 236               ( 11101100 )
```

Arbeitet man mit Zeichen, so läßt sich das Bitmuster über den Umweg der Funktion ASC(X$) ermitteln. Diese Funktion liefert den zu dem Zeichen gehörenden ASCII-Zahlwert, der zwischen 0 und 255 liegt, worauf wieder auf das Bitmuster geschlossen werden kann.

In den folgenden Programmen - besonders Kapitel 2.2 - werden Sie viele Beispiele für die Verwendung von boolschen Operatoren finden.

1.8 Führende Nullen anfügen

Führende Nullen sind in der Regel zwar nicht üblich, jedoch können Sie bei Verwendung von Zahlen als Auftragsnummer oder Rechnungsnummer durchaus ihren Sinn haben, um das Voranstellen von Ziffern zu vermeiden.

Im folgenden bedeuten:

A = natürliche Zahl (größer 0, ohne Dezimalstellen)
N = gewünschte Gesamtzahl der Ziffernstellen

```
100 A$ = STR$(A)
110 IFLEN(A$) GT N+1 THEN PRINT "FEHLER"
120 A$ = RIGHT$("OOOOOOOOO"+MID$(A$,2)
```

Es ist zu beachten, daß nicht nur führende Nullen anzufügen sind, sondern auch das (i.a. positive) Vorzeichen eleminiert werden muß, um keine Lücke zu erhalten.

1.9 Eliminieren anhängender Leerzeichen

In der kommerziellen Datenverarbeitung werden häufig Direktzugriffsdateien verwendet, die nur eine konstante Datensatzlänge zulassen. Um diese zu erreichen, und um gleichartige Einträge immer an der gleichen Stelle innerhalb des Datensatzes zu finden, werden in der Regel (Name, Vorname, Straße, etc) mit Leerzeichen aufgefüllt, z.B. mit:

```
100 H$ = "                         "
110 NA$ = LEFT$(NA$+H$,25)
```

Will man nun Name und Vorname in der gleichen Zeit drucken, etwa mit

```
        PRINT NA$;',';VN$
```

so erhält man vielleicht

```
        MÜLLER           ,PETER
```

was sicherlich nicht beabsichtigt ist. Dies kann man mit einer kleinen Schleife umgehen:

```
100 FOR I=LEN(NA$) TO 1 STEP -1
200 IF RIGHT$(NA$,1)=" " THEN
                  NA$ = LEFT$(NA$,LEN(NA$)-1):NEXT
300 PRINT NA$;',';VN$
```

1.10 Kaufmännische Zahldarstellung

Bei Computern, die in ihrer Programmiersprache kein 'PRINT USING' aufweisen oder bei denen formatiertes Drucken nicht möglich ist, ergibt sich immer das Problem der kaufmännischen Zahldarstellung, da der Computer mit einem Dezimalpunkt statt mit Dezimalkomma arbeitet, keinen Tausender-

Punkt kennt und in der Regel anhängende Nullen unterdrückt.

In 'A' soll eine bereits gerundete Zahl in Computerdarstellung abgelegt sein, die in eine Zeichenreihe 'A$' umgewandelt werden soll, die die kaufmännische Zahldarstellung enthält. Dazu gehen wir wie folgt vor:

```
100 A$  = STR$(A)                                                  <209>
110 VZ$ = LEFT$(A$,1)                                              <207>
120 A$  = RIGHT$(A$,LEN(A$)-1)                                     <143>
130 Y   = ABS(A)                                                   <213>
140 IF Y<1 AND Y>0 THEN A$ = "0"+A$                                <126>
150 IF Y-INT(Y) < 0.001 THEN A$=A$+".00" : GOTO 170                <032>
160 IF Y*10 - INT(Y*10) < 0.001 THEN A$ = A$+"0"                   <139>
170 A$  = RIGHT$("              "+A$,14)                           <191>
180 A$=LEFT$(A$,11)+","+RIGHT$(A$,2)                               <120>
190 IF Y>999.99 THEN A$=LEFT$(A$,8)+"."+RIGHT$(A$,6)               <229>
200 IF Y>999999.99 THEN A$=LEFT$(A$,5)+"."+RIGHT$(A$
    ,10)                                                           <194>
210 A$  = A$+VZ$                                                   <204>
220 PRINT A$                                                       <218>
230 RETURN                                                         <116>
```

In Zeile 100 wird A$ mit dem Wert von A vorbesetzt und in Zeile 110 wird das Vorzeichen extrahiert. Im weiteren wird an Y der Absolut-Betrag von A zugewiesen.

Die eigentliche Umwandlung beginnt in Zeile 140 mit der Behandlung des Falles, daß ein Wert zwischen '0' und '1' vorliegt, der Computer also keine Ziffer vor dem Dezimalpunkt vermerkt hat. Zeile 150 behandelt das Fehlen von Nachkommastellen. Da die INT-Funktion in manchen Fällen nicht richtig arbeitet, sollte ein Test auf Gleichheit vermieden werden.

In Zeile 160 wird in ähnlicher Weise der Fall nur einer Nachkommastelle behandelt und anschließend wird die jetzt erhaltene Zeichenreihe auf 14 Zeichen Länge normiert.

Ab Zeile 180 wird der Dezimalpunkt durch ein Komma ersetzt und die entsprechenden 'Tausender-Punkte' werden eingefügt. Als Abschluß wird in Zeile 210 das Vorzeichen als letztes Zeichen angefügt.

1.11 Runden

Im folgenden bedeuten:

A = Zahl, die gerundet werden soll
N = Anzahl der Nachkommastellen
** = Potenzierungsoperator 'hoch'

```
100 A = INT ( A * 10 ** N + 0.5 ) / 10 ** N
```

Für N = 2 ergibt sich die übliche kaufmännische 5/4-Rundung auf zwei Dezimalstellen:

```
100 A = INT(A*100+0.5)/100
```

Beispiel: A = 125.25896

```
A * 100                    = 12525.896
A * 100 + 0.5              = 12526.396
INT ( A * 100 + 0.5        = 12526
INT ( A * 100 + 0.5)/100   = 125.26
```

Das Runden wird im Prinzip auf ein Abschneiden der letzten Ziffern zurückgeführt, was die Funktion INT(X) bewirkt. Um an der richtigen Stelle den Schnitt durchzuführen, muß zuerst mit der entsprechenden Zehnerpotenz multipliziert werden. Um eine korrekte 5/4 Rundung zu erreichen, wird der Wert 0.5 addiert, womit sich das Runden auf die Schnittstelle verschiebt. Anschließend wird wieder durch die entsprechende Zehnerpotenz dividiert.

1.12 Präfix-Suche

Nicht immer ist es beim Suchen in Datensätzen (besonders bei Verwendung von Schlüsseln) erforderlich, daß der Suchbegriff vollständig eingegeben wird. Sucht man nach einem

längeren Namen, reichen die ersten Buchstaben zu eindeutiger Identifizierung meist aus. Nimmt man an, daß in A$ jeweils der zu überprüfende Name (innerhalb einer Such-Schleife) steht, und in SU$ der Suchbegriff, so führt bei A$="Kowalczyk" und SU$="Kow" die bedingte Abfrage 'IF A$ = SU$ THEN ...' allerdings nicht auf die gewünschte Gleichheit.

Dies kann man jedoch dadurch umgehen, indem man in A$ nur die Anzahl der Zeichen überprüfen läßt, die SU$ aufweist:

```
IF LEFT$(A$,LEN(SU$))=SU$ THEN ...
```

was das gewünschte Ergebnis bringt.

Sind aber A$ Und SU$ bis auf eine bestimmte Länge (z.B. 20) mit Leerzeichen aufgefüllt, d.h. A$="Kowalczyk " und SU$ ="Kow " wenn man eine Input-Routine verwendet, so führt obiges Verfahren auch nicht zum Ziel.

Abhilfe schafft hier die Eingabe eines Sonderzeichens (z.B.'*') nach dem letzten gültigen Eingabezeichen im Suchbegriff :

```
SU$ = "Kow*                  ".
```

Daraufhin muß zuerst in einer Schleife die Position des Sonderzeichens gesucht werden, bevor die Abfrage erfolgt, was aus Rechenzeitgründen außerhalb der eigentlichen Suchschleife erfolgen sollte:

```
100 N = 20
110 FOR PO = 1 TO N
110 IF MID$(SU$,PO,1) NE "*" THEN NEXT PO


200 'Such - Schleife'
.
.
.
300 IF LEFT$(A$,PO-1)=LEFT$(SU$,PO-1) THEN GOTO 'Gefunden'
.
.
.
```

Von der Position des Sonderzeichens muß bei der Teilstringbildung noch eine Stelle abgezogen werden, damit das Sonderzeichen nicht mit überprüft wird, was nie zu einem brauchbaren Ergebnis führen kann. Das Sonderzeichen darf

selbstverständlich nicht in den Daten vorkommen.

Ein weiteres Problem stellt die Ähnlichkeitssuche dar. Meier, Maier, Mayer, Meyer und Mayr sind für den Rechner unterschiedliche Namen, wo er auch keine Assoziation bilden kann. Hier ist die Zulassung eines weiteren Sonderzeichens (z.B. '?') sehr hilfreich, wenn eine zeichenweise Überprüfung der Vergleichsdaten erfolgt.

Wenn man nicht genau weiss, ob ein gesuchter Meier auch als Meyer oder als Maier gespeichert ist, genügt dann die Eingabe von SU$="M??er", wenn folgendes Programmstück zum Suchen implementiert ist:

```
100 N = 'Länge'
110 FOR I = 1 TO N
120 IF MID$(SU$,I,1)="?"THEN GOTO 150
130 IF MID$(A$,I,1) = MID$(SU$,I,1) THEN GOTO 150
140 GOTO 'Nicht ähnlich'
150 NEXT
160 ' Zeichenreihen sind im gegebenen Umfang ähnlich
```

Beide hier dargestellten Verfahren können auch kombiniert werden, so daß auch eine Abfrage SU$ = "M??e*" möglich ist. Als Eingabe möglich, aber logisch sinnlos wäre SU$ = "M??*", was durch SU$ = "M*" ersetzt werden könnte.

Da in den Suchabfragen jedoch nicht immer '?' vorkommen, sollte man durch

```
10 FOR I = 1 TO N
20 IF MID$(SU$,I,1) = "?" THEN GOTO 'Zeichenweise
        abfragen'
30 NEXT
40 GOTO 'Gesamtvergleich'
```

vor der Such-Schleife die Rechenzeit optimieren, da das kurze Unterprogramm extrem weniger Rechenzeit verbraucht als das zeichenweise Vergleichen von 1000 Daten, wenn keine '?' vorliegen.

1.13 Speicherplatz einsparen

Nicht immer ist es sinnvoll Speicherplatz zu sparen, weil meist dadurch die Möglichkeit der Eigendokumatation von Programmen sehr stark eingeschränkt, wenn nicht sogar unmöglich gemacht, wird. Weiterhin sind Speicherplatzeinsparung und Rechenzeitverkürzung im allgemeinen konkurrierende Ziele, so daß ein gesunder Mittelweg gefunden werden muß.

Abgesehen von den logischen Möglichkeiten der Speicherplatzeinsparung (vgl. Datum platzsparend speichern), die immer nur anhand der Problemstellung abzuschätzen sind, sollen im folgenden einige allgemeine Tips gegeben werden:

Schreiben Sie mehrere Anweisungen in eine Zeile

Jede Zeile benötigt 5 Bytes zur internen Verwaltung:

2 Byte für die Zeilennummer
2 Byte für den Verweis auf die nächste Zeile im RAM
1 Byte für die Zeilenendemarkierung

Durch jede eingesparte Zeile gewinnen Sie 4 Byte.

Beispiel:

```
100 FOR I = 1 TO N
110 PRINT N
120 NEXT
```

oder

```
100 FOR I = 1 TO N:PRINT N:NEXT
```

sparen Sie 8 Byte: 2*5 für die Zeile - 2*1 für die ':'.

Löschen Sie alle unnötigen Leerzeichen

Jedes Leerzeichen - außer dem Leerzeichen zwischen Zeilennummer und erster Anweisung - muß gespeichert werden. Mit:

```
100 FORI=1TON:PRINTN:NEXT
```

sparen Sie also gegenüber dem letzten Beispiel weitere 5 Byte.

Löschen Sie alle REM-Befehle

Jeder REM-Befehl benötigt ein Byte + Anzahl der Zeichen hinter dem REM. Steht das REM in einer gesonderten Zeile sogar noch weitere 5 Byte mehr (s.o.).

Benutzen Sie Variablen statt Konstanten

Wenn Sie in einem Programm 10 mal den Wert 1.23456789 benötigen, setzen Sie einmal A=1.23456789 und benutzen Sie A statt 1.23456789. Jedes Zeichen einer Konstanten benötigt Speicherplatz. Auch Zahlkonstanten werden zeichenweise gespeichert.

Benutzen Sie temporäre Variablen häufiger

Kurzlebige Variablen (z.B. sofortiges Verarbeiten nach dem Einlesen) sollten häufiger in verschiedenen Programmteilen verwendet werden, da Basic keine lokalen Variablen kennt, und die Lebensdauer im Speicher für eine Variable von der Definition bis zum Programmende fortläuft. Durch die Wiederverwendung von Variablen sparen Sie Platz in der Variablenliste.

Häufige wiederkehrende Programmteile als Unterprogramm schreiben

Ein GOSUB XXXXX mit zugehörigem RETURN braucht viel weniger Speicherplatz als die mehrmalige Programmierung desselben Programmstückes.

Setzen Sie häufige benutzte Unterprogramme an den Anfang

Diese Unterprogramme können durch ein GOTO 'erste Zeile nach den Unterprogammen' sehr einfach übersprungen werden, aber das Voranstellen hat zwei Vorteile:

- Die Zeilenangabe nach einem GOSUB wird zeichenweise gespeichert, d.h. GOSUB3 benötigt 4 Byte weniger als GOSUB30000. Wird ein Unterprogramm sehr oft aufgerufen, macht sich diese Ersparnis jedesmal bemerkbar.

- Der Rechner sucht nach dem Unterprogramm intern ab der aufrufenden Zeile, wenn das aufgerufene Unterprogramm eine höhere Zeilennummer hat, ansonsten vom Programmanfang her. Wird das Unterprogramm von verschiedenen Stellen im Hauptprogramm aufgerufen, ist es günstiger, dies an den Programmanfang zu setzen. Dies trifft auch zu, wenn z.B. das Unterprogramm mit vielen GOTO und GOSUB versehen ist, die auf frühere Zeilennummern zeigen. Die Einsparung liegt bei großen Programmen dadurch nicht selten im Sekunden-Bereich.

Benutzen Sie die 0-Elemente von Feldern

In Basic kann zwar die Feldlänge mit DIM A(x) bestimmt werden, jedoch wird damit nur das letzte Element festgelegt. Die Felder beginnen immer mit dem Element '0'. Statt DIM A(100) - wenn 100 Elemente gewünscht sind - genügt also DIM A(99).

Wie leicht zu sehen ist, leidet durch die Speicherplatzeinsparung die Übersichtlichkeit der Programme sehr. Bevor Sie also zu obigen Hilfsmitteln greifen, sollten Sie andere Möglichkeiten (z.B. Modularisierung) in Betracht ziehen.

1.14 Rechenzeit verkürzen

Die Rechenzeit läßt sich am besten durch eine genaue Problemanalyse und entsprechende Programmierung verkürzen. Besonders wichtig sind dabei ineinandergeschachtelten FOR ...NEXT-Schleifen.

Optimieren Sie Anweisungen innerhalb von Schleifen

Hier genügt es meist, die innerste Schleife zu optimieren:

```
100 FOR I = 1 TO 10
110   FOR J = 1 TO 10
120      FOR K = 1 TO 10
130         FOR M = 1 TO 10
```

```
140          ' Anweisungen '
150         NEXT
160       NEXT
170    NEXT
180 NEXT
```

Jede im Teil 'Anweisungen' ersparte Millisekunde macht sich in der Rechenzeit mit 10 Sekunden bemerkbar. Beachten Sie auch, daß die Schleifen-Grenzen entsprechend der Problemstellung gewählt werden, überlegen Sie anhand des anstehenden Problemes ob z.B. M nicht nur von 2 bis 8 laufen muß. Bei einer 'Anweisungen'-Rechenzeit von 10 Sekunden werden dadurch insgesamt 3000 Sekunden = 50 Minuten eingespart. Durch ungeschickte Programmierung in diesen Punkten sind normale Rechenzeiten von Stunden sehr leicht auf Jahre zu verlängern und werden damit undurchführbar.

Vermeiden Sie unnötige Leerzeichen und REM-Befehle

Das Weglassen von unnötigen Leerzeichen und REM-Befehlen bewirkt auch eine Rechenzeitverkürzung, da der Rechner nicht unnötige Zeichen zu überlesen braucht.

Benutzen Sie Variablen statt Konstanten

Auch die Benutzung von Variablen statt Konstanten spart Rechenzeit, da eine Konvertierung der Daten entfällt (Faktor 10).

Definieren Sie häufig benutzte Variablen am Programmanfang

Häufig benutzte Variablen sollten am Programmanfang definiert werden (z.B. A=0), da Sie dann beim Suchen in der Variablentabelle schneller gefunden werden.

Geben Sie bei NEXT die Laufvariable nicht an

Wenn Sie NEXT ohne Index-Variable (also nicht NEXT I) benutzen, entfällt die Überprüfung durch das Betriebssystem.

Zu den oben genannten Mitteln sollten Sie allerdings nur greifen, wenn das Einsparen von Rechenzeit wirklich sehr wichtig ist, da Ihnen viele Möglichkeiten der Programmdokumentation genommen werden.

1.15 Index-Funktion

```
100 IF LEN(AA$) LT LEN(A$) THEN I=0 : RETURN
110 FOR I=1 TO LEN(AA$)-LEN(A$)+1
120 IF MID$(AA$,I,LEN(A$)) NE A$ THEN NEXT I
130 IF I GT LEN(AA$)-LEN(A$)+1 THEN I=0
140 RETURN
```

Die Index-Funktion ist ein in anderen Programmiersprachen häufig fest integrierter Bestandteil, der feststellt, ob in einer vorgegebenen Zeichenreihe eine andere (kleinere) Zeichenreihe eingebettet ist. Da das Commodore-Basic uns diesen Befehl nicht zur Verfügung stellt, muß ein kleines Unterprogramm geschrieben werden, das uns die gleiche Leistung bringt. Die Zeichenreihe, in der gesucht werden soll, muß an das Unterprogramm in der Variablen AA$ übergeben werden und die Zeichenreihe, nach der gesucht werden soll in der Variablen A$ abgelegt sein. Ausgabeparameter für die Position des ersten übereinstimmenden Zeichens beim erstenmal, wo Übereinstimmung erzielt wurde, ist die Variable I.

In Zeile 100 wird zunächst festgestellt, ob AA$ überhaupt in A$ von der Länge her enthalten sein kann. Dann wird die Teilzeichenreihe in der großen Zeichenreihe jeweils um ein Zeichen nach rechts geschoben und überprüft, ob Übereinstimmung besteht. Wäre die erste Position soweit in der zu suchenden Zeichenreihe hinten, daß nicht mehr die gesamte zu suchende Zeichenreihe enthalten sein kann, so wird I wiederum auf 0 gesetzt und das Unterprogramm verlassen.

1.16 Zahlkonvertierungen

Wer etwas tiefer in seinen Rechner einsteigt, hat nicht nur mit Dezimalzahlen zu tun, sondern auch mit Hexadezimalzahlen, Binärzahlen und manchmal auch mit Oktalzahlen. In diesem Kapitel wollen wir einige kurze Programme vorstellen, die jeweils eingegebene Zahlen in ein anderes Format umrechnen. Für die Verwendung als Basic-Erweiterung und ihre Realisierung in Maschinensprache sei auf Kapitel 2.2.2 verwiesen.

Binär - Dezimal

```
100 REM *** ZAHLKONVERTIERUNG BINAER-DEZIMAL          <119>
110 A$="10010110"                                     <077>
120 FA=1                                              <226>
130 FOR I=LEN(A$) TO 1 STEP -1                        <209>
140 A = A + FA*VAL(MID$(A$,I,1))                      <005>
150 B = B + 2^(LEN(A$)-I)*VAL(MID$(A$,I,1))           <040>
160 FA=FA*2                                           <062>
170 NEXT                                              <044>
180 PRINT A$;" = ";A;" = ";B                          <035>
190 END                                               <062>
```

Binärzahlen liegen in der Regel als Zeichenreihe - bestehend aus den Zeichen '1' und '0' vor. In dem Programm wurden gleichzeitig zwei Wege beschritten. In jedem Fall muß aber mit der Zählweise von hinten begonnen werden. Da die einzelnen 'Ziffern' jeweils Potenzen der Basis darstellen, ist eine Umrechnung recht einfach. In der Version A wird ein Faktor (FA) jeweils mit 2 multipliziert, was den eigentlichen Ausdruck zur Umrechnung sehr vereinfacht. Wie im Fall B wird direkt die Zweierpotenz verwendet, bei der Exponent sich aus der Position - von hinten - in der Zeichenreihe ergibt. In jedem Fall wird jedoch mit dem Wert des Zeichens multipliziert, d.h. der Faktor wird nur zu dem bereits vorhandenen Wert hinzuaddiert, wenn ein '1' in der Zeichenreihe steht.

Dezimal - Binär (bis 32767)

```
200 REM *** ZAHLKONVERTIERUNG DEZIMAL-BINAER 8-STELL
    IG (BIS 255)                                      <031>
210 A=230                                             <090>
220 FOR I=7 TO 0 STEP-1                               <232>
230 IF A AND 2^I THEN A$=A$+"1":GOTO 250              <038>
240 A$=A$+"0"                                         <138>
250 PRINT 2^I,A$                                      <077>
255 NEXT                                              <129>
260 PRINT A;" = ";A$                                  <059>
270 END                                               <143>
```

Sofern nur kleinere Dezimalzahlen (bis 32767) in eine binäre Zeichenreihe umzuwandeln sind, kann man sich der UND-Verknüpfung bedienen, da diese jeweils 15-Bit miteinander vergleicht. Im Prinzip hilft uns hier die interne Binärdarstellung einer Zahl im Computer. Im Prinzip wird ein einzelnes Bit von der Bit-Position 14 bis zur Position

0 (von vorne nach hinten) durchgeschoben. Ist in der ursprünglichen Zahl dieses Bit gesetzt, so wird an die Zeichenreihe A$ eine '1' angehangen, im anderen Fall ein '0'.

Dezimal - Binär (beliebig groß)

```
300 REM *** ZAHLKONVERTIERUNG DEZIMAL-BINAER BELIEBI
    G GROSS                                          <001>
310 A=1048000 : B=A                                  <246>
320 FOR I=31 TO 0 STEP-1                             <122>
330 IF A >= 2^I THEN A$=A$+"1" : A=A-2^I : GOTO 350  <130>
340 A$=A$+"0"                                        <239>
350 NEXT                                             <225>
360 PRINT A$;" = ":PRINT B                           <056>
370 END                                              <243>
```

Beliebig groß in diesem Falle heißt bis zur größten, im Commodore 64, darstellbaren Zahl (2**31). Auch in diesem Falle wird das Bitmuster von links nach rechts erstellt, allerdings muß jetzt jeweils abgefragt werden, ob ein Wert größer als die entsprechende Zweierpotenz vorliegt. Wenn ja, wird eine '1' an den String A$ angehangen und die entsprechende Zweierpotenz vom ursprünglichen Wert subtrahiert. Im Gegensatz zu dem o.a. Verfahren wird dabei der ursprüngliche Wert in A zerstört. Soll er weiterhin bestehen bleiben, so ist er zunächst an eine entsprechende Hilfsvariable zu übergeben.

Hilfsfunktionen für Hexadezimal-Umwandlungen

```
400 REM *** HILFSFUNKTIONEN UND AUFRUF               <197>
405 HE$="0123456789ABCDEF"                           <223>
410 H=210                                            <040>
415 GOSUB 500                                        <194>
420 PRINT"{CLR,RVSON,ORANGE}ZWEISTELLIGE HEX-ZAHL BI
    LDEN{BLUE}                                       <044>
425 PRINT : PRINT"DEZ.";H,"= HEX. ";H$               <085>
430 GOSUB 600                                        <210>
435 PRINT"{3DOWN,RVSON,ORANGE}VIERSTELLIGE HEX-ZAHL
    BILDEN{BLUE}                                     <210>
440 PRINT : PRINT"DEZ.";HH,"= HEX. ";HH$             <244>
445 DEF FN V(X)=X-48+7*(X>64)                        <106>
450 H$="D2"                                          <155>
455 GOSUB 700                                        <236>
460 PRINT"{3DOWN,RVSON,ORANGE}DEZ. AUS ZWEISTELLIGER
     HEXZAHL{BLUE}                                   <101>
465 PRINT : PRINT"HEX. ";H$,"= DEZ.";H               <125>
```

```
470 HH$="6590"                                                    <085>
475 GOSUB 800                                                    <001>
480 PRINT"{3DOWN,RVSON,ORANGE}DEZ. AUS VIERSTELLIGER
     HEXZAHL{BLUE}                                               <112>
485 PRINT : PRINT"HEX. ";HH$,"= DEZ.";HH                         <033>
```

Wichtig ist die Definition der Zeichenreihe HE$ in Zeile 405, die alle hexadezimalen Ziffern in aufsteigender Reihenfolge (als Zeichen) enthält, und die Definition der Funktion FN(V) in Zeile 445.

Diese Funktion bedarf sicherlich einer näheren Erklärung. Betrachten wir uns zunächst Teile der Funktion, unter der Berücksichtigung, daß nur korrekte hexadezimale Werte (siehe HE$) vorliegen. Die Zeichen '0', '1',...,'9' haben einen ASCII-Code in vorbezeichneter Reihenfolge, beginnend bei 48. Der ASCII-Code der Buchstaben beginnt bei 65. Liegt also das Zeichen für eine Ziffer zwischen '0' und '9' vor, so liefert der Teil 'X-48' der Funktion den dezimalen Wert des Zeichens. Liegt ein Buchstabe zwischen 'A' und 'F' vor, so liefert der Teilausdruck '(X GT 64)' den Wert '-1'. Durch das negative Vorzeichen und den Faktor 7 werden also im Falle eines Buchstabens weitere sieben Einheiten vom Wert des ASCII-Codes abgezogen. Zusammen mit dem ersten Teilausdruck liefern somit die Buchstaben einen Wert zwischen 10 und 15.

Dezimal - Zweistellig Hexadezimal

```
500 REM *** ZWEISTELLIGE HEXZAHL                                 <174>
510 H$=MID$(HE$,H/16+1,1) + MID$(HE$,(H AND 15)+1,1)             <049>
520 RETURN                                                       <152>
```

Liegt ein dezimaler Wert zwischen 0 und 255 vor, so kann eine zweistellige Hexadezimalzahl gebildet werden. Das erste Zeichen gibt die ganzzahligen Vielfachen von 16 an und das zweite Zeichen den Rest. Da Hexadezimalzahlen auch als Zeichenreihen dargestellt werden müssen, werden die entsprechenden Ziffern anhand der Position in der Zeichenreihe HE$ festgestellt.

Dezimal - Vierstellig Hexadezimal

```
600 REM *** VIERSTELLIGE HEXZAHL                                 <010>
610 HH=26000                                                     <158>
620 H=INT(HH/256)                                                <072>
```

```
630 GOSUB 500                                   <154>
640 HH$=H$                                      <084>
650 H=HH-256*H                                  <082>
660 GOSUB 500                                   <184>
670 HH$=HH$+H$                                  <208>
680 RETURN                                      <056>
```

Das Bilden einer vierstelligen Hexadezimalzahl kann aufgespalten werden in: zwei mal eine zweistellige Hexadezimalzahl bilden. Dabei geben die ersten beiden Ziffern ganzzahlige Vielfache von 256 an (die erste Ziffer entsprechen dann ganzzahlige Vielfache von 4096) und die letzten beiden Ziffern - analog zum letzten Programm - den Rest. In Zeile 620 wird die erste Division ausgeführt und eine zweistellige Hexadezimalzahl gebildet, die der Variablen HH$ übergeben wird. In Zeile 56 werden dann die ganzzahligen Vielfachen von 256 vom ursprünglichen Wert subtrahiert und für den Rest eine zweistellige Hexadezimalzahl gebildet, die mit dem ersten Teil dann verknüpft wird.

Zweistellig Hexadezimal - Dezimal

```
700 REM *** DEZ. AUS 2-STELLIGER HEXZAHL        <227>
710 H=16*FN V(ASC(H$))+FN V(ASC(MID$(H$,2)))    <150>
720 RETURN                                      <096>
```

Aufgrund der oben vorgeführten Funktion, die die entsprechenden ASCII-Codes für die hexadezimalen Ziffern in Dezimal umrechnet, läßt sich die vorliegende Umrechnung sehr einfach gestalten. Für beiden Zeichen wird der Dezimalwert aufgrund dieser Funktion errechnet, wobei der erste Wert noch mit 16 multipliziert wird.

Vierstellig Hexadezimal - Dezimal

```
800 REM *** DEZ. AUS 4-STELLIGER HEXZAHL        <042>
810 H$=LEFT$(HH$,2)                             <118>
820 GOSUB 700                                   <091>
830 HH=256*H                                    <020>
840 H$=RIGHT$(HH$,2)                            <149>
850 GOSUB 700                                   <121>
860 HH=HH+H                                     <035>
870 RETURN                                      <247>
```

Analog der Umrechnung in die andere Richtung kann auch das Errechnen einer Dezimalstelle aus einer vierstelligen Hexadezimalzahl in zwei gleichen Bereichen erfolgen. Dazu wird der Einfachheit halber das zuletzt beschriebene Unterprogramm aufgerufen, was eine Speicherplatzersparnis bringt. Die Auswertung des ersten Teils muß dann allerdings noch mit 256 multipliziert werden.

Oktal - Dezimal

```
900 REM *** ZAHLKONVERTIERUNG OKTAL-DEZIMAL          <100>
910 A$="3776700"                                     <090>
920 FOR I=LEN(A$) TO 1 STEP -1                       <234>
930 A = A + 8↑(LEN(A$)-I)*VAL(MID$(A$,I,1))          <059>
940 NEXT                                             <049>
950 PRINT"{CLR}";A$;" = ";A                          <001>
960 END                                              <067>
1000 REM *** ZAHLKONVERTIERUNG DEZIMAL-OKTAL         <200>
```

Die Umwandlung einer Oktalzahl in eine Dezimalzahl kann analog der Konvertierung einer Binärzahl (Version B) erfolgen. Es braucht lediglich die Basis in '8' umgewandelt zu werden (Zeile 930).

Dezimal - Oktal

```
1000 REM *** ZAHLKONVERTIERUNG DEZIMAL-OKTAL         <200>
1010 A=1048000 : B=A : A$=""                         <073>
1020 FOR I=10 TO 0 STEP-1                            <053>
1030 A$=A$+RIGHT$(STR$(INT(A/8^I)),1)                <223>
1040 IF A>= 8^I THEN A=A-8^I * INT(A/8^I)            <246>
1050 NEXT                                            <160>
1060 PRINT" ";A$;" = ";B                             <222>
1070 END                                             <178>
```

Auch die Umwandlung von Dezimal nach Oktal erfolgt analog der Konvertierung von Dezimal nach Binär. Da bei Binärzahlen jedoch außer der '0' nur eine Ziffer vorhanden ist, müssen die unterschiedlichen Ziffern bei der Umrechnung nach Oktal noch berücksichtigt werden. Abgezogen wird also

in Zeile 1040 nicht nur die direkte Potenz von 8, sondern hier mit der entsprechenden Ziffer multipliziert.

Daß in Zeile 1030 nur der rechte Teil der in eine Zeichenreihe umgewandelten Ziffer herangezogen wird, liegt an dem Vorzeichen. Würde dies nicht in dieser Weise durchgeführt, ergäbe sich die Oktaldarstellung der Zahl mit Zwischenräumen.

2

Utilities

2. Utilities

Nachdem wir im ersten Kapitel allgemeine Routinen vorgestellt haben, die sich auch leicht auf andere Rechner übertragen lassen, wollen wir im zweiten Kapitel spezielle Routinen für den Commodore 64 (sie sind teilweise auch für Rechner anderer Commodore-Serien geeignet) vorstellen. Das Kapitel gliedert sich in die beiden großen Bereiche Basic-Utilities und Maschinenroutinen, wobei einige Themen auch in beiden Abschnitten behandelt werden. Wir haben versucht, Ihnen mit diesen Routinen die wichtigsten Hilfsmittel in die Hand zu geben, die Sie bei Ihrer Arbeit mit dem Rechner benötigen. Bitte haben Sie Verständnis dafür, wenn wir die eine oder andere Routine, die Sie gerade benötigen, nicht mit abgebildet haben. Wir waren bestrebt, Routinen auszuwählen, die von allgemein gültigem Interesse sind.

2.1 Basic-Utilities

In diesem Kapitel sind einige Unterprogramme zusammengefaßt, die zwar speziell auf den Commodore 64 abgestimmt sind, aber von allgemein gültiger Bedeutung sind. So wird die Joystick-Handhabung (und teilweise auch die Steuerung mittels Paddles) sicherlich bei sehr vielen Spielen verwendet, aber auch andere Anwendungsmöglichkeiten sind denkbar.

Für benutzerfreundliche Programme ist eine Cursor-Positionierung unerläßlich, und wenn die Programme einen größeren Umfang annehmen und eine Teilung erfolgen muß, ist es auch sehr hilfreich zu wissen, wie man Daten von einem Programm an das andere übergibt.

Etwas spezieller sind schon die Kapitel ab 2.1.5. Wer eine Maschinenroutine nicht immer getrennt vom Basic-Programm laden möchte, kann diese als DATA-Zeilen in sein Basic-Programm übernehmen, und wenn die Floppy-Lampe flackert ist es anzuraten, bei der Floppy nachzufragen, was ihr denn Unbehagen bereitet.

Die letzten Unterkapitel beschäftigen sich mit Themen, die zwar nicht in jedem Programm gebraucht werden, aber von Zeit zu Zeit doch eine nützliche Unterstützung darstellen.

2.1.1 Joystick-Steuerung

Dieses Kapitel ist einem Peripheriegerät gewidmet, das hauptsächlich für Spiele herangezogen wird. Aber nicht nur hier liegt ein Einsatzgebiet des Joysticks, sondern auch bei 'normalen' Anwendungen. In gewisser Weise kann ein Joystick die mittlerweile vielgerühmte 'Maus' ersetzen. Gehen wir zunächst auf die Abfrage des Joysticks und dessen Verwendung bei hochauflösender Grafik ein.

Joystick-Abfrage

In der Regel wird für den Joystick Control-Port 2 herangezogen. Beide Control-Ports sind mit Vorsicht zu genießen, da ihre Abfrage teilweise mit der Tastaturbehandlung zusammenfällt. Dieser Umstand tritt bei Control-Port 1 stärker auf, so daß auch hier der offensichtliche Grund für die hauptsächliche Verwendung von Control-Port 2 liegt. Für jeden Control-Port ist ein Register vorgesehen. Für Control-Port 2 ist dies die Speicherzelle 56320 oder in hexadezimaler Schreibweise ausgedrückt $DC00. Für den Control-Port 1 ist es gleich die nächste Speicherzelle (56321, $DC01).

Abgefragt werden kann dies mit

```
100 JO = PEEK(56320)      : REM Control-Port 2
110 PRINT JO
120 GOTO 100
```

bzw.

```
100 JO = PEEK(56321)      : REM Control-Port 1
110 PRINT JO
120 GOTO 100
```

Damit bringen Sie die dezimalen Werte des jeweiligen Registerzustandes auf den Bildschirm. Von vornherein wollen wir die Handhabung so gestalten, daß zunächst der Wert des

Registers in eine Variable (die wir durchgehend mit JO bezeichnen werden) übernehmen. Bei größeren Programmen kann dann mit der Variablen weitergearbeitet werden, was einerseits Speicherplatz im Programm spart (gegenüber vielen PEEK-Abfragen), andererseits können keine Fehler im Programmablauf auftreten, wenn sich der Wert im Register während des Programmablaufes ändert.

Wenn Sie die Dezimalwerte für Control-Port 1 und 2 jeweils mit der Richtung des Steuerknüppels (mit oder ohne 'Feuertaste') notieren, so ergibt sich die Belegung, wie Sie in den Abbildungen 2.1.1-1 und 2.1.1-2 dargestellt ist.

Die Kollision mit Eingaben von der Tastatur wird hier schon deutlich, wenn Sie an Control-Port 1 den Joystick angeschlossen haben und den Steuerknüppel nach links bewegen. Es tritt eine Verzögerung der Ausgabe ein, die mit der Verlangsamung durch die CTRL-Taste übereinstimmt.

Rechnet man die Dezimalzahlen in Binärzahlen um, so sieht man, daß den verschiedenen Richtungen und dem Feuerknopf jeweils ein Bit im Register zugeordnet ist. Mit folgendem Programm holen Sie die Bitmuster auf den Bildschirm:

```
100 JO=PEEK(56320)                                    <194>
110 A$="0":A=JO                                       <191>
120 GOSUB 200                                         <151>
130 PRINT JO,A$                                       <069>
140 GOTO 100                                          <166>
200 FOR I=6 TO 0 STEP-1                               <211>
210 IF A AND 2^I THEN A$=A$+"1":GOTO 230              <016>
220 A$=A$+"0"                                         <118>
230 NEXT                                              <104>
240 RETURN                                            <126>
```

Das Programm ist für Control-Port 2 ausgelegt. Zunächst wird auch hier an die Variable JO wieder der aktuelle Wert des Registers übergeben. In der Variablen A$ wollen wir das Bitmuster des Registers zusammensetzen. Da die Werte am Control-Port 2 immer kleiner als 128 sind, kann das erste Bit im Register nie '1' werden, und wir machen eine Vorbesetzung mit '0' in Zeile 110. Der Rest des Programmes entspricht der Zahlkonvertierung aus Kapitel 1.16.

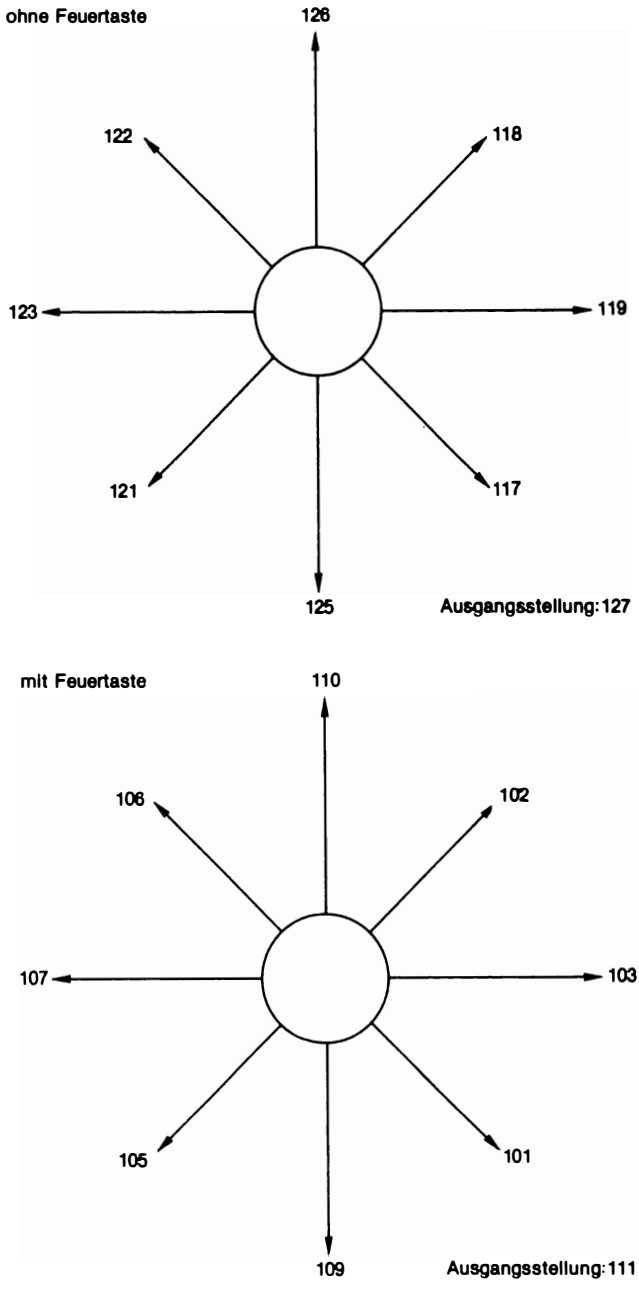


Bild 2.1.1-1 : Joystickwerte an Control-Port 2

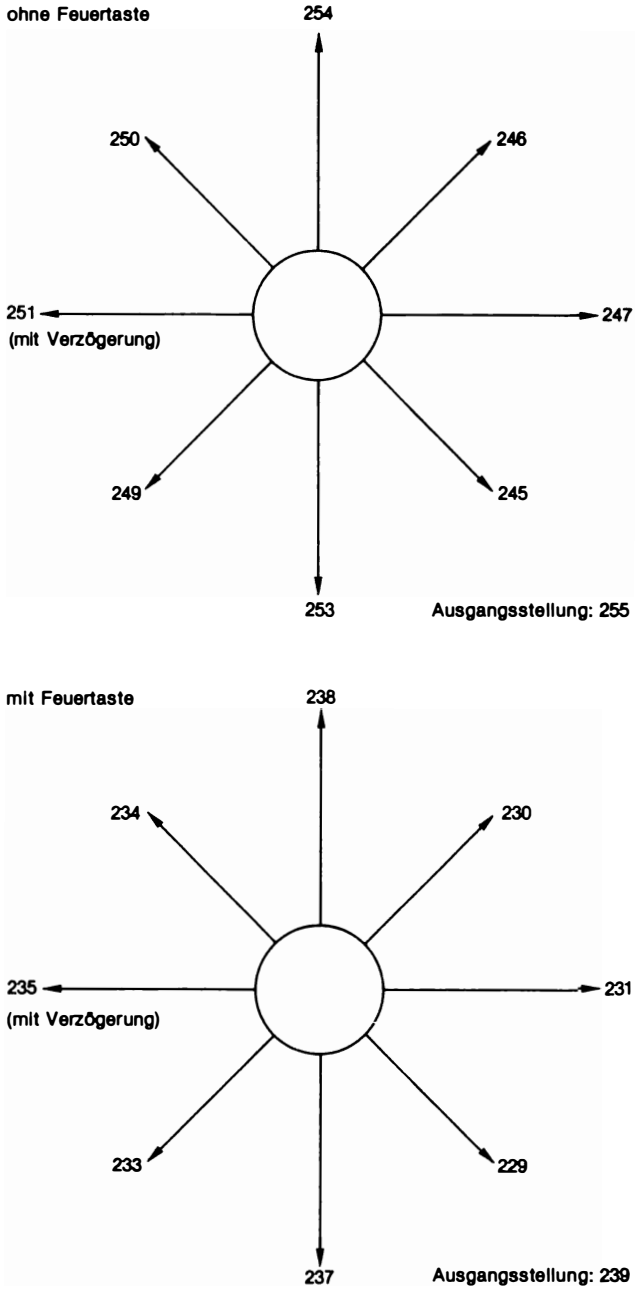


Bild 2.1.1-2 : Joystickwerte an Control-Port 1

Grundprogramm

Für den Control-Port 2 wollen wir ein kurzes Programm vorstellen, mit dem Sie den Joystick als Zeichenstift am Bildschirm verwenden können. Damit wird auch die programmäßige Verwertung der Registerwerte deutlich. Zunächst das Listing:

```
100 POKE 53280,7                                                    <096>
110 POKE 53281,7                                                    <107>
120 PRINT"{BLUE}                                                    <082>
130 GREIN6,7                                                        <144>
135 PX=160                                                          <120>
136 PY=100                                                          <116>
140 JO=PEEK(56320)                                                  <234>
145 REM ----- OBEN RECHTS  -----                                    <207>
150 IF (JO AND 9)=0 THEN PY=PY-1 : PX=PX+1 : GOTO 230               <203>
155 REM ----- UNTEN RECHTS -----                                    <063>
160 IF (JO AND 10)=0 THEN PY=PY+1 : PX=PX+1 : GOTO 230              <252>
165 REM ----- UNTEN LINKS  -----                                    <001>
170 IF (JO AND 6)=0 THEN PY=PY+1 : PX=PX-1 : GOTO 230               <220>
175 REM ----- OBEN LINKS   -----                                    <165>
180 IF (JO AND 5)=0 THEN PY=PY-1 : PX=PX-1 : GOTO 230               <230>
185 REM ----- OBEN         -----                                    <046>
190 IF (JO AND 1)=0 THEN PY=PY-1 : GOTO 230                         <212>
195 REM ----- UNTEN        -----                                    <158>
200 IF (JO AND 2)=0 THEN PY=PY+1 : GOTO 230                         <222>
205 REM ----- LINKS        -----                                    <159>
210 IF (JO AND 4)=0 THEN PX=PX-1 : GOTO 230                         <233>
215 REM ----- RECHTS       -----                                    <241>
220 IF (JO AND 8)=0 THEN PX=PX+1 : GOTO 230                         <246>
225 REM ----- FEUERKNOPF   -----                                    <039>
230 IF (JO AND 16)=0 THEN PF=1 : GOTO 250                           <205>
240 PF=0                                                            <104>
250 PUNKTPF,PX,PY                                                   <203>
260 GOTO 140                                                        <035>
```

Zunächst werden für die normale Arbeit Bildschirmhintergrund- und Rahmenfarbe gesetzt und die Zeichenfarbe auf blau geschaltet. Dann wird mit dem GREIN-Befehl (siehe Kapitel 2.2.13) die Grafik eingeschaltet (gelber Hintergrund / blaue Farbe der Zeichnung). Für die jeweiligen X- und Y-Koordinaten wollen wir die Variablen PX und PY (Punkt X-Koordinate / Punkt Y-Koordinate heranziehen. Diese werden in den Zeilen 135 und 136 auf den Bildschirmmittelpunkt gesetzt (auch der Befehl PUNKT wird in Kapitel 2.2.13.3 besprochen).

In Zeile 140 wird das Register für Control-Port 2 abgefragt. Anschließend befindet sich in den Zeilen ab 150 die Abfrage, in welcher Richtung der Cursor bewegt wurde. Wie

wir in Kapitel 1.1 gesehen haben, ist für die vier normalen Richtungen (oben, unten, rechts, links) jeweils ein Bit zuständig, mit folgender Verteilung:

1 - oben
2 - unten
4 - links
8 - rechts
16 - Feuerknopf

Die Diagonalen ergeben sich aus der Kombination von den beiden daran beteiligten Richtungen mit:

5 - oben links
6 - unten links
9 - oben rechts
10 - unten rechts

Zunächst müssen aus logischen Gründen die Diagonalen abgefragt werden. Würde man erst die normalen Richtungen abfragen, so würde die Abfrage der Diagonalen nicht mehr behandelt. Wählen Sie als Beispiel das Bitmuster für '9'. In diesem Falle ist das Bit für '1' gesetzt und die Behandlung des Falles 'Joystick nach oben' würde durchgeführt, ohne daß noch die Behandlung der Fälle 'oben rechts' oder 'oben links' erfolgte. Aus Rechenzeitgründen wird jeweils die Abfrage des Feuerknopfes übergangen. Dadurch erspart man sich zu viele Abfragen.

In jedem Falle werden die entsprechenden PX- und PY-Werte aktualisiert. Die Schrittweite für die Bewegung wird generell auf einen Bildpunkt festgelegt.

In Zeile 230 wird erfragt, ob der Feuerknopf gedrückt ist. In diesem Fall wird die Variable PF auf '1' gesetzt, was angezeigen soll, daß hier ein Punkt gesetzt werden soll. Im anderen Fall wird in Zeile 240 die Variable PF auf '0' zurückgesetzt.

Dieses Grundprogramm können Sie bestimmt auch noch für andere Anwendungen heranziehen. Versuchen Sie es doch einmal, indem Sie Sprite-Koordinaten variieren.

2.1.2 Paddle-Steuerung

Ähnlich wie beim Joystick kann die Abfrage für Paddles geschaltet werden. Wir wollen hier zunächst nur den Fall der

Paddles am Control-Port 1 behandeln. Beachten sie bitte, daß jeweils zwei Paddles an einem Control-Port angeschlossen werden. Wie weiter unten beschrieben, kann wegen der analogen Schaltung keine gleichzeitige Abfrage von vier Paddles erfolgen. Auf die Möglichkeit des Umschaltens gehen wir zum Schluß ein.

Wenn Sie Ihre Paddles am Control-Port 1 anschließen und das Register 56321 abfragen, so werden Sie feststellen, daß das Register nur auf einen Tastendruck an einem der beiden Paddles reagiert. Der übliche Wert von 255 wird auf 251 bzw. 247 verringert. 243 ist dann logischerweise der Wert, wenn beide Knöpfe gleichzeitig gedrückt sind. Wie man leicht daraus ersieht, sind also Bit 2 und Bit 3 für die Abfrage der Knöpfe an den Paddles zuständig.

Die Potentiometerwerte der Paddles können über dieses Register nicht abgefragt werden. Dazu sind die Register 25 und 26 des SID zuständig. Die entsprechenden Speicheradressen sind 54297 und 54298.

Da die Paddles einen analogen Wert an den Rechner abgeben, müssen diese zuerst über einen Analog-/ Digitalwandler umgewandelt werden. Dies geschieht bei den genannten Registern des SID. Wen es interessiert: ein Kondensator wird über die Paddles entladen, und der Rechner stellt die Zeit fest, die der Kondensator für das vollständige Entladen benötigt. Daß nicht die genauesten Werte zurückgegeben werden, können Sie schon daran sehen, wenn Sie das unten aufgeführte kurze Testprogramm laufen lassen, ohne die Paddles zu bewegen. In der Regel schwanken die ausgegebenen Werte um zwei Einheiten, aber auch größere Differenzen sind möglich.

Analog zu dem Programm bei der Joystick-Abfrage wollen wir uns ein kurzes Programm schreiben, mit dem die drei Register abgefragt werden können:

```
100 PT = PEEK(56320)
110 P1 = PEEK(54297)
120 P2 = PEEK(54298)
130 PRINT PT,P1,P2
140 GOTO 100
```

Umschaltung zwischen zwei Paddle-Paaren

Da für die analoge Steuerung nur zwei Register zur Verfügung stehen, aber bei Verwendung von zwei Paddle-Paaren vier gebraucht werden, muß man jeweils umschalten. In der

Regel wird man ein Maschinenprogramm dazu verwenden, aber auch Basic bietet diese Möglichkeit:

```
200 POKE 56333,1                                                    <192>
210 POKE 56320,64:POKE 56322,192                                    <154>
220 PRINT PEEK(54297);PEEK(54298),                                  <025>
230 PRINT"{WHITE}";PEEK(56320)AND 12;"{YELLOW}";                    <128>
240 POKE 56320,128                                                  <078>
250 PRINT'PEEK(54297);PEEK(54298)                                   <011>
260 PRINT"{WHITE}";PEEK(56321)AND 12;"{YELLOW}";                    <160>
270 POKE 56333,129                                                  <114>
280 GOTO 200                                                        <052>
```

Bei Verwendung der in Kapitel 2.2.2 vorgestellten Zahlkonvertierungen ergibt sich folgender Ablauf:

```
100 POKE$DC0D,1                                                     <119>
110 POKE$DC00,$40:POKE$DC02,$C0                                     <101>
120 PRINT PEEK($D419);PEEK($D41A),                                  <178>
130 PRINT"{WHITE}";PEEK($DC00)AND 12;"{YELLOW}";                    <039>
140 POKE$DC00,$80:POKE$DC02,$C0                                     <135>
150 PRINT PEEK($D419);PEEK($D41A)                                   <164>
160 PRINT"{WHITE}";PEEK($DC01)AND 12;"{YELLOW}";                    <070>
170 POKE$DC0D,129                                                   <040>
180 GOTO 100                                                        <206>
```

In Zeile 100 (200) wird der Interrupt ausgeschaltet, indem im CIA-ICR-Register das Bit 'Unterlauf Timer A' gelöscht wird. Dadurch wird die einzige standardmäßige Interruptquelle gesperrt. Notwendig ist dies, da ein Interrupt die Tastatur ansprechen würde, was ebenfalls über das Register 56320 geschieht.

In den Zeilen 110 und 140 (210 und 240) geschieht die eigentliche Umschaltung, in den Zeilen 120 und 150 (220 und 250) erfolgt die Abfrage und abschließend in den Zeilen 130 und 160 (230 und 260) die Kontrollausgabe auf dem Bildschirm. Natürlich muß der Interrupt wieder eingeschaltet werden (Zeilen 170 bzw. 270).

2.1.3 Cursor-Positionierung

Ein immer wieder vorkommendes Problem ist die richtige Positionierung des Cursors am Bildschirm. Wir wollen zunächst eine 'normale' Programmversion vorstellen und anschließend eine Cursor-Positionierung mittels POKE-Befehlen behandeln.

Wie positioniert man den Cursor in eine gewünschte Zeile und Spalte, wenn man nicht weiß, wo er sich befindet? Dazu gibt man zunächst den Befehl 'HOME' für den Cursor, um eine definierte Ausgangsstellung zu erreichen, und anschließend die gewünschte Anzahl von 'Cursor rechts' und 'Cursor nach unten' ein, wobei zu berücksichtigen ist, daß die PRINT-Befehle alle mit einem ';' abgeschlossen werden, um auch die erreichte Position für die weitere Ausgabe zu erhalten.

Besetzen der Variablen am Programmanfang:

```
100 CH$ = 'Cursor HOME'
110 CR$ = 'Cursor rechts'
120 CU$ = 'Cursor unten'
130 FOR I = 1 TO 6
140     CR$ = CR$ + CR$
150     CU$ = CU$ + CU$
160 NEXT
170 CR$ = LEFT$(CR$,40)
180 CU$ = LEFT$(CU$,25)
```

Das eigentliche Unterprogramm wird mit den Variablen 'Z' für die Zeile und 'S' für die Spalten aufgerufen:

```
1000 PRINT CH$ ; LEFT$(CR$,S) ; LEFT$(CU$,Z);:RETURN
```

Zählt man die Zeilen und Spalten - wie das Betriebssystem auch - von '0' ab, gibt aber am Bildschirm abgezählte Zeilen und Spalten ein, so lautet die Zeile:

```
1000 PRINT CH$ ; LEFT$(CR$,S-1); LEFT(CU$,Z-1);:RETURN
```

Da die Cursor-Position auch in der Zero-Page abgelegt ist,

kann sie dort mittels zweier POKE-Befehle geändert werden:

POKE 214,(Zeile) : POKE 211,(Spalte)

Damit ist aber noch keine Positionierung erfolgt. Dies wird mit

SYS 58732

veranlaßt.

2.1.4 Parameterübergabe zwischen Programmen

Haben Sie sehr große Basic-Programme, so ist es manchmal notwendig, vom Programm aus ein weiteres Basic-Programm (ein sogenanntes Overlay) nachzuladen. Dieses Verfahren ist auch sinnvoll, wenn Sie mit Menüs arbeiten.

Das direkte Nachladen mit LOAD wird im ersten Unterkapitel mit seinen Vor- und Nachteilen beschrieben. Danach beschreiben wir einen sogenannten Kaltstart, wobei jedoch die Variablen gelöscht werden. Wie Sie dennoch die Parameter zwischen den Programmen übergeben können, wird in den folgenden drei Kapiteln erläutert.

2.1.4.1 Parameterübergabe durch Overlay

Ein LOAD-Befehl, der im Programm aufgerufen wird, arbeitet ähnlich wie im Direktmodus, jedoch startet er das Programm nach dem Ladevorgang sofort mit der ersten Zeile. Die Variablentabelle und die Strings bleiben dabei im allgemeinen erhalten, und auch Floppy-Dateien bleiben offen.

Zu beachten ist allerdings, daß ein LOAD-Befehl im Programm keine Basic-Zeiger verändert. Insbesondere gilt dies für den Zeiger auf den Beginn der Variablen, so daß die Variablen durch das Overlay überschrieben werden, wenn dieses größer ist als das aufrufende Programm.

Wir merken uns also: das erste Programm einer Overlay-Sequenz muß das größte sein. Insbesondere ein Menüprogramm, das nur als Verteiler arbeitet, erfüllt i.a. diese Voraussetzung nicht.

Eine weitere unangenehme Eigenschaft der Overlays ist, daß String-Variablen, die auf Strings im Basic-Text zeigen, durch Overlay zerstört werden. Davon betroffen sind die Variablen im folgenden Programmstück:

```
10 A$="GHIJK"
20 B$=A$
30 READC$
40 DATA HALLO
```

Nicht betroffen sind dagegen die Variablen im folgenden Programmstück, weil durch Stringverknüpfungen der Basic-Interpreter gezwungen wird, die Stringinhalte in den Stringbereich zu legen.

```
10 A$="GHIJK"+""
20 B$=A$
30 INPUT A$
40 READC$:C$=C$+""
```

Ein ähnliches Problem wie bei den Strings ergibt sich bei den Funktionen, die mit DEFFN() definiert werden. Diese können im Overlay nur dann verwendet werden, wenn sie dort nochmals definiert wurden.

2.1.4.2 Kaltstart

Kaltstart soll hier bedeuten, daß sämtliche Variablen des aufrufenden Programms gelöscht werden und somit das neue Programm ein Zustand vorfindet, als ob es mit LOAD und RUN gestartet worden wäre. Diesen Kaltstart können Sie auf zwei Arten erzeugen: Zum einen können Sie den LOAD- und den RUN-Befehl in den Tastaturpuffer legen und das aufrufende Programm beenden, so daß die entsprechenden Zeichen am Bildschirm erscheinen und durch die RETURN's, die Sie ebenfalls im Tastaturpuffer ablegen, müssen ausgeführt werden (vgl. auch Kapitel 2.1.4.3).

Eleganter ist jedoch diese Methode: Zu Beginn jedes Programms setzen Sie die folgenden zwei Zeilen:

```
1 IF PEEK(45)=PEEK(47) AND PEEK(46)=PEEK(48) THEN 3
2 POKE45,PEEK(174) : POKE46,PEEK(175) : CLR
3 REM Programmbeginn
```

Nach dem Laden des Overlays steht in den Zellen 174 und 175 die Endadresse des gerade geladenen Programms. Diese wird in Zeile 2 in die Zellen 45, 46 kopiert, die auf den Beginn der Variablentabelle zeigen. Die restlichen Basic-Zeiger werden durch CLR richtig gestellt. Die Zeile 1 ist wichtig für das Stadium der Programmentwicklung. Würden Sie nämlich ein mit Zeile 2 präpariertes Programm ändern und dann auch noch starten, würde die Programmgröße wieder auf ihren alten Wert zurückgesetzt, wodurch Zeilen am Programmende verloren gehen können. Nach einer Programmänderung sind die Zellen 45, 46 und 47, 48 gleich; Zeile 1 verzweigt dann in diesem Fall gleich zum Programmbeginn.

2.1.4.3 Parameterübergabe über den Bildschirm

Sehr oft stellt sich die Schwierigkeit, daß irgendwelche Variablen von einem Programm zu einem anderen, das nachgeladen wird, übernommen werden müssen. Dies ist besonders bei Verwendung von Kassettenrekordern der Fall, wo die Daten nicht so einfach zwischengespeichert werden können wie auf einer Floppy. Ein paar Bytes im Betriebssystem, die nicht benutzt werden (Puffer für zweiten Rekorder o.ä.) reichen meist nicht aus, und sind auch nur für Zahlen zwischen 0 und 255 geeignet.

Hier bietet sich die Möglichkeit der Parameterübergabe über den Bildschirm an. Die Daten werden dabei vom ersten Programm auf den Bildschirm geschrieben und vom zweiten Programm von dort eingelesen. Um das ganze automatisch zu machen, müssen jedoch einige Kunstgriffe angewendet werden, und der Rechner muß einen Tastaturpuffer besitzen (beim Commodore 64 zehn Zeichen).

Vom aufrufenden Programm werden die Daten mittels PRINT-Befehlen auf den Bildschirm ausgegeben, wobei zu beachten ist, daß der INPUT-Befehl des aufgerufenen Programmes ein '?' vorgibt, und in der Regel erst ab dem zweiten Zeichen nach dem '?' einliest. Daher dürfen die beiden ersten Stellen einer Bildschirmzeile keine zu übertragenden Daten enthalten. Um Platz zu sparen, können mehrere Variablen hintereinander in eine Zeile geschrieben werden. In diesem

Fall ist nur sicherzustellen, daß im aufgerufenen Programm diese Variablen auch mit **einem** INPUT-Befehl aufgerufen werden (INPUTA,B$,C%,D,E,F$...). Die Daten am Bildschirm sind dabei durch Kommata zu trennen (aber ohne Leerzeichen).

Sind die Daten vom aufrufenden Programm auf dem Bildschirm ausgegeben, so ist für jede Bildschirmzeile ein CHR$(13) (RETURN-Taste) in den Tastaturpuffer (Speicherzellen 631 bis 640) zu schreiben mit:

```
FOR I=0 TO AZ              : REM AZ - Anzahl der Zeilen
   POKE 631+I,13
NEXT
```

Außerdem muß noch die Anzahl der gültigen Zeichen des Tastaturpuffers im dafür vorgesehenen Byte des Betriebssystem-RAM abgespeichert werden, damit die 'RETURN-Tastendrücke' auch anerkannt werden. Dies geschieht mit

```
POKE 198,I
```

Es ist selbstverständlich, daß nach diesen Operationen keine 'Hand-Eingabe' über die Tastatur mehr erfolgen darf, da einerseits bei einer Eingabe zuerst die CHR$(13) des Tastaturpuffers verarbeitet werden, und andererseits für das aufrufende Programm der Tastaturpuffer dann nicht mehr korrekt gefüllt ist.

Im aufgerufenen Programm müssen dann die INPUT-Befehle an der richtigen Stelle des Bildschirms - notfalls mittels Cursorpositionierung - ausgegeben werden. Die beste Möglichkeit ist vor der Ausgabe im aufrufenden Programm ein 'Clear Screen / CLR' und im aufgerufenen Programm ein 'HOME'.

Da der Rechner nach einem INPUT-Befehl den Tastaturpuffer abarbeitet (bis zu einem CHR$(13)), werden die Daten dann automatisch eingelesen. Steht kein Tastaturpuffer zur Verfügung, genügt das Drücken der RETURN-Taste.

Da der Commodore 64 nur einen Tastaturpuffer von zehn Zeichen aufweist und 78 Zeichen je Eingabezeile zur Verfügung stehen, lassen sich somit 780 Zeichen übertragen.

2.1.4.4 Parameterübergabe über freien RAM-Bereich

Eine einfache Möglichkeit, Parameter an Folgeprogramme zu übergeben, ist, die notwendigen Bytes einfach in einen Speicherbereich zu schreiben, der vom Basic-Interpreter nicht beim Laden des Overlays verändert wird. Dazu eignen sich außer den Bereichen, die auch für Maschinenprogramme geeignet sind, unter anderem noch die Zellen des Basic-Eingabepuffers ab $0200=512. Wollen Sie z.B. das Tagesdatum an das Folgeprogramm übergeben, so schreiben Sie im ersten Programm:

```
100 POKE512,T : POKE513,M : POKE514,J
110 LOAD"FOLGEPROGRAMM",8
```

wobei J natürlich die '19' der Jahreszahl nicht enthalten darf, und im Folgeprogramm:

```
100 T=PEEK(512) : M=PEEK(513) : J=PEEK(514)
```

Wenn Sie die erweiterten POKE und PEEK-Funktionen aus Kapitel 2.2.3 geladen haben, können Sie natürlich auch den Bereich unter dem Basic-Interpreter oder dem KERNAL zur Übergabe verwenden.

2.1.4.5 Parameterübergabe über temporäre Dateien

Sind die Daten zu umfangreich, um über den Bildschirm oder einen freien RAM-Bereich übertragen zu werden, so hilft nur die - relative zeitaufwendige - Methode über temporäre Dateien. Das Verfahren funktioniert ähnlich der Übergabe über den Bildschirm, da das aufrufende Programm auch seine Daten mittels PRINT-Anweisungen (hier PRINT#) übergibt und das aufgerufene Programm dies mit INPUT-Befehlen (hier INPUT#) übernimmt. Da es sich in der Regel um fortlaufende Daten handelt, genügt hierzu eine sequentielle Datei. Der Ablauf könnte wie folgt aussehen:

```
100 OPEN 2,8,2,DN$+",S,W"
110 PRINT#2,A",",B%",",C$",",D(2)",",E",",G$",",F%
120 CLOSE2
```

Durch Ausgabe des ',' in Anführungszeichen als Trennzeichen (Delimiter), wird das Lesen der Daten durch den INPUT-Befehl ermöglicht. Das Einlesen muß in der gleichen Reihenfolge geschehen und die Datei muß auch zum Lesen geöffnet werden mit:

```
100 OPEN 2,8,2,DN$+",S,R"
110 INPUT#2,A,B%,C$,D(2),E,G$,F%
120 CLOSE2
```

Beim OPEN-Befehl wird mit der ersten Ziffer jeweils die logische Dateinummer, mit der zweiten die Gerätenummer und mit der dritten die Sekundäradresse besetzt. Für DN$ können Sie einen beliebigen Dateinamen mit den üblichen Einschränkungen heranziehen. Der Teil in den Anführungszeichen bedeutet nur sequentielle Datei (S), Lesen (READ/R) und Schreiben (WRITE/W).

2.1.5 DATA-Anweisungen generieren

```
500 PRINT"{CLR}MACHE DATA-ZEILEN AUS SPEICHERINHALTE
    N"                                                    <096>
510 INPUT"{DOWN}STARTADRESSE IM SPEICHER";SA              <038>
520 INPUT"ANZAHL BYTES";AB                                <214>
530 INPUT"STARTZEILE FUER DATA";SZ                        <024>
540 IF SZ<700 THEN 530                                    <223>
550 PRINT"{CLR}";                                         <211>
560 FOR Z=0 TO (AB-1)/9                                   <041>
570 PRINT MID$(STR$(SZ+Z),2);"DB";                        <211>
580 FOR I=0 TO 8                                          <206>
590 IF 9*Z+I >= AB THEN 620                               <050>
600 PRINT MID$(STR$(PEEK(SA+9*Z+I)),2);",";               <234>
610 NEXT I                                                <047>
620 PRINT CHR$(20)                                        <129>
630 POKE 632+Z,13                                         <062>
640 NEXT Z                                                <094>
650 POKE 631,19                                           <083>
660 POKE 198,Z+1                                          <048>
```

Mit diesem Programm können DATA-Zeilen aus Speicherinhalten generiert werden, was besonders natürlich bei den Sprites von Vorteil ist, wenn diese schon im Speicher stehen, was aber auch bei anderen Programmen (z.B. herstellen eines Basic-Loaders für ein Maschinenprogramm) sinnvoll eingesetzt werden kann.

Zunächst wird die Startadresse im Speicher abgefragt, die die absolute Stelle im RAM angibt. In der Variablen AB wird die Anzahl der Bytes abgefragt, für die die DATA-Statements generiert werden sollen. Als letztes wird noch die Zeilennummer erfragt, in der die DATA-Statements beginnen sollen. Die Startzeile für die Datas muß natürlich größer als 700 sein, da sonst das Programm selbst überschrieben wird.

In der folgenden Schleife werden alle Bytes in Neunergruppen in DATA-Zeilen gepackt. Zunächst wird in Zeile 570 die Zeilennummer der DATA-Zeile und der Befehl DATA am Bildschirm ausgegeben. In einer weiteren Schleife werden dann die Speicherinhalte ausgelesen und als Dezimalzahl am Bildschirm durch Komma getrennt dargestellt. In Zeile 620 wird noch das letzte Komma in der DATA-Zeile eliminiert. Zeile 630 schreibt in den Tastaturpuffer eine '13' hinein, womit später - nach Abschluß des Programms - für jede DATA-Zeile ein RETURN ausgelöst wird, wodurch die Zeile dann in den Speicher übernommen wird. In Zeile 650 wird noch das erste Zeichen im Tastaturpuffer mit dem Befehl 'HOME' belegt, und in Zeile 660 wird die Anzahl der gültigen Zeichen im Tastaturpuffer festgelegt.

2.1.6 Disk-Status bestimmen

Der Disk-Status kann sehr leicht über den sogenannten Kommandokanal (Kanal 15) erfragt werden. Da der Befehl INPUT - und somit auch INPUT# - im Direktmodus nicht zugelassen ist, muß auf alle Fälle ein kleines Programm geschrieben werden. Wenn Sie - wie bei Zeile 30 in unserem Beispiel - den Kommando-Kanal schließen, werden auch alle anderen Floppy-Dateien geschlossen; hier ist also Vorsicht geboten.

```
10 OPEN 15,8,15                                    <005>
20 INPUT#15,A,B$,C,D                               <220>
```

```
30 CLOSE 15                                    <036>
40 PRINT"FEHLERNUMMER :",A                     <054>
50 PRINT"FEHLERTEXT   :",B$                    <214>
60 PRINT"SPUR         :",C                     <012>
70 PRINT"SEKTOR       :",D                     <165>
```

Die Fehlermeldung gliedert sich - in dieser Reihenfolge - in die Fehlernummer, den Fehlertext und gegebenenfalls werden auch noch (z.B. bei READ ERROR) Spur und Sektor des Fehlers angegeben (in der Regel sind diese beiden Werte '0'). Statt der Spur wird aber auch z.B. bei dem Löschen von Dateien die Anzahl der gelöschten Dateien (FILES SCRATCHED) angegeben.

Eine Liste der Fehlermeldungen, sowohl vom Rechner als auch der Floppy, finden Sie im Kapitel 5.

2.1.7 Eingebaute Uhr

Wie auch die anderen Commodore-Rechner, hat der Commodore 64 eine eingebaute Uhr, die mit einer potentiellen Genauigkeit von 1/60 Sekunde arbeitet. Die tatsächliche Genauigkeit dieser Uhr ist jedoch nicht sehr hoch, die Abweichung kann bis zu 30 Minuten pro Tag betragen. Während Floppy-Operationen steht die Uhr fast still. Wie Sie genauere Zeiten erhalten lesen Sie in Kapitel 2.2.11.

Für die Uhrzeit werden in der Zero-Page drei Byte verwendet: die Adressen 160 bis 162 ($A0-$A2). Einen Zugriff von Basic auf die Uhrzeit haben Sie über die Variablen TI und TI$. Das Setzen der Uhrzeit geht ausschließlich über die Variable TI$, die insgesamt sechs Zeichen umfaßt, je zwei Zeichen für Stunden, Minuten und Sekunden. Eine Zuweisung an die Variable TI$ muß immer sechs Zeichen beinhalten. Folgendes kurze Programm bringt Ihnen jeweils die aktuelle Uhrzeit in die linke obere Ecke auf dem Bildschirm:

```
500 PRINT "{CLR}"                              <101>
510 INPUT"ZEIT HHMMSS";TI$                     <208>
520 T$=TI$                                     <245>
530 PRINT"{HOME}";LEFT$(T$,2);":";MID$(T$,3,2);":";R
    IGHT$(T$,2)                                <086>
540 GOTO 520                                   <062>
```

Die Zuweisung in Zeile 520 geschieht aus Gründen der Konsistenz. Während das Basic arbeitet, wird die Uhr trotzdem weiter gesetzt, so daß während des Bearbeitens in Zeile 530 sich die Uhrzeit ändern kann.

Die Variable TI ist ein Zähler, der seit Einschalten bzw. Setzen der Uhr jeweils in Schritten von 1/60 Sekunde geändert wird. Folgendes kurzes Programmstück leistet das gleiche wie obiges Programm:

```
10 PRINT "{CLR}"                                          <122>
20 INPUT"ZEIT HHMMSS";TI$                                 <229>
30 T = TI                                                 <193>
40 H = INT(T/216000)                                      <082>
50 M = INT((T-H*216000)/3600)                             <026>
60 S = INT((T-H*216000-M*3600)/60)                        <052>
70 PRINT "{HOME}"; : X=H : GOSUB 200                      <086>
80 PRINT ":"; : X=M : GOSUB 200                           <140>
90 PRINT ":"; : X=S : GOSUB 200                           <156>
100 GOTO 30                                               <080>
200 PRINT RIGHT$("00"+MID$(STR$(X),2),2);                 <072>
210 RETURN                                                <096>
```

Besonders hier ist die Übergabe der Zeit TI an eine gesonderte Variable (T) wichtig, da während des weiteren Programmlaufes sicherlich mehr als 1/60 Sekunde vergeht.

2.1.8 Zufallsgenerator

Die Basic-Funktion RND arbeitet je nach Vorzeichen des Argumentes unterschiedlich.

Für negative Argumente ist der Wert nicht zufällig, sondern eine Funktion des Arguments.

Für positive Argumente wird der neue Zufallswert aus dem alten durch einen einfachen Algorithmus gebildet. Das Setzen des Startwertes kann durch einen Aufruf der RND-Funktion mit negativen Argumenten erfolgen.

Das Argument 0 bewirkt, daß die Zufallszahl abhängt von dem momentanen Stand des Timers A im CIA1 und dem Stand der Uhr (Sekunden und 1/10 Sekunden) und diesem CIA. Es ist einzusehen, daß weder für den Argumentwert 0 noch für positive Argumente eine optimale Verteilung der Zufallswerte vorliegen kann. Einen etwas besseren Verlauf erhält man, wenn man das Rauschregister des SID verwendet. Das folgende kleine Programm ergibt zufällige Werte zwischen 0 und 255.

```
100 SI=54272      : REM BASISADRESSE SID          <216>
110 POKE SI+24,128 : REM STIMME 3 STILL           <203>
120 POKE SI+18,128 : REM STIMME 3 RAUSCH          <022>
130 POKE SI+15,255 : REM FREQUENZ ST. 3           <206>
200 PRINT PEEK(SI+27) : REM ZUFALLSWERT           <079>
210 GOTO 200          : REM ENDLOSSCHLEIFE        <190>
```

Kehren wir aber nun wieder zu den RND-Zufallsgenerator zurück. In idealisierter Form haben die Dichtefunktion und die Verteilungsfunktion dieses Zufallsgenerators folgendes Aussehen:

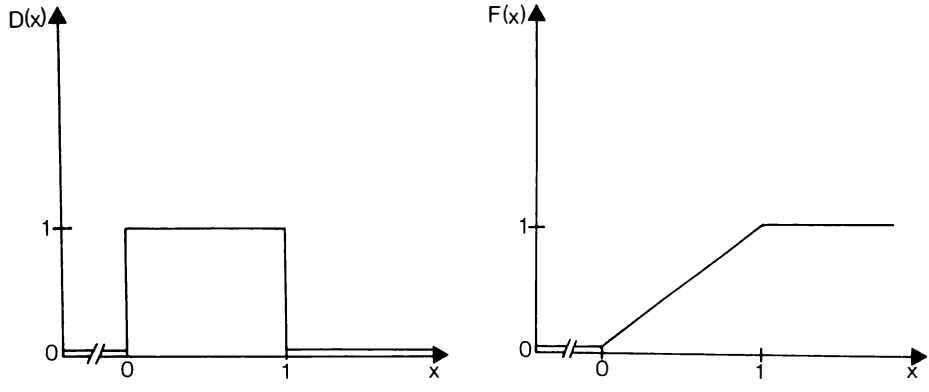


Bild 2.1.8-1 : Dichtefunktion und Verteilungsfunktion von X=RND(1)

Wollen Sie andere Dichtefunktionen, so können Sie - weil die Verteilungsfunktion linear ist - Ihre gewünschte Zufallsfunktion erhalten, indem wir die Umkehrfunktion der gewünschten Verteilungsfunktion der RND-Funktion nachschalten.

Dazu ein Beispiel: Wir wollen eine Zufallsfunktion mit einem Wertebereich von 100 bis 200 und einer parabelförmigen Dichtefunktion, d.h. Werte, die näher bei 100 oder 200 liegen, sollen häufiger kommen als Werte um 150. Dazu müssen wir zunächst die Verteilungsfunktion unserer Dichtefunktion bilden, die definiert ist als Integral von '-unendlich' bis X über die Dichtefunktion. Die folgenden Bilder sollen dies verdeutlichen.

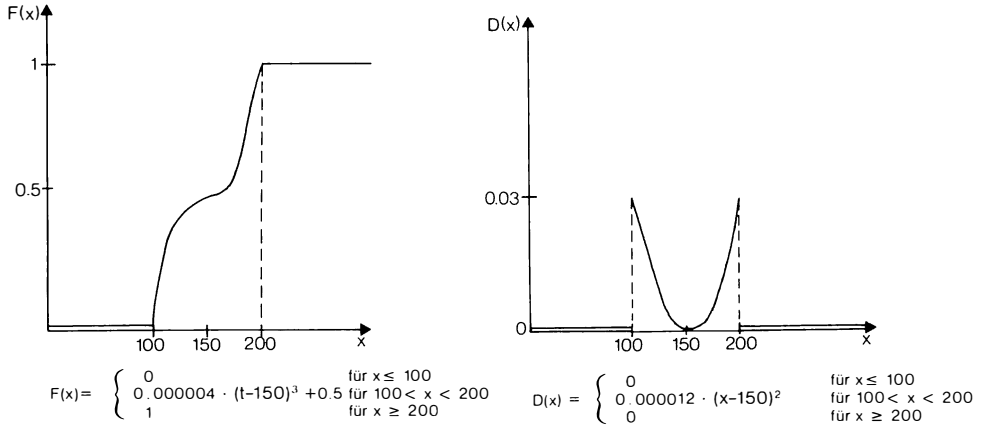


Bild 2.1.8-2 : Dichtefunktion und Verteilungsfunktion des Beispieles

Die Berechnung der Umkehrfunktion geschieht entweder grafisch durch Spiegelung der Abbildung an der Winkelhalbierenden oder durch algebraische Auflösung. Die Umkehrfunktion ist nur dort definiert, wo die Funktion streng monoton steigt, hier also von 0 bis 1.

Nun müssen wir noch die Umkehrfunktion der Verteilungsfunktion bilden. Da die Dichtefunktion quadratische Form hat, ist die Verteilungsfunktion eine kubische Parabel und

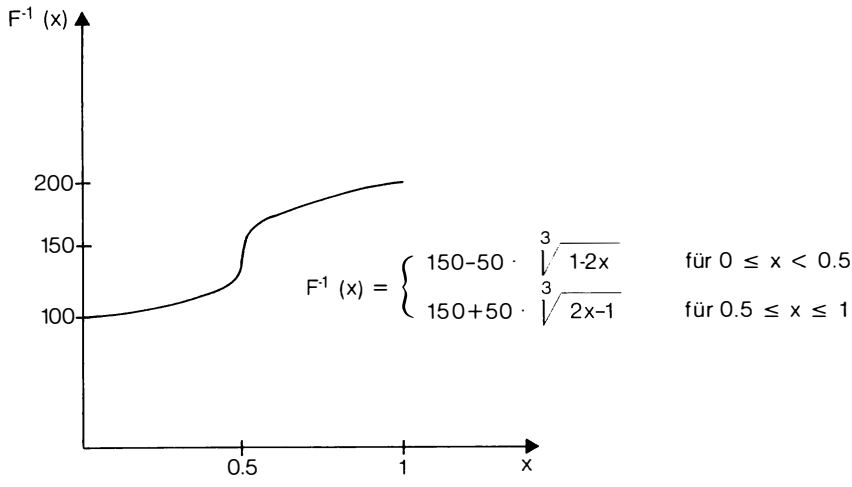


Bild 2.1.8-3 : Umkehrfunktion der Verteilungsfunktion des Beispieles

somit die Umkehrfunktion der Verteilungsfunktion eine Kubikwurzelfunktion. Das entscheidende Resultat ist die Funktionsgleichung der Umkehrfunktion. Diese wollen wir gleich als Basic-Funktion FNUF wie folgt definieren:

```
DEF FNUF(X)=150+SGN(2*X-1)*50*(ABS(2*X-1)**(1/3))
```

Bemerkung: '**' ist der Potenzierungsoperator 'hoch'. Die Konstruktion mit SGN() und ABS() ist notwendig, um die richtige Kubikwurzel zu erhalten, wenn 2x-1 negativ ist.

Durch FNUF(RND(1)) erhalten wir nun Werte, die unserer gewünschten Dichtefunktion genügen. Das eben beschriebene Verfahren ist vielleicht etwas umständlich, aber es eignet sich für beliebige Wertebereiche und Dichtefunktionen, da es sehr allgemein gehalten ist. Bei der Realisierung als Basic-Programm wird allerdings manchmal die Umsetzung der Funktionen in eine Basic-Zeile Schwierigkeiten bereiten. Dann muß man notfalls den Weg beschreiten und den Funktionswert in einem Unterprogramm berechnen, wo man mehr Platz zur Formulierung hat.

2.1.9 Zeichensatz ändern

Zuerst zeigen wir ein kleines Unterprogramm, mit dem Sie den Zeichensatz aus dem ROM in den RAM-Bereich ab $E000 = 57344 kopieren können. Danach beschreiben wir die notwendigen POKE-Befehle zur Verlagerung des Zeichengenerators und des Video-RAM. Zum Schluß zeigen wir Ihnen noch ein Beispiel für neue Zeichen.

```
1000 POKE 56333,1:REM IRQ AUS                          <129>
1005 P=PEEK(1)                                         <054>
1010 POKE 1,P AND 251:REM CHAREN                       <250>
1020 FOR I=53248 TO 57343                              <043>
1030 POKE I+4096,PEEK(I):REM KOPIEREN                  <021>
1040 NEXT                                              <150>
1045 POKE 1,P                                          <093>
1050 POKE 56333,129:REM IRQ EIN                        <018>
1060 RETURN                                            <182>
```

Dieses Unterprogramm schaltet zunächst die Quelle für Interrupts aus und setzt im Prozessorport die Leitung CHAREN auf 0. Dann wird der gesamte Zeichengenerator um 4096 Byte nach hinten verlegt und anschließend die ursprünglichen Werte für den Prozessorport und das Interruptkontroll-Register wieder hergestellt. Dieses Unterprogramm ist nicht notwendig, wenn Sie die Basic-Erweiterungen aus den Kapitel 2.2.4 und 2.2.5 geladen haben. Dann nämlich können Sie statt 'GOSUB 1000' schreiben 'CHARKOP:SWAP$E000,$D000,$1000'.

```
90 V=53248                                             <104>
100 GOSUB 1000:REM CHARKOP:SWAP$E000,$D000,$1000       <038>
110 POKE 56576,148:REM VIC-BEREICH                     <165>
120 POKE V+24,57:REM ADR.-SCHALTER IM VIC              <214>
130 POKE 648,204                                       <125>
140 PRINT"{CLR}"                                       <252>
```

In Zeile 100 wird der Zeichensatz kopiert, in Zeile 110 der Arbeitsbereich des VIC in den letzten 16 KB-Block verlegt, in Zeile 120 die Zeichengenerator-Basisadresse auf das achte und neunte KByte des VIC-Arbeitsbereiches gesetzt und schließlich in Zeile 130 dem Betriebssystem

mitgeteilt, daß sich das Video-RAM nun in der 204. Page (ab $CC00) befindet. Da dort aber noch undefinierte Zeichen stehen, wird der Bildschirm noch in Zeile 140 gelöscht.

```
200 CH=77:GOSUB 500                                  <207>
210 CH=87:GOSUB 500                                  <218>
399 END                                              <016>
400 DATA 14,  6, 10,112,144,144, 96,  0              <169>
410 DATA 56, 68, 68, 56, 16, 56, 16,  0              <109>
500 FOR I=0 TO 7                                     <124>
510 READ A                                           <199>
520 POKE 57344+8*CH+I,A                              <033>
530 NEXT                                             <150>
540 RETURN                                           <172>
```

In diesem kleinen Programmstück zeigen wir Ihnen, wie man die Zeichen für SHIFT-M und SHIFT-W durch das männliche und weibliche Symbol ersetzen kann. Dazu benötigen wir zunächst ein kleines Unterprogramm ab Zeile 500 das aus DATA-Zeilen 8 Werte ausliest und ab einer vorgegebenen Position (57344+8*CH) einträgt. Die Bildschirmcodes der Zeichen SHIFT-M und SHIFT-W sind 77 bzw. 87, die Daten für die neuen Zeichen befinden sich in den Zeilen 400 und 410.

Wenn Sie mit dem Cursor über die neuen Zeichen gehen, merken Sie folgende Besonderheit: Das Reverse-Zeichen ist noch das ursprüngliche SHIFT M bzw. W. Durch einen weiteren POKE-Befehl 'POKE 57344+1024+8*CH+i, NOT A' in Zeile 525 können Sie dies abstellen. Sie sehen daraus, daß Sie insgesamt 512 neue Zeichen definieren können (Umschaltung mit RVS bzw. SHIFT/C=).

2.1.10 Sprite-Editor

Die Verwendung von Sprites ist Sache des Anwenderprogrammes. Für Beispiele sei auf die Buchreihe 'Das Commodore 64-Buch, Band 1, Band 3 und Band 7' hingewiesen. Aber zur Verwendung von Sprites wollen diese zuerst einmal entworfen sein. In diesem Kapitel stellen wir Ihnen einen komfortablen Editor für mehrfarbige Sprites vor.

Haben Sie schon einmal ein Sprite entworfen? Dann kennen

Sie sicher auch die Rechnerei nach dem Entwerfen, bei der Sie jedes einzelne Byte des Sprites ausrechnen müssen. Sieht das Sprite, das Sie mühsam berechnet haben, durch die winzige Darstellung anders aus, als man es sich vorgestellt hat, so verliert man leicht den Spaß am Programmieren. Mit dem folgenden Programm werden Sie bestimmt nicht den Freude am Programmieren verlieren. Im Gegenteil; es wird Ihnen Spaß machen, diese 'digitalen' Bilder zu entwerfen und zur Verwendung in anderen Programmen werden die DATA-Zeilen gleich mitgeliefert.

Bedienungsanleitung

Haben Sie das Programm fehlerlos eingetippt oder eingeladen und mit RUN gestartet, so müssen Sie sich zwischen der Wahl eines einfarbigen oder eines Multicolor-Sprites entscheiden. Haben Sie sich für Letzteres entschieden, so müssen Sie die drei Farben angeben, die verwendet werden sollen. Bei einem einfarbigen Sprite wird hellblau als Farbe gewählt. Im Anschluß daran erfolgt der Aufbau des Spriteeditors. In der linken oberen Ecke des Grafikrahmens erscheint eine Art Cursor, der mit den Cursortasten wie gewohnt gesteuert wird.

Haben Sie den Multicolor-Modus gewählt, so können Sie durch Drücken der Tasten 1 bis 3 einen Punkt mit der entsprechenden Farbe setzen. Im Einfarben-Modus wird ein Punkt durch Drücken der A-Taste gesetzt. Gelöscht wird ein Punkt durch Drücken der Z-Taste. Zur gleichen Zeit, zu der Sie einen Punkt setzen, wird der entsprechende Punkt im Sprite gesetzt. Das Bild des Sprites wird rechts oben neben dem Grafikrahmen in Originalgröße und zusätzlich in Y-Richtung, X-Richtung und in X-/Y-Richtung in doppelter Größe dargestellt. So können Sie leicht kontrollieren, ob das Sprite Ihren Wünschen entspricht. Ist dies nicht der Fall so können Sie dieses durch Drücken der CLR-Taste löschen.

Sprite-Bytes übernehmen

Sind Sie der Meinung, Ihr Sprite vollendet zu haben, so drücken Sie 'B'. Dabei werden die Sprite-Bytes gelesen und in DATA-Zeilen ab der Zeile 1000 abgelegt. Das Programm steht Ihnen nun für einen neuen Spriteentwurf zur Verfügung.

Programm löschen

Haben Sie genug Sprites entworfen, so drücken Sie einfach 'E'. Hierbei wird das gesamte Programm gelöscht. Doch keine Angst, die Sprite-Bytes ab der Zeile 1000 werden nicht gelöscht. Sie können diese sofort in Ihrem eigenen Programm übernehmen.

Programmaufbau

```
  3 -   5 Wahl zwischen Multicolor und einfarbigem Sprite
  6 -  14 (sofern Multicolor-Modus gewählt) Farbwahl
 25 -  50 Grafik erstellen und die Sprites einschalten
 53 -  75 Tastaturabfrage und Cursorbewegung
 77 -  78 Punkt setzen (Ein-Farben-Modus)
 81 -  85 Punkt auf dem Bildschirm und der entsprechende
          im Sprite wird gelöscht.
 87 -  92 ein Punkt mit entsprechender Farbe, je nach
          Tastenwahl (1-3), auf dem Bildschirm und im
          Sprite setzen (Multicolor-Modus)
 94 -  99 Sprite-Bytes in DATA-Zeilen übernehmen
114 - 119 Programm bis Zeile 119 löschen
```

Übrigens: Wenn Sie in Zeile 102 die Variable Z durch einen anderen Wert ersetzen, können Sie den Zeilenanfang der Sprite-Bytes verändern.

Verwendete Adressen im RAM

```
  198          : Anzahl der gedrückten Tasten
  631 -   640  : Tastaturpuffer
  646          : augenblickliche Cursorfarbe
  832 -   895  : Kassettenpuffer als Spriteblock 13
53248 - 53255  : Koordinate der Sprites 1-4
53264          : höchstes Bit für X-Koordinate
53269          : Sprite ein/aus
53271          : Sprite in X-Richtung verdoppeln
53276          : Sprite Multicolor
53277          : Sprite in Y-Richtung verdoppeln
53280          : Randfarbe
53281          : Grundfarbe
53285          : Farbe Multicolor 1
53286          : Farbe Multicolor 2
53287 - 53290  : Farbe der Sprite 1-4
```

Variablenübersicht

F(0-2) : Farbe 1-3
A : Bildschirmanfangsadresse im Grafikrahmen für Grafik
B : Bildschirmanfangadresse im Grafikrahmen für Farbe
N : augenblickliche Spalte, in der der Cursor ist (0,8,16)
X : augenblickliche Teilspalte der Spalte N(0-7)
Q : augenblickliche Byte des Sprites, indem der Cursor ist
Y : augenblickliche Zeile (*40)

Listing

```
1 CLR:POKE 53269,0:Z=1:C=32                                  <222>
2 POKE 53281,0:POKE 53280,0:PRINT"{CLR,BLUE}"                <240>
3 PRINT"{2DOWN,4RIGHT,RVSON}MULTICOLOR SPRITE ?"             <168>
4 WAIT 203,63:GET J$                                          <009>
5 IF J$<>"J"THEN F(2)=14:POKE 53276,0:GOTO 17                <214>
6 PRINT"{CLR,3DOWN,4RIGHT,RVSON,GREEN}FARBE 1:{RVOFF,S
  PACE,RVSON,WHITE,2SPACE,RVOFF}"                             <083>
7 PRINT"{DOWN,4RIGHT,BLUE,RVSON}WAHL MIT <>{RVOFF}"          <188>
8 PRINT"{DOWN,4RIGHT,RVSON}W=WEITER{3SPACE,RVOFF}"           <114>
9 GET A$:IF A$=""THEN 9                                       <015>
10 IF A$=","THEN POKE 646,PEEK(646)-1:IF PEEK(646)=0 T
   HEN POKE 646,16                                            <128>
11 IF A$="."THEN POKE 646,PEEK(646)+1:IF PEEK(646)=255
    THEN POKE 646,238                                         <036>
12 IF A$="W"THEN POKE 1154,PEEK(1154)+1:F(A)=PEEK(646)
   :A=A+1:IF A=3 THEN 15                                      <239>
13 PRINT"{HOME,3DOWN,13RIGHT,RVSON,2SPACE,RVOFF}"            <077>
14 GOTO 9                                                     <208>
15 POKE 53285,F(1):POKE 53286,F(0)                            <109>
16 POKE 53276,15:Z=2                                          <184>
17 FOR A=0 TO 62:POKE 832+A,0:NEXT                            <050>
18 FOR A=0 TO 3:POKE 53287+A,F(2):POKE 2040+A,13:NEXT         <021>
19 POKE 53248,247:POKE 53249,75                               <233>
20 POKE 53250,24:POKE 53251,75                                <165>
21 POKE 53252,247:POKE 53253,109                              <015>
22 POKE 53254,24:POKE 53255,109                               <221>
23 POKE 53277,10:POKE 53271,12                                <165>
24 POKE 53264,10                                              <064>
25 REM *******GRAFIK ERSTELLEN*******                         <086>
26 PRINT"{CLR,LIG.BLUE,DOWN,2RIGHT}012345678901234567 8
   90123"                                                     <079>
27 PRINT"{RIGHT}A************************SA***R*******S
   "                                                          <067>
28 FOR A=0 TO 1:FOR B=0 TO 9                                  <081>
29 PRINT"{LEFT}"B"{LEFT}_{24SPACE}_":NEXT B,A                 <223>
30 PRINT"{RIGHT}Z************************X{HOME}"            <149>
31 A$="{27RIGHT}"                                             <137>
32 PRINT A$"{2DOWN}_{3SPACE}_{6SPACE}_"                       <251>
```

```
33 PRINT A$"⌴{3SPACE}⌴{6SPACE}⌴"                                    <250>
34 PRINT A$"{UP}⌴{3SPACE}⌴{6SPACE}⌴"                                <140>
35 PRINT A$"┌**********┐"                                           <094>
36 PRINT A$"{UP}⌴{3SPACE}⌴{6SPACE}⌴"                                <142>
37 PRINT A$"⌴{3SPACE}⌴{6SPACE}⌴"                                    <254>
38 PRINT A$"{UP}⌴{3SPACE}⌴{6SPACE}⌴"                                <144>
39 PRINT A$"⌴{3SPACE}⌴{6SPACE}⌴"                                    <000>
40 PRINT A$"{UP}⌴{3SPACE}⌴{6SPACE}⌴"                                <146>
41 PRINT A$"⌴{3SPACE}⌴{6SPACE}⌴"                                    <002>
42 PRINT A$"{UP}└**********┘{DOWN}"                                 <233>
43 PRINT A$"{UP,RVSON,ORANGE}A=SETZEN{4SPACE,RVOFF}"                <122>
44 PRINT A$"{RVSON,ORANGE}Z=LOESCHEN{2SPACE,RVOFF}"                 <123>
45 PRINT A$"{UP,RVSON}B=BEARBEITEN{RVOFF}"                          <244>
46 PRINT A$"{RVSON}E=ENDE{6SPACE,RVOFF}"                            <178>
47 IF J$<>"J"THEN 51                                                <039>
48 POKE 646,F(0):PRINT A$"{RVSON}1=FARBE 1{RVOFF}"                  <121>
49 POKE 646,F(1):PRINT A$"{UP,RVSON}2=FARBE 2{RVOFF}"               <014>
50 POKE 646,F(2):PRINT A$"{RVSON}3=FARBE 3{RVOFF}"                  <129>
51 POKE 53269,15                                                    <101>
52 REM *******TASTATURABFRAGE *******                               <111>
53 A=1146:B=55418:C1=160:GOTO 55                                    <077>
54 GET A$:IF A$=""THEN 54                                           <108>
55 POKE A+Y+X+N,C1:POKE B+Y+X+N,C2                                  <093>
56 Q=Y/40*3+INT(N/8)                                                <103>
57 IF A$="{UP}"THEN Y=Y-40                                          <202>
58 IF A$="{DOWN}"THEN Y=Y+40                                        <074>
59 IF A$="{RIGHT}"THEN X=X+Z:IF X>7 THEN X=0:N=N+8:IF
   N>16 THEN N=0                                                    <157>
60 IF A$="{LEFT}"THEN X=X-Z:IF X<0 THEN X=8-Z:N=N-8:IF
    N<0 THEN N=16                                                   <042>
61 IF A$="A"THEN 76                                                 <120>
62 IF A$="Z"THEN 80                                                 <141>
63 IF A$="{CLR}"THEN 93                                             <203>
64 IF A$="B"THEN 100                                                <160>
65 IF A$="E"THEN 113                                                <168>
66 IF A$="1"THEN 87                                                 <111>
67 IF A$="2"THEN 88                                                 <114>
68 IF A$="3"THEN 89                                                 <117>
69 IF Y<0 THEN Y=760                                                <091>
70 IF Y>760 THEN Y=0                                                <090>
71 IF X>23 THEN X=0                                                 <033>
72 IF X<0 THEN X=24-Z                                               <042>
73 C1=PEEK(A+Y+X+N):POKE A+Y+X+N,102                                <142>
74 C2=PEEK(B+Y+X+N):POKE B+Y+X+N,14                                 <100>
75 GOTO 54                                                          <061>
76 REM ************N SET***************                            <047>
77 POKE A+X+Y+N,160:POKE B+X+Y+N,14                                 <134>
78 POKE 832+Q,PEEK(832+Q)OR 128/(2↑X)                               <213>
79 GOTO 73                                                          <066>
80 REM *******SPRITE LOESCHEN********                               <125>
81 K=Z-1                                                            <132>
82 POKE A+X+N+Y+K,32:POKE B+X+N+Y+K,14                              <067>
83 POKE 832+Q,PEEK(832+Q)AND(255-(2↑(7-X-K)))                       <082>
```

```
84 K=K-1:IF K>=0 THEN 82                                        <044>
85 GOTO 73                                                      <072>
86 REM *********MULTISET************                            <248>
87 POKE 832+Q,PEEK(832+Q)OR 128/(2↑X)+128/(2↑(X+1)):GO
   TO 90                                                        <177>
88 POKE 832+Q,PEEK(832+Q)OR 128/(2↑(X+1)):GOTO 90               <055>
89 POKE 832+Q,PEEK(832+Q)OR 128/(2↑X):GOTO 90                   <012>
90 FOR C=0 TO 1                                                 <213>
91 POKE A+X+N+Y+C,160:POKE B+X+N+Y+C,F(VAL(A$)-1)               <247>
92 NEXT:GOTO 69                                                 <016>
93 REM **************"{CLR}"*************                       <049>
94 FOR T=0 TO 62:POKE 832+T,0:NEXT                              <165>
95 FOR Y=0 TO 19                                                <041>
96 FOR X=0 TO 23                                                <036>
97 POKE A+X+Y*40,32:POKE B+X+Y*40,0                             <099>
98 NEXT X,Y:X=0:Y=0                                             <170>
99 GOTO 73                                                      <086>
100 REM ********DATA ZEILE**********                            <122>
101 PRINT"{CLR}":POKE 53269,0                                   <011>
102 U=0:Z= 1000                                                 <164>
103 FOR A=0 TO 7                                                <230>
104 IF U=56 AND A=6 THEN NEXT:GOTO 108                          <194>
105 B=PEEK(832+A+U)                                             <247>
106 D$=STR$(B):F$=F$+MID$(D$,2,3)+","                           <159>
107 NEXT                                                        <237>
108 IF U>57 THEN PRINT"{CLR,BLACK}102 U=0:Z="Z:PRINT"R
    UN":GOTO 112                                                <243>
109 PRINT"{CLR,BLACK}"STR$(Z)+" DATA"+F$+"{LEFT,SPACE}
    "                                                           <131>
110 PRINT"102 U="U+8":Z="Z+1"                                   <019>
111 PRINT"GOTO 101"                                             <023>
112 POKE 631,19:POKE 632,13:POKE 633,13:POKE 634,13:PO
    KE 198,4:END                                                <187>
113 REM *********LOESCHEN**********                            <111>
114 A=1:B=11:POKE 53269,0                                       <092>
115 PRINT"{CLR,BLACK}114 A="B+1":B="B+11"                       <017>
116 FOR I=A TO B:PRINT I:NEXT                                   <239>
117 IF B<100 THEN PRINT"GOTO114"                                <217>
118 IF B>100 THEN POKE 646,14:PRINT"LIST"                       <071>
119 POKE 631,19:FOR I=1 TO 14:POKE 631+I,13:NEXT:POKE
    198,15                                                      <221>
```

2.2 Maschinenroutinen

In diesem Kapitel wollen wir Ihnen zeigen, wie Sie Ihren Rechner mit zusätzlichen Befehlen, Funktionen und Interrupt-Routinen ausstatten können, die in Maschinensprache geschrieben sind. Dazu sei zunächst in Kapitel 2.2.1 ein Rahmenprogramm vorgestellt, in dem sämtliche zusätzliche Befehle und Funktionen mit Namen und Adresse enthalten sind, so daß mit diesen Programmen die neuen Befehle wie

normale Basic-Befehle aufgerufen werden. Daneben gibt es bei einigen Routinen noch die Möglichkeit, diese direkt mit SYS aufzurufen. Bei der Beschreibung der Syntax der Basic-Erweiterungen ist dies jeweils angegeben.

Die Beschreibung der einzelnen Maschinen-Routinen wurde einheitlich aufgebaut. Zunächst kommt der Name der Erweiterung, wobei damit das Label des Maschinenprogramms, das die entsprechende Funktion wirklich ausführt, gemeint ist. In der nächsten Zeile erfolgt eine kurze Erläuterung der Funktion dieser Erweiterung. In der dritten Zeile wird die Syntax, also die Art und Weise, wie der Befehl aufgerufen wird, beschrieben. Handelt es sich um eine Funktion, so wird extra darauf hingewiesen. Danach folgt die Erläuterung der verwendeten Parameter, gefolgt von einem kleinen Beispiel. Als nächstes werden die benötigten Hilfsroutinen aufgelistet, damit Sie wissen, welche Speicherbereiche Sie zusätzlich laden müssen, wenn Sie nur diesen Befehl isoliert verwenden wollen. Dazu erfolgt auch die Auflistung der belegten Adreßbereiche von den einzelnen Routinen und Hilfsroutinen.

Nach der Abbildung des Source-Listings, in dem die Ziel-Adressen für Vorwärtssprünge nicht ausgefüllt sind, folgt die funktionale Beschreibung der Erweiterung.

Wie bekommen Sie nun das Maschinenprogramm in Ihren Rechner? Dazu gibt es drei Möglichkeiten:

- Sie verwenden einen Assembler und geben die benötigten Routinen als Source-Code ein. Dabei müssen Sie natürlich auch die benötigten Hilfsroutinen sowie die Tabelle der Basic-Symbole und der Kernal-Symbole eingeben. Diese Methode hat den Vorteil, daß Sie die Routinen leicht auf Ihre eigenen Bedürfnisse abändern können.

- Sie bedienen sich eines käuflichen Monitor-Programmes und geben die angegebenen Adreßbereiche mit Hilfe des im Anhang abgebildeten Hex-Listings direkt als Maschinenprogramm ein.

- Sie verwenden das Basic-Loader-Programm, das sich ebenfalls im Anhang befindet. In diesem Loader-Programm sind sämtliche Basic-Erweiterungen, die in diesem Buch vorgestellt wurden, enthalten. Falls es Ihnen zu viel Mühe macht, das gesamte Listing abzutippen (es handelt sich immerhin um etwa 7 KByte Maschinenprogramm) können Sie sich die Arbeit er-

leichtern, indem Sie nur diejenigen DATA-Zeilen eingeben, die für die jeweilige Routine und Hilfsroutine benötigt werden. Die Zeilennummer der Data-Zeile entspricht dabei genau der Adresse des ersten Bytes dieser Zeile. Falls die Grenze des benötigten Bereiches mitten in eine Data-Zeile fällt, geben Sie bitte die ganze Zeile ein.

Diese Vorgehensweise wollen wir Ihnen im Beispiel des 16-Bit-POKE-Befehls ("DOKE") verdeutlichen. Bei der Beschreibung des DOKE-Befehls steht unter 'Adreßbereich':

```
DOKE  :    $C457 - $C471  =  50263 - 50289
FLPINT:    $C5EA - $C5FD  =  50666 - 50685
```

Wenn Sie den Basic-Loader aus dem Anhang verwenden wollen, können Sie dann alle DATA-Zeilen weglassen, außer den Zeilen 50256 bis 50288 und den Zeilen 50656 bis 50672.

Die Erweiterung ist dann allerdings nicht mit DOKE, wie ein Basic-Befehl, aufrufbar, weil Sie ja den erweiterten Basic-Befehls-Dekoder (Adreßbereich 51200 bis 51967) nicht geladen haben. Statt dessen können Sie den Befehl mit 'SYS (50263) Adresse,Wert' aufrufen.

Hier noch eine Übersicht über die globale Aufteilung der Speicherbereiche für verschiedene Gruppen der Basic-Erweiterungen:

```
Frei definierbarer Testbereich: $8E00 - $93FF
Grafik-Befehle:                 $9400 - $9DFF
Floppy-Befehle:                 $9E00 - $9FFF
Keyclick und Funktionstasten    $C000 - $C2FF
Speicherverwaltungs-Routinen:   $C400 - $C5E3
Zahlkonvertierungen:            $C5E4 - $C738
String-Befehle:                 $C739 - $C785
PAUSE-Befehl:                   $C786 - $C7CC
Rahmenprogramm:                 $C800 - $CAFF
CIA-Uhrenbehandlung:            $CB00 - $CBFF
```

Zum Schluß des Kapitels stellen wir Ihnen noch ein

Programm zur Unterstützung der Musik-Fähigkeiten des Commodore 64 vor. Dieses Programm ist allerdings nicht mit den anderen kompatibel und sollte mit dem abgebildeten Basic-Loader geladen werden. Es belegt den Adreßbereich von $C000 bis $C13F (Start mit SYS 49152).

2.2.1 Basic-Erweiterungen

Das Rahmenprogramm zum Dekodieren der neuen Befehlsworte und zum Verzweigen auf die neuen Routinen gliedert sich in sechs Abschnitte, welche entsprechend in den Kapiteln 2.2.1.1 bis 2.2.1.6 beschrieben werden. Mit in dieses Kapitel aufgenommen wurde schließlich noch eine Übersicht über sämtliche neue Befehle, sowie eine theoretische Übersicht über die Möglichkeiten, Parameter für Basic-Erweiterungen einzulesen.

Das Rahmenprogramm belegt den Speicherbereich $C800 (51200) bis $CAFF (51967). Es werden keine weiteren Hilfsroutinen benötigt. Allerdings müssen die angesprochenen neuen Befehle natürlich noch geladen werden, bevor man sie ansprechen kann. Der einzige Befehl, der bereits in diesem Speicherbereich mit enthalten ist, ist der Befehl AUS zum Abschalten der Basic-Erweiterung.

2.2.1.1 Symbolvereinbarungen und Wortreservierungen

```
C400            ADR=$14         ;FUER FACADR
C400            DECVEC=$0308    ;ADRESSE DES VEKTORS
C400            ;                DER DEKODIERROUTINE
C400            BASBZ =$7A      ;ENTHAELT BASIC-
C400            ;                BEFEHLS-ZEIGER
C400            BIRQ  =$EA31    ;STANDARD-IRQ-ROUTINE
C400            IRQVEC=$0314    ;VEKTOR FUER IRQ
C400            ARIVEC=$030A    ;VEKTOR F. ARITHM.
C400            VARADL =$47     ;VARIABLENADRESSE
C400            ;
C400            ;BASIC-ROUTINEN
C400            YFLP  =$B3A2    ;WANDELT Y IN FAC
C400            CHRGET=$0073    ;EIN BASIC-Z. LESEN
```

```
C400            CHRGOT=$0079  ;LETZEN KODE LESEN
C400            XFLP  =$BC49  ;WAND. -> FAC; X=EXP.
C400            INTERPRET=$A7AE;INTERPRETERSCHLEIFE
C400            BASBEF=$A7E4  ;BASIC-BEFEHL BEARB.
C400            BASBEF1=$A7E7 ;BASIC-BEFEHL BEARB.
C400            GETARI=$AE86  ;ARITHM. ELEM. AUSW.
C400            GETARI1=$AE8D ;ARITHM. ELEM. AUSW.
C400            FACADR=$B7F7  ;FAC NACH ADR,ADR+1
C400            CHKKLA=$AEFA  ;KLAMMER AUF PRUEFEN
C400            CHKKLZ=$AEF7  ;KLAMMER ZU PRUEFEN
C400            FEHLER=$A437  ;FEHLERMELDUNG AUSG.
C400            SYNTERR=$AF08 ;'SYNTAX ERROR'
C400            FRMNUM=$AD8A  ;HOLT NUM. AUSDRUCK
C400            CHNUM =$AD8D  ;PRUEFT AUF NUMERISCH
C400            FRMKLA=$AEF1  ;HOLT AUSDRUCK IN ()
C400            FRESTR=$B6A3  ;HOLT STRING
C400            FRMEVL=$AD9E  ;HOLT AUSDRUCK
C400            CHKCOM=$AEFD  ;PRUFT AUF KOMMA
C400            STROUT=$AB1E  ;GIBT STRING AUS
C400            GETBYT=$B79E  ;HOLT BYTE-PARAMETER
C400            GETBYTC=$B7F1 ;CHKCOM & GETBYT
C400            PARLOSA=$E1D4 ;HOLT LOAD/SAVE-PARAMETER
C400            GETAB  =$B7EB ;HOLT 2 PARAMETER
C400            BCHKIN =$E11E ;SETZT EINGABEDATEI
C400            BCHKOUT=$E118 ;SETZT AUSGABEDATEI
C400            BGETIN =$E124 ;HOLT EIN ZEICHEN
C400            BSOUT  =$E10C ;GIBT EIN ZEICHEN AUS
C400            BASIN  =$E112 ;ZEICHEN EINLESEN
C400            NEUSTR =$B4F4 ;PLATZ F. NEUEN STR.
C400            VARSUC =$B08B ;SUCHT VARIABLE
C400            FACXY  =$BBD4 ;FAC NACH VARIABLE
C400            FMALK  =$BA28 ;FAC MAL KONSTANTE
C400            PUSHSTR=$B4CA ;DES. AUF STRINGSTACK
C400            ADROUT =$BDCD ;ADRESSE AUSGEBEN
C400            SPOUT  =$AB3F ;LEERZEICHEN AUSGEBEN
C400            ;
C400            ACPTR =$FFA5 ; BYTE HOLEN
C400            CHRIN =$FFCF ; BYTE VOM KANAL HOLEN
C400            CHROUT=$FFD2 ; BYTE AUF KANAL AUSG.
C400            CIOUT =$FFA8 ; BYTE AUSGEBEN
C400            CLRCHN=$FFCC ; KANAL ABMELDEN
C400            LISTEN=$FFB1 ; LISTEN-KOMMANDO
C400            LOAD  =$FFD5 ; LADE-ROUTINE
C400            READST=$FFB7 ; STATUS LESEN
C400            SAVE  =$FFD8 ; SPEICHER-ROUTINE
C400            SECOND=$FF93 ; SEK.ADR. NACH LISTEN
C400            STOP  =$FFE1 ; PRUEFT STOPTASTE
C400            TALK  =$FFB4 ; TALK SENDEN
C400            TKSA  =$FF96 ; SEK.ADR NACH TALK
C400            UNLSN =$FFAE ; UNLISTEN SENDEN
C400            UNTALK=$FFAB ; UNTALK SENDEN
```

```
C800            ; *** SPRUNGLEISTE ***
C800  4C21C8    JMP INIT
C803  4C5BC8    JMP AUS
C806  4CB8CA    JMP INTMANS
C809  00      IRQM:     BYT 0  ;MASKE F. IRQ-ROUT.
C80A          ISPTAB:          ;SPRUNGTAB. IRQ-ROUT.
C80A  0C90      WOR $900C      ;SMOVE
C80C  00C0      WOR $C000      ;KEYKLICK
C80E  30C0      WOR $C030      ;FTAST
C810  20C8      WOR IDUMMY
C812  20C8      WOR IDUMMY
C814  20C8      WOR IDUMMY
C816  20C8      WOR IDUMMY
C818  20C8      WOR IDUMMY
C81A  80C8    IADR:    WOR IDUMMY ;PLATZ F. SPRUNGZ.
C81C  00      IRQZ:     BYT 0 ;ZAEHLER IRQ-ROUTINEN
C81D  00      BEFNR:    BYT 0 ; NUMMER DES BEFEHLS
C81E  00      FNNR:    BYT 0  ; NUMMER DER FUNKTION
C81F  00      AKKUSICH:    BYT 0 ;ZWISCHENSPEICHER
C820            ;
C820  60      IDUMMY:    RTS  ; DUMMY-ROUTINE
C821            :
```

Hier werden zunächst die Symbole für die Basic- und KERNAL-Routinen vereinbart, dann folgt die Sprungtabelle auf die wichtigsten drei Labels dieses Abschnittes, und schließlich werden noch 23 Byte reserviert, die unter anderem die Sprungtabelle für den Interrupt-Manager aufnehmen sollen.

In diese Sprungtabelle werden später die Adressen der Interrupt-Routinen eingetragen, die nacheinander automatisch aufgerufen werden sollen. Die ersten Adressen sind bereits eingetragen und zwar für die Spritebewegung, den Key-Click und die Funktionstastenabfrage. Die restlichen Adressen sind mit dem Label IDUMMY vorbesetzt, welches auf eine Routine zeigt, die lediglich ein RTS enthält.

2.2.1.2 Initialisierung

```
C821            ;
C821            ;INITIALISIERUNG
C821            ;
```

```
C821          INIT:
C821  A980      LDA #BEFERWL  ;DECODE-VECTOR AUF
C823  8D0803    STA DECVEC    ;VERTEILER FUER NEUE
C826  A9C8      LDA #BEFERWH  ;BEFEHLE STELLEN
C828  8D0903    STA DECVEC+1
C82B            ;
C82B  A9FC      LDA #FKTERWL  ;ARTITHM.-VEKTOR AUF
C82D  8D0A03    STA ARIVEC    ;AUSWERTUNG VON $/%-
C830  A9C9      LDA #FKTERWH  ;AUSDRUECKEN STELLEN
C832  8D0B03    STA ARIVEC+1
C835            ;
C835  A98E      LDA #$8E
C837  8538      STA $38       ;BASIC-RAM END=$8E00
C839  8534      STA $34       ;STRING-UNTERGRENZE
C83B            ;
C83B  78        SEI
C83C  A9B3      LDA #INTMANL  ;IRQ-VEKTOR AUF
C83E  8D1403    STA IRQVEC    ;MANAGER STELLEN
C841  A9CA      LDA #INTMANH
C843  8D1503    STA IRQVEC+1
C846  58        CLI
C847            ;
C847  A94F      LDA #MELDUNGL  ;ZEIGER (A/Y) AUF
C849  A0C8      LDY #MELDUNGH  ;MELDUNG
C84B  201EAB    JSR STROUT     ;STRING AUSGEBEN
C84E  60        RTS        ;INITIALISIERUNG FERTIG
C84F            ;
C84F            MELDUNGL = <$
C84F            MELDUNGH = >$
C856  312E30    %ASC "PROFI V1.0"
C859  0D        BYT $0D
C85A  00        BYT 0
C85B            ;
```

In diesem Abschnitt werden einige Basic-Vektoren verstellt, um neue Basic-Befehle, neue Basic-Funktionen und bis zu 8 Interrupt-Routinen ansprechen zu können. Der Vektor $0308, $0309 wird auf die Befehlsdekodier-Routine (siehe Kapitel 2.2.1.4) gestellt, der Vektor $030A, $030B auf die Funktiondekodier-Routine (Kapitel 2.2.1.5), dann wird das Basic-RAM auf den Bereich bis $8E00 begrenzt, so daß genügend Platz für eigene Maschinenprogramme bleibt. Nachdem schließlich noch der IRQ-Vektor ($0314, $0315) auf

den Interrupt-Manager gerichtet wurde, wird noch eine Meldung auf dem Bildschirm ausgegeben.

2.2.1.3 Der AUS-Befehl

```
C85B            ;** ERWEITERUNG AUSSCHALTEN **
C85B            ;
C85B          AUS:
C85B  A9E4      LDA #<BASBEF
C85D  8D0803    STA DECVEC
C860  A9A7      LDA #>BASBEF
C862  8D0903    STA DECVEC+1    ;DECODE-VEKTOR
C865            ;                AUF NORMALEN WERT
C865  A986      LDA #<GETARI    ;ARITH.-VEKTOR AUF
C867  8D0A03    STA ARIVEC      ;NORMALEN WERT
C86A  A9AE      LDA #>GETARI    ;STELLEN
C86C  8D0B03    STA ARIVEC+1
C86F            ;
C86F  78        SEI
C870  A931      LDA #<BIRQ      ;IRQ-VEKTOR AUF
C872  8D1403    STA IRQVEC      ;STANDARDWERT
C875  A9EA      LDA #>BIRQ      ;SETZEN
C877  8D1503    STA IRQVEC+1
C87A  58        CLI
C87B            ;
C87B  A9A0      LDA #$A0
C87D  8538      STA $38         ;BASIC-RAM END=$A000
C87F  60        RTS             ;FERTIG
C880            ;
```

Hier werden die bei der Initialisierung verstellten Basic-Vektoren auf ihre ursprünglichen Standardwerte zurückgesetzt und das Basic-RAM bis $A000 freigegeben.

2.2.1.4 Dekodierung von Befehlen

```
C880            BEFERWH = >$    ; ** NEUE BEFEHLE **
C880            BEFERWL = <$    ; ** AUSFUEHREN   **
C880  207300    JSR CHRGET      ;BASIC-ZEICHEN HOLEN
```

```
C883  901E      BCC BEBAS      ;CODE WAR ZIFFER
C885  C960      CMP #$60       ;TEST AUF BUCHSTABE
C887  B01A      BCS BEBAS      ;BASIC-BEFEHLS-CODE
C889  C941      CMP #$41       ;VERGLEICH MIT "A"
C88B  9015      BCC BEBASC     ;SONDERZEICHEN
C88D  8D1FC8    STA AKKUSICH   ;AKKU SICHERN
C890  A200      LDX #0
C892  8E1DC8    STX BEFNR      ;BEFNR =0
C895          BE1:
C895  A000      LDY #0
C897  EE1DC8    INC BEFNR      ;BEFNR=BEFNR+1
C89A  BD24C9    LDA BTAB,X     ;ZEICH. AUS BEF.TAB.
C89D  D007      BNE BE2        ;KEIN TRENNZEICHEN
C89F  AD1FC8    LDA AKKUSICH   ;AKKU RESTAURIEREN
C8A2          BEBASC:
C8A2  38        SEC            ;CARRY WIEDER SETZEN
C8A3          BEBAS:
C8A3  4CE7A7    JMP BASBEF1    ;BASIC-BEF. AUSF.
C8A6          BE2:
C8A6  D17A      CMP (BASBZ),Y  ;VERGL. M. BASIC-TEXT
C8A8  D028      BNE BE4        ;KEINE UEBEREINST.
C8AA  C8        INY            ;NEXT ZEI. BASIC-TEXT
C8AB  E8        INX            ;NEXT ZEI. BEF.TAB.
C8AC  BD24C9    LDA BTAB,X     ;ZEICHEN AUS BEF.TAB.
C8AF  D0F5      BNE BE2        ;NEXT ZEICH. PRUEFEN
C8B1  18        CLC
C8B2  98        TYA
C8B3  657A      ADC BASBZ      ;BEFEHLSZEIGER UM
C8B5  857A      STA BASBZ      ;BEF.-LAENGE ERHOEH.
C8B7  9002      BCC BE3        ;KEIN UEBERTRAG
C8B9  E67B      INC BASBZ+1    ;UEBERTRAG
C8BB          BE3:
C8BB  AD1DC8    LDA BEFNR      ;BEFEHLSNUMMER HOLEN
C8BE  0A        ASL A          ;VERDOPPELN UND ALS
C8BF  AA        TAX            ;ZEIGER IN SPRUNGTAB.
C8C0  BDDCC8    LDA STAB,X
C8C3  8DCDC8    STA BEFADR
C8C6  BDDDC8    LDA STAB+1,X
C8C9  8DCEC8    STA BEFADR+1   ;BEFADR=SPRUNGZIEL
C8CC            ;
C8CC  20        BYT $20        ;CODE FUER 'JSR'
C8CD  0000    BEFADR:   WOR 0  ;SPRUNGZIEL
C8CF  4CAEA7    JMP INTERPRET  ;ZURUECK ZUR
C8D2            ;               INTERPRETERSCHLEIFE
C8D2            ;
C8D2          BE4:
C8D2  E8        INX            ;NEXT ZEI. BEF.TAB.
C8D3  BD24C9    LDA BTAB,X
C8D6  D0FA      BNE BE4        ;BEFEHL NICHT ZU ENDE
C8D8  E8        INX            ;'0' UEBERLESEN
C8D9  4C95C8    JMP BE1        ;NEXT BEFEHL CHECKEN
```

```
C8DC            ;
C8DC            ;SPRUNGTABELLE BEFEHLE
C8DC            ;
C8DC          STAB:
C8DC  E7A7      WOR BASBEF1     ;BEFNR=0
C8DE  5BC8      WOR AUS         ;AUS
C8E0  57C4      WOR DOKE        ;DOKE
C8E2  9EC4      WOR ROKE        ;ROKE
C8E4  B3C4      WOR CHARKOP     ;CHARKOP
C8E6  18C5      WOR SWAP        ;SWAP
C8E8  77C5      WOR LAD         ;LAD
C8EA  A7C5      WOR SPEI        ;SPEI
C8EC  8BC7      WOR PAUSE       ;PAUSE
C8EE  009E      WOR $9E00       ;DISK
C8F0  039E      WOR $9E03       ;LEST
C8F2  069E      WOR $9E06       ;LESP
C8F4  099E      WOR $9E09       ;VIEW
C8F6  00CB      WOR $CB00       ;SETTIME
C8F8  008E      WOR $8E00       ;TEST-BEFEHL
C8FA  0094      WOR $9400       ;GREIN
C8FC  0394      WOR $9403       ;GRAUS
C8FE  0694      WOR $9406       ;GRLOE
C900  0994      WOR $9409       ;FARBE
C902  0C94      WOR $940C       ;GRAFEIN
C904  0F94      WOR $940F       ;PUNKT
C906  1294      WOR $9412       ;CHAR
C924            ;
C924          BTAB:
C924  415553    %ASC "AUS"
C927  00        BYT 0
C929  4F4B45    %ASC "DOKE"
C92C  00        BYT 0
C92E  4F4B45    %ASC "ROKE"
C931  00        BYT 0
C936  4B4F50    %ASC "CHARKOP"
C939  00        BYT 0
C93B  574150    %ASC "SWAP"
C93E  00        BYT 0
C93F  4C4144    %ASC "LAD"
C942  00        BYT 0
C944  504549    %ASC "SPEI"
C947  00        BYT 0
C94A  555345    %ASC "PAUSE"
C94D  00        BYT 0
C94F  49534B    %ASC "DISK"
C952  00        BYT 0
C954  455354    %ASC "LEST"
C957  00        BYT 0
C959  455350    %ASC "LESP"
C95C  00        BYT 0
C95E  494557    %ASC "VIEW"
```

```
C961  00        BYT 0
C966  494D45    %ASC "SETTIME"
C969  00        BYT 0
C96B  455354    %ASC "TEST"
C96E  00        BYT 0
C971  45494E    %ASC "GREIN"
C974  00        BYT 0
C977  415553    %ASC "GRAUS"
C97A  00        BYT 0
C97D  4C4F45    %ASC "GRLOE"
C980  00        BYT 0
C983  524245    %ASC "FARBE"
C986  00        BYT 0
C98B  45494E    %ASC "GRAFEIN"
C98E  00        BYT 0
C991  4E4B54    %ASC "PUNKT"
C994  00        BYT 0
C996  484152    %ASC "CHAR"
C999  00        BYT 0
C9FB  00        BYT 0          ;ENDE BEFEHLSTABELLE
C9FC            :
```

In diesem Unterprogramm werden die Zeichen ab der momentanen Stelle des Basic-Befehlszeigers mit den Wörter aus der Befehlstabelle verglichen. Wird ein Befehlswort im Basic-Text gefunden, so wird an die entsprechende Routine verzweigt, deren Adresse in der Sprungtabelle vermerkt ist. Der Basic-Befehlszeiger steht nach dieser Dekodierung auf dem ersten Byte hinter dem Befehl. Die Dekodierung erfolgt nicht, wenn das erste Zeichen des vermeintlichen neuen Befehls kein Buchstabe ist. Deshalb ist es auch nicht möglich, neue Befehlswörter mit einem anderen Zeichen als einem Buchstaben beginnen zu lassen.

Haben Sie einen Assembler, so ist es einfach, die Sprungtabelle und die Befehlstabelle um entsprechende selbstdefinierte neue Befehle zu erweitern. Ein Befehl wird vom nächsten Befehl durch ein 0-Byte getrennt, das Ende der Befehlstabelle wird durch zwei aufeinanderfolgende 0-Bytes erkannt. Für diejenigen unter Ihnen, die keinen Assembler besitzen, um diese Erweiterung durchzuführen, wurde am Schluß der Befehlstabelle ein Befehl 'TEST' eingefügt, dessen Adresse hier konstant auf $8E00 gesetzt wurde. Dadurch können Sie die Routine für einen neuen Befehl dort ablegen und diesen mit TEST aufrufen.

2.2.1.5 Dekodieren von Funktionen

```
C9FC                FKTERWL=<$     ;** ARITHM. ELEM.**
C9FC                FKTERWH=>$     ;** AUSWERTUNG   **
C9FC  A900          LDA #0
C9FE  850D          STA $0D        ;TYPFLAG NUMERISCH
CA00  207300        JSR CHRGET     ;NAECHSTES ZEICHEN
CA03  902A          BCC FNBAS      ;CODE WAR ZIFFER
CA05  C9C5          CMP #$C5       ;VAL-KODE
CA07  F05C          BEQ FKTVAL
CA09  C960          CMP #$60       ;TEST AUF BUCHSTABE
CA0B  B022          BCS FNBAS      ;
CA0D  C924          CMP #$24       ;"$"
CA0F  F057          BEQ HEXCON     ;JA, DANN HEXZAHL
CA11  C925          CMP #$25       ;"%"
CA13  F056          BEQ BINCON     ;JA, DANN BINZAHL
CA15  C941          CMP #$41       ;VERGLEICH MIT "A"
CA17  9015          BCC FNBASC     ;SONDERZEICHEN
CA19  8D1FC8        STA AKKUSICH   ;AKKU SICHERN
CA1C  A200          LDX #0
CA1E  8E1EC8        STX FNNR       ;BEFNR =0
CA21              FE1:
CA21  A000          LDY #0
CA23  EE1EC8        INC FNNR       ;BEFNR=BEFNR+1
CA26  BD82CA        LDA FTAB,X     ;ZEICH. AUS BEF.TAB.
CA29  D007          BNE FE2        ;KEIN TRENNZEICHEN
CA2B  AD1FC8        LDA AKKUSICH   ;AKKU RESTAURIEREN
CA2E              FNBASC:
CA2E  38            SEC            ;CARRY WIEDER SETZEN
CA2F              FNBAS:
CA2F  4C8DAE        JMP GETARI1    ;BASIC-BEF. AUSF.
CA32              FE2:
CA32  D17A          CMP (BASBZ),Y  ;VERGL. M. BASIC-TEXT
CA34  D025          BNE FE4        ;KEINE UEBEREINST.
CA36  C8            INY            ;NEXT ZEI. BASIC-TEXT
CA37  E8            INX            ;NEXT ZEI. BEF.TAB.
CA38  BD82CA        LDA FTAB,X     ;ZEICHEN AUS BEF.TAB.
CA3B  D0F5          BNE FE2        ;NEXT ZEICH. PRUEFEN
CA3D  18            CLC
CA3E  98            TYA
CA3F  657A          ADC BASBZ      ;BEFEHLSZEIGER UM
CA41  857A          STA BASBZ      ;BEF.-LAENGE ERHOEH.
CA43  9002          BCC FE3        ;KEIN UEBERTRAG
CA45  E67B          INC BASBZ+1    ;UEBERTRAG
CA47              FE3:
CA47  AD1EC8        LDA FNNR       ;BEFEHLSNUMMER HOLEN
CA4A  0A            ASL A          ;VERDOPPELN UND ALS
CA4B  AA            TAX            ;ZEIGER IN SPRUNGTAB.
CA4C  BD6ECA        LDA FSTAB,X
CA4F  8D59CA        STA FNADR
```

```
CA52  BD6FCA    LDA FSTAB+1,X
CA55  8D5ACA    STA FNADR+1    ;FNADR=SPRUNGZIEL
CA58  4C        BYT $4C        ;CODE FUER 'JMP'
CA59  0000    FNADR:   WOR 0   ;SPRUNGZIEL
CA5B            ;
CA5B          FE4:
CA5B  E8        INX            ;NEXT ZEI. BEF.TAB.
CA5C  BD82CA    LDA FTAB,X
CA5F  D0FA      BNE FE4        ;BEFEHL NICHT ZU ENDE
CA61  E8        INX            ;'0' UEBERLESEN
CA62  4C21CA    JMP FE1        ;NEXT BEFEHL CHECKEN
CA65            ;
CA65          FKTVAL:
CA65  4CD1C6    JMP VAL
CA68            ;
CA68          HEXCON:
CA68  4C9BC6    JMP HEXCONV
CA6B            ;
CA6B          BINCON:
CA6B  4CB6C6    JMP BINCONV
CA6E            ;
CA6E            ;
CA6E            ;SPRUNGTABELLE FUNKTIONEN
CA6E            ;
CA6E          FSTAB:
CA6E  8DAE      WOR GETARI1    ;FNNR=0
CA70  13C7      WOR HEXSTR     ;HEX$
CA72  26C7      WOR BINSTR     ;BIN$
CA74  37C4      WOR DEEK       ;DEEK
CA76  01C4      WOR REEK       ;REEK
CA78  72C4      WOR DREEK      ;DREEK
CA7A  39C7      WOR INDEX      ;INDEX
CA7C  05C4      WOR CHEEK      ;CHEEK
CA7E  8BCB      WOR $CB8B      ;TIME
CA80  008E      WOR $8E00      ;TEST-FUNKTION
CA82            ;
CA82          FTAB:
CA83  455824    %ASC "HEX$"
CA86  00        BYT 0
CA88  494E24    %ASC "BIN$"
CA8B  00        BYT 0
CA8D  45454B    %ASC "DEEK"
CA90  00        BYT 0
CA92  45454B    %ASC "REEK"
CA95  00        BYT 0
CA98  45454B    %ASC "DREEK"
CA9B  00        BYT 0
CA9E  444558    %ASC "INDEX"
CAA1  00        BYT 0
```

```
CAA4  45454B    %ASC "CHEEK"
CAA7  00        BYT 0
CAA9  494D45    %ASC "TIME"
CAAC  00        BYT 0
CAAE  455354    %ASC "TEST"
CAB1  00        BYT 0
CAB2  00        BYT 0           ;ENDE FUNKTIONSTAB.
CAB3            ;
```

Im Prinzip geschieht hier das gleiche wie bei der Befehlsdekodierung. Ein wesentlicher Unterschied ist jedoch, daß drei Codes extra dekodiert werden. Dies ist zum einen der Code für den Basic-Befehl VAL, zum anderen das $-Zeichen und das %-Zeichen. Taucht einer dieser drei Codes auf, wird sofort zu den Labeln VAL, HEXCONV, BINCONV verzweigt, die nicht in der eigentlichen Funktions-Sprungtabelle stehen.

Die Funktions-Sprungtabelle und die Funktions-Wort-Tabelle sind genauso aufgebaut wie die Tabellen bei den Befehlen; auch hier wurde eine TEST-Funktion vorgesehen, mit der Sie leicht experimentieren können.

2.2.1.6 Interrupt-Manager

```
CAB3            INTMANL=<$      ; ** IRQ-MANAGER **
CAB3            INTMANH=>$
CAB3  A901      LDA #1
CAB5  8D1CC8    STA IRQZ        ;ZAEHLER AUF 1
CAB8          INTMANS:
CAB8  AD1CC8    LDA IRQZ        ;ZAEHLER UND-VERKN.
CABB  2D09C8    AND IRQM        ;MIT MASKE
CABE  F003      BEQ INTMAN1     ;FLAG NICHT GES.
CAC0  20CBCA    JSR INTAUSF     ;EINEN INT. AUSF.
CAC3          INTMAN1:
CAC3  0E1CC8    ASL IRQZ        ;ZAEHLERBIT ROTIEREN
CAC6  90F0      BCC INTMANS     ;SCHLEIFE
CAC8  4C31EA    JMP BIRQ        ;FERTIG
CACB            ;
CACB          INTAUSF:
CACB  A2FE      LDX #$FE        ;X = -2
CACD          INTAUSF1:
CACD  E8        INX             ;X UM 2 ERHOEHEN
CACE  E8        INX
```

```
CACF  4A       LSR A          ;AKKU NACH RECHTS
CAD0  90FB     BCC INTAUSF1   ;BIS NUMMER GEFUNDEN
CAD2  BD0BC8   LDA ISPTAB+1,X;SPRUNGZIEL HIGH/LOW
CAD5  8D1BC8   STA IADR+1     ;IN RESERVIERTE
CAD8  BD0AC8   LDA ISPTAB,X   ;ZELLEN SCHREIBEN
CADB  8D1AC8   STA IADR
CADE  6C1AC8   JMP (IADR)     ;INDIREKTER SPRUNG
CAE1           ;
```

Die folgende Interrupt-Manager-Routine wurde geschrieben, um den Betrieb von mehreren Interrupt-Routinen zu ermöglichen und außerdem das Ein- und Ausschalten dieser Routinen zu erleichtern.

Beim Einschalten der Basic-Erweiterungen wird der IRQ-Vektor auf den Manager gestellt. Beim Ausschalten wird der Standardwert wieder hergestellt, so daß das Zwischenspeichern des alten Wertes überflüssig ist. Von $C80A bis $C819 befindet sich eine Tabelle, in der die Adressen der acht Interrupt-Routinen, die aufgerufen werden können, abgelegt werden. In der Zelle $C809 bedeutet ein gesetztes Bit, daß die der Bitnummer entsprechende Routine aufgerufen werden soll. Das Ein- und Ausschalten der Routinen geschieht also nur noch, indem das entsprechende Bit in dieser Zelle gesetzt oder gelöscht wird. Die Plätze für die nicht benützten Routinen wurden mit einer Dummy-Routine belegt, die einfach aus einem RTS besteht.

Ab dem Label INTMANL, INTMANH befindet sich die Schleife, in der die einzelnen Interrupt-Routinen aufgerufen werden, wenn das Zählerbit in der Zelle IRQZ und das entsprechende Bit in der oben erwähnten Maskenzelle IRQM gesetzt sind. Die Schleife wird beendet, wenn das Zählerbit herausroutiert wurde.

Beim Ausführen eines Interrupts wird zunächst das X-Register mit dem doppelten Wert der Bitnummer des gesetzten Bits im Akku geschrieben und anschließend werden die entsprechenden Werte aus der Sprungtabelle in die Zellen IADR und IADR+1 übertragen, über welche dann ein indirekter Sprung ausgeführt wird.

2.2.1.7 Übersicht der neuen Befehle

Befehl ('*' - Funktion)	Bedeutung
$hhhh	* Präfix für Hexadezimalwert
%bbbbbbbbbbbbbbbb	* Präfix für Binärwert
AUS	Befehlserweiterung ausschalten
BIN$(AD)	* Binärstring zu AD bilden
CHEEK(AD)	* PEEK in Zeichengenerator-ROM
CHAR PF,PX,PY,BC	Zeichen in Grafik
CHARKOP	Zeichengenerator-ROM in darunterliegendes RAM kopieren
DEEK(AD)	* 16-Bit-PEEK
DISK	Disk-Status einlesen in DS$
DOKE AD,W	16-Bit-POKE
DREEK(AD)	* 16-Bit-PEEK in RAM unter ROM
GRAUS	Grafik ausschalten
GRAFEIN	Grafik wiedereinschalten
GREIN HF,ZF	Grafik einschalten (einfarbig)
GREIN F1,F2,F3	Grafik einschalten (mehrfarbig)
HEX$(AD)	* Hexadezimalstring zu AD bilden
INDEX(A$,AA$)	* String-in-String-Suche
LAD ZA,DN$,GA,SA	Laden ohne Veränderung von Basic-Zeigern
LESP# DN,Z$	Programmzeile lesen
LEST# DN,Z$	Textzeile lesen
PAUSE SEK	Pause
PUNKT PF,PX,PY	Punkt setzen, löschen, invertieren
REEK(AD)	* PEEK in RAM unter ROM
ROKE AD,B	POKE in RAM unter I/O
SETTIME U,ZE$	Uhrzeit/Alarmzeit im CIA setzen
SPEI AA,EA,DN$,GA,SA	RAM-Bereich abspeichern
SWAP A1,A2,AZ	RAM-Bereiche austauschen
TIME bzw. TIME2	* Uhrzeit aus CIA1 (CIA2) lesen
VAL(V$)	* Zahlwert einer Zeichenreihe
VIEW DN$,GA	Programmdatei direkt listen

Dabei haben die Parameter folgende Bedeutung:

Parameter	Bedeutung
b	Binärziffer
h	Hex-Ziffer
A$	Zeichenreihe, die gesucht wird
A1	Quelladresse beim Vertauschen
A2	Zieladresse beim Vertauschen

AA	Startadresse beim Speichern
AA$	Zeichenreihe, in der gesucht wird
AD	Adresse, Zahl zwischen 0 und 65535
AZ	Anzahl der zu vertauschenden Bytes
B	Byteparameter (Zahl von 0 bis 255)
BC	Bildschirmcode (0-511)
DN	logische Dateinumer
DN$	Dateiname
EA	Endadresse+1 beim Speichern
F1	erste Farbe
F2	zweite Farbe
F3	dritte Farbe
GA	Geräteadresse
HF	Hintergrundfarbe
PF	Punktfarbe
PX	X-Koordinate eines Punktes
PY	Y-Koordinate eines Punktes
SA	Sekundäradresse
SEK	Sekunden
U	Nummer der zu setzenden Zeit: 1 = Uhr im CIA1, 2 = Uhr im CIA2 3 = Alarm im CIA1, 4 = Alarm im CIA2
V$	String (dezimal, $hex. oder %binär), dessen Wert bestimmt werden soll
W	Wortparameter (Zahl von 0 bis 65535)
Z$	eingelesene Zeile
ZF	Zeichenfarbe (-1, 0, 1, 2, 3)

2.2.1.8 Parameterübergabe

Die meisten Befehle und Funktionen besitzen einen oder mehrere Parameter. Parameter bedeutet hier eine Größe, von deren Wert das Verhalten der Funktion des Befehls abhängt. Zunächst wollen wir uns ansehen, welche Datentypen als Parameter in Frage kommen, und anschließend wollen wir uns mit den Möglichkeiten beschäftigen, diese Datentypten bei Befehlen und Funktionen als Parameter zu verwenden. Als Beispiel wollen wir uns dazu die Integer-Division ansehen. Diese Routine hat zwei Eingabeparameter, nämlich Dividend und Divisor, und das Ergebnis (ganzahliger Quotient oder Rest) als Ausgabeparameter. Zu jeder Möglichkeit wollen wir ein kleines Rahmenprogramm für die Integer-Division schreiben, das die Parameter in entsprechender Weise übernimmt.

Numerische Parameter

Zahlen werden im Commodore-Basic immer als Fließkommazahlen verarbeitet. In dieser Form werden zur Speicherung mindestens fünf Byte (8 Bit Exponent, 32 Bit Mantisse) benötigt. Für viele Anwendungen ist jedoch nur ein eingeschränkter Zahlbereich von Interesse. Dies sind vor allem folgende:

Byte-Parameter:	ganze Zahlen von	0	bis +255
Adress-Parameter:	ganze Zahlen von	0	bis +65535
Integer Werte:	ganze Zahlen von	0	bis +65535
Signed Integer:	ganze Zahlen von	-32768	bis +32767

Variablen als Parameter

Auch eine Variable kann als Parameter übergeben werden, wobei darunter zu verstehen ist, daß die Adresse der Variablen der eigentliche Parameter ist. Der Inhalt der Variablen kann dann dabei sowohl als Eingabeparameter als auch als Ausgabeparameter benützt werden.

String-Parameter

Ein String ist beim Commodore-Basic eine Zeichenreihe mit einer Länge von 0 bis 255. Als Parameter für einen Befehl oder eine Funktion wird jedoch eigentlich nur der sogenannte Descriptor übergeben. Dies sind 3 Byte, wovon der erste die Länge des Strings und die beiden anderen die Startadresse des Strings angeben.

Außer der Möglichkeit, Strings als Variableninhalte zu übergeben, besteht auch die Möglichkeit, mit Hilfe der Routine FRMEVL derartige Parameter auszuwerten. Nach Beendigung der Routine FRMEVL steht der Descriptor des Strings im Stringstapel. Von dort kann er mit der Routine FRESTR geholt werden, wonach die Startadresse des Strings in den Zellen $22, $23 steht und die Länge im Akkumulator.

Wollen wir Strings als Funktionswert, so müssen wir den Descriptor vor Verlassen unserer Routine auf den Stapel legen. Dies geschieht mit Hilfe der Basic-Routine PUSHSTR (Adresse: $B4CA=46282), welche ihrerseits als Eingabeparameter die Stringlänge in der Zelle $61 und Stringadresse in $62,$63 (L,H) benötigt.

Parameterübergabe durch spezifizierte Speicherzellen

Vor allem für Byte- und Word-Parameter eignet sich folgende Methode: Im Maschinenprogramm werden einige Speicherzellen vereinbart, aus denen zu Beginn der Routine die Eingabeparameter gelesen und in die vor Beendigung der Routine die Ausgabeparameter zurückgeschrieben werden.

Von Basic aus kann man die Eingabeparameter mittels POKE übergeben und nach Aufruf der Routine die Ausgabe mit PEEK vollziehen. Diese Methode ist zwar vom Basic-Programm etwas umständlich zu bedienen, jedoch im Maschinenprogramm leicht zu realisieren und leicht zu verstehen.

Parameterübergabe durch Prozessor-Register

Der SYS-Befehl des Commodore 64 hat eine angenehme Eigenschaft: Unmittelbar vor Aufruf der eigentlichen Maschinenroutine werden aus den Speicherzellen 780, 781, 782 und 783 die Prozessor-Register, Akkumulator, X-Register, Y-Register und Statusregister geladen und nach Aufruf der Routine die neuen Werte in diese Zellen zurückgeschrieben. Dadurch steht in dem Moment, wo die Maschinenroutine begonnen wird, ein vom Benutzer definierter Zustand in diesen Prozessor-Registern.

Als Beispiel wollen wir Ihnen eine Möglichkeit zeigen, mit Hilfe der KERNAL-Routine PLOT den Cursor an eine bestimmte Position zu setzen.

```
10 POKE 781, Zeile
20 POKE 782, Spalte
30 POKE 783, 0      :REM Alle Flags auf Null
40 SYS 65520        :REM PLOT-Routine
```

Weitere Beispiele für Parameterübergaben

Hier haben wir als Beispiel die Integer-Division, die im Kapitel 2.2.10 beschrieben wird, ausgewählt. Die Routine verlangt in den Zellen $8F47, $8F48 den Dividenden und in $8F45, $8F46 den Divisor in der Form: Low-Byte, High-Byte. Das Ergebnis (der Quotient) wird in den Zellen $8F47 und $8F48, der Rest in den Zellen $8F49 und $8F4A abgelegt.

Diese Angaben genügen, um Programme zu schreiben, die die Parameter für die Routine IDIV entsprechend setzen.

Die Methode 'Parameterübergabe durch spezifizierte Speicherzellen' können wir hier direkt anwenden, da die angegebenen Speicherzellen in einem Bereich liegen, der vom Basic-Interpreter nicht verändert wird. Der Aufruf vom Basic-Programm könnte also so aussehen:

```
10 D1=13 : REM Dividend
20 D2=3  : REM Divisor
30 POKE 36680, D1/256 : POKE 33679, D1-256*INT(D1/256)
40 POKE 36678, D2/256 : POKE 33677, D2-256*INT(D2/256)
50 SYS 36608
60 PRINT PEEK(36679)+256*PEEK(36680), : REM Quotient
70 PRINT PEEK(36681)+256*PEEK(36682)  : REM Rest
```

Etwas einfacher sieht es aus, wenn Sie die Basic-Erweiterungen aus Kapitel 2.2 benutzen:

```
10 D1=13 : REM Dividend
20 D2=3  : REM Divisor
30 DOKE 36679, D1
40 DOKE 36677, D2
50 SYS $8F00
60 PRINT DEEK(36679),                 : REM Quotient
70 PRINT DEEK(36681)                  : REM Rest
```

Integer-Division als Befehl

Eingabe-Parameter aus Basic-Text lesen

```
8E00  209EAD   JSR FRMEVL  ;AUSDRUCK (DIVIDEND) HOLEN
8E03  208DAD   JSR CHNUM   ;NUMERISCH PRUEFEN
8E06  20F7B7   JSR FACADR  ;NACH INTEGER WANDELN
8E09  A514     LDA $14     ;LOW-BYTE
8E0B  8D478F   STA $8F47   ;DES DIVIDENDEN
8E0E  A515     LDA $15     ;HIGH-BYTE
8E10  8D488F   STA $8F48
8E13  20FDAE   JSR CHKCOM
8E16  208AAD   JSR FRMNUM  ;WERT (DIVISOR) HOLEN
8E19  20F7B7   JSR FACADR  ;NACH INTEGER WANDELN
8E1C  A514     LDA $14     ;LOW-BYTE
8E1E  8D458F   STA $8F45   ;DES DIVISORS
8E21  A515     LDA $15     ;HIGH-BYTE
8E23  8D468F   STA $8F46
```

```
8E26  20008F    JSR $8F00    ;IDIV-ROUTINE
8E29  60        RTS          ;FERTIG
8E2A
```

Die Eingabeparameter werden hier direkt aus dem Basic-Text gelesen. Das erspart uns einige POKE-Befehle, auch können hier komplizierte Ausdrücke als Parameter auftreten. Zur Auswertung der Parameter benützen wir zunächst die Basic-Routine FRMEVL. Diese Routine wertet einen beliebigen Parameter bis zum nächsten auftauchenden Trennzeichen (Komma, Doppelpunkt o.ä.) aus. Das funktioniert sowohl für Stringparameter als auch für numerische Parameter. Der Typ wird in der Zelle $0D (=13) angezeigt, die 0 ist, wenn es sich um einen numerischen Parameter handelt, und $FF (255) bei einem String-Parameter. Zur Überprüfung dieses Flags existiert die Routine CHNUM, die 'TYPE MISMATCH ERROR' ausgibt, wenn der Typ nicht numerisch ist.

Für den kombinierten Aufruf FRMEVL und CHNUM gibt es auch die Routine FRMNUM. Nach dem Aufruf dieser Routine steht der Parameter im Fließkomma-Akkumulator (Kurz: FAC). Für die Integer-Division brauchen wir ihn jedoch als Integerzahl, wozu wir die Routine FACACR aufrufen, welche den Fließkomma-Akkumulator in eine Integerzahl umwandelt und in $14,$15 (Low, High) ablegt. Die Trennung der beiden Parameter wollen wir durch ein Komma bewerkstelligen, die Überprüfung der Syntax erfolgt dann mit der Routine CHKCOM.

Die Ausgabeparmater werden noch genauso behandelt wie im vorigen Fall, so daß die Anwendung im Basic-Programm etwa so aussieht:

```
10 D1=13 : REM Dividend
20 D2=3  : REM Divisor
50 SYS (36608) D1,D2
60 PRINT PEEK(36679)+256*PEEK(36680), : REM Quo.
70 PRINT PEEK(36681)+256*PEEK(36682)  : REM Rest
```

Oder mit Basic-Erweiterungen:

```
10 D1=13 : REM Dividend
20 D2=3  : REM Divisor
50 TEST D1,D2
60 PRINT DEEK(36679),          : REM Quotient
70 PRINT DEEK(36681)           : REM Rest
```

Integer-Division als Funktion

Hier beschränken wir uns darauf, den Quotienten als Funktionswert zu übergeben.

```
8E00  20FAAE   JSR CHKKLA  ;KLAMMER AUF PRUEFEN
8E03  209EAD   JSR FRMEVL  ;AUSDRUCK (DIVIDEND) HOLEN
8E06  208DAD   JSR CHNUM   ;NUMERISCH PRUEFEN
8E09  20F7B7   JSR FACADR  ;NACH INTEGER WANDELN
8E0C  A514     LDA $14     ;LOW-BYTE
8E0E  8D478F   STA $8F47   ;DES DIVIDENDEN
8E11  A515     LDA $15     ;HIGH-BYTE
8E13  8D488F   STA $8F48
8E16  20FDAE   JSR CHKCOM
8E19  208AAD   JSR FRMNUM  ;WERT (DIVISOR) HOLEN
8E1C  20F7AE   JSR CHKKLZ  ;KLAMMER ZU PRUEFEN
8E1F  20F7B7   JSR FACADR  ;NACH INTEGER WANDELN
8E22  A514     LDA $14     ;LOW-BYTE
8E24  8D458F   STA $8F45   ;DES DIVISORS
8E27  A515     LDA $15     ;HIGH-BYTE
8E29  8D468F   STA $8F46
8E2C  20008F   JSR $8F00   ;IDIV-ROUTINE
8E2F  AD478F   LDA $8F47   ;LOW-BYTE ERGEBNIS
8E32  8563     STA $63     ;FUER XFLP-ROUTINE
8E34  AD488F   LDA $8F48   ;HIGH-BYTE ERGEBNIS
8E37  8562     STA $62
8E39  38       SEC         ;FLAG FUER POSITIV
8E3A  A290     LDX #$90    ;EXPONENT= 16 (+$80)
8E3C  4C49BC   JMP XFLP    ;NACH FLIESSK. WANDELN
8E3F
```

Die Syntax des Funktionsaufrufs soll wie folgt aussehen:
PRINT IDIV(Dividend, Divisor)

Da das Ergebnis im FAC zurückgegeben werden soll, kann diese Routine nicht mehr mit SYS aufgerufen werden. Statt dessen kann die Funktion als USR-Funktion definiert werden oder mit Hilfe des in Kapitel 2.2.1.5 beschriebenen Funktionsdekoders angesprungen werden. Bei Realisierung als USR-Funktion entfallen die Aufrufe von CHKKLA, FRMEVL und CHKKLZ, welche automatisch vom Basic-Interpreter ausgeführt werden. Unbedingt notwendig ist die Umwandlung des erhaltenen Integer-Ergebnisses nach Fließkomma.

Für den Anfänger ist gerade der wichtige Punkt der Parameterübergabe ein schwieriges Problem. Jedoch finden Sie im Kapitel 2.2 und in #6# und #7# noch ausreichend Beispiele, die Ihnen die möglichen Vorgehensweisen erläutern.

2.2.2 Zahlkonvertierungen

In diesem Kapitel wollen wir verschiedene Routinen zur Umwandlung zwischen den Datenformaten 'Integerzahl', 'Fließkommazahl', 'Binär-String', 'Hex-String' und 'Dezimal-String' beschreiben. Es handelt sich hierbei um Maschinenroutinen, die für sich nicht von Basic aus aufgerufen werden können. Die nötigen Rahmenprogramme sind in Kapitel 2.2.11 zusammengestellt.

2.2.2.1 Integer nach Fließkomma

Name	INTFLP
Funktion	16-Bit-Zahl in Fließkommazahl umwandeln
Aufruf	JSR INTFLP
Parameter	$62: High-Byte der Integerzahl $63: Low-Byte der Integerzahl FAC: Fließkommazahl (Ergebnis)
Adressbereich	$C5E4 - $C5E9 = 50660 - 50665

```
C5E4            INTFLP:
C5E4  A290        LDX #$90      ;EXPONENT + $80
C5E6  38          SEC           ;POSITIV
C5E7  4C49BC      JMP XFLP      ;ROM-ROUTINE
C5EA              ;
```

Beschreibung Im X-Register wird der Exponent der gedachten Fließkommazahl festgehalten, hier also '+16'. Das Carry-Flag muß gesetzt werden, da es sich um eine positive Zahl handeln soll. Danach wird die Basic-Routine XFLP angesprungen, die alles übrige erledigt.

2.2.2.2 Fließkomma nach Integer

Name	FLPINT
Funktion	Fließkomma-Akkumulator in 16-Bit-Integerzahl umwandeln
Aufruf	JSR FLPINT
Parameter	FAC: Fließkommazahl $62: Integerwert High-Byte $63: Integerwert Low-Byte
Adressbereich	$C5EA - $C5FD = 50666 - 50685

```
C5EA          FLPINT:
C5EA  A514      LDA $14         ;$14,$15 SICHERN
C5EC  48        PHA
C5ED  A515      LDA $15
C5EF  48        PHA
C5F0  20F7B7    JSR FACADR      ;FAC -> $14,$15 (Y/A)
C5F3  8562      STA $62         ;HIGH-BYTE
C5F5  8463      STY $63         ;LOW-BYTE
C5F7  68        PLA             ;$14,$15
C5F8  8515      STA $15         ;RESTAURIEREN
C5FA  68        PLA
C5FB  8514      STA $14
C5FD  60        RTS
C5FE            ;
```

Beschreibung Hier wird die Basic-Routine FACADR verwendet, die den FAC in $14, $15 sowie im Akku/Y-Register ablegt. Um den alten Wert von $14, $15 nicht zu überschreiben, wird er zunächst gesichert und anschließend wieder hergestellt.

2.2.2.3 Signed-Integer nach Fließkomma

Name	SGIFLP
Funktion	vorzeichenbehaftete 16-Bit-Zahl in Fließkommazahl umwandeln
Aufruf	JSR $B399
Parameter	$62: High-Byte der Integerzahl $63: Low-Byte der Integerzahl

FAC: Fließkommazahl (Ergebnis)
Adressbereich im Basic-ROM

Beschreibung Die Routine arbeitet von den Parametern her völlig analog zu unserer Routine INTFLP, jedoch wird hier das höchstwertigste Bit (MSB) der 16-Bit-Zahl mit der Wertigkeit -32768 belegt.

2.2.2.4 Fließkomma nach Signed-Integer

Name FLPSGI
Funktion FAC in vorzeichenbehaftete 16-Bit-Zahl umwandeln
Aufruf JSR FLPSGI
Adressbereich $C5FE - $C605 = 50686 - 50693

```
C5FE          FLPSGI:
C5FE  20AAB1    JSR $B1AA      ;FLPAY (ROM)
C601  8562      STA $62        ;HIGH-BYTE
C603  8463      STY $63        ;LOW-BYTE
C605  60        RTS
C606            ;
```

Beschreibung In Basic befindet sich eine Routine, die den FAC in eine 16-Bit vorzeichenbehaftete Zahl umwandelt, und das Ergebnis in Akku- und Y-Register zurückgibt. Diese beiden Prozessor-Register werden in den Zellen $62 und $63 abgelegt.

2.2.2.5 Dezimalstring nach Fließkomma

Name ASCFLP
Funktion String aus Dezimalziffern in Fließkommawert umwandeln
Aufruf JSR $B7AD
Parameter $22, $23: Startadresse des umzuwandelnden Strings

	Akkumulator:Länge des Strings FAC: Wert des Strings
Adressbereich	in Basic-ROM
Beschreibung	Diese Routine funktioniert nur, wenn der String eine von 0 verschiedene Länge hat. Dann entspricht die Routine der VAL-Funktion.

2.2.2.6 Fließkomma nach Dezimalstring

Name	FLPASC
Funktion	FAC in Zeichenreihe aus Dezimalziffer umwandeln
Aufruf	JSR $BDDD
Parameter	FAC: Fließkommawert Speicherzellen ab $00FF: beinhalten umgewandelten FAC als String
Adressbereich	innerhalb des Basic-ROM
Beschreibung	Diese Routine wandelt den FAC entsprechend der STR$-Funktion um.

2.2.2.7 Hexadezimal nach Integer

Name	HEXINT
Funktion	String aus Hexadezimalziffern in Integerwert umwandeln
Aufruf	JSR HEXINT
Parameter	$22, $23:Startadresse des Strings Y-Register: Länge des Strings Y-Register: Anzahl der ausgewerteten Zeichen
Adressbereich	$C606 - $C63C = 50694 - 50748

```
C606          HEXINT:
C606  8461      STY $61       ;MAXIMALZAHL MERKEN
C608  A000      LDY #0
C60A  8462      STY $62
C60C  8463      STY $63
```

```
C60E          HEXINT1:
C60E  C461      CPY $61         ;MAXIMALZAHL VERGL.
C610  F02A      BEQ HEXINT9
C612  B122      LDA ($22),Y     ;NAECHSTES ZEICHEN
C614  38        SEC
C615  E930      SBC #$30        ;CODE FUER "0" SUB.
C617  C90A      CMP #10
C619  900A      BCC HEXINT2     ;KLEINER 10, DANN OK
C61B  E907      SBC #7          ;"A" ENTSPRICHT 10
C61D  C90A      CMP #10
C61F  901B      BCC HEXINT9     ;KLEINER 10, DANN E.
C621  C910      CMP #16
C623  B017      BCS HEXINT9     ;>=16, DANN ENDE
C625          HEXINT2:
C625  0663      ASL $63         ;$63,$62 MIT 16
C627  2662      ROL $62         ;MULTIPLIZIEREN
C629  0663      ASL $63
C62B  2662      ROL $62
C62D  0663      ASL $63
C62F  2662      ROL $62
C631  0663      ASL $63
C633  2662      ROL $62
C635  0563      ORA $63         ;WERT IN DIE LETZTEN
C637  8563      STA $63         ;4 BIT EINTRAGEN
C639  C8        INY             ;NAECHSTE HEXZIFFER
C63A  D0D2      BNE HEXINT1
C63C          HEXINT9:
C63C  60        RTS
C63D            ;
```

Beschreibung: Die maximale Länge wird in der Zelle $61 abgelegt, die Zellen $62 und $63 dienen zur Aufnahme des Ergebnisses und werden annulliert. Ist die maximale Länge erreicht oder ist das folgende Zeichen keine Hex-Ziffer, so wird zu HEXINT9 verzweigt, wo das Unterprogramm beendet wird. Zur Berechnung wird jeweils der alte Wert mit 16 multipliziert und dann der neue Wert hinzugeodert.

2.2.2.8 Binär nach Integer

Name BININT
Funktion Wandle Zeichenreihe aus binären Ziffern in

ganzzahligen Wert um
Aufruf JSR BININT
Parameter Y-Register : maximale Anzahl der zuverarbeitenden Zeichen; Anzahl der verarbeiteten Zeichen
$22, $23 Startadresse der Zeichenreihe
$63, $62 Ergebnis (L,H)
Adressbereich $C63D - $C65C = 50749 - 50780

```
C63D          BININT:
C63D  8461      STY $61       ;MAXIMALZAHL MERKEN
C63F  A000      LDY #0
C641  8462      STY $62
C643  8463      STY $63
C645          BININT1:
C645  C461      CPY $61
C647  F013      BEQ BININT9
C649  B122      LDA ($22),Y   ;NAECHSTE ZIFFER
C64B  C930      CMP #$30      ;"0"
C64D  900D      BCC BININT9
C64F  C931      CMP #$31      ;CARRY BEACHTEN !
C651  F002      BEQ BININT2
C653  B007      BCS BININT9
C655          BININT2:
C655  2663      ROL $63       ;CARRY REINROTIERN
C657  2662      ROL $62
C659  C8        INY
C65A  D0E9      BNE BININT1   ;SCHLEIFE
C65C          BININT9:
C65C  60        RTS
C65D            ;
```

Beschreibung Die maximale Länge wird in $61 zwischengespeichert, das Ergebnis annuliert. Ist die Länge erreicht, oder das nächste Zeichen keine Binärziffer, so wird zu BININT9 verzweigt. Der Wert der Binär-Ziffer wird im Carry-Flag durch geschickten Vergleich gewonnen, und anschließend in die Speicherzellen $63, $62 hineinrotiert, wodurch automatisch die Verdoppelung des alten Wertes vorgenommen wird.

2.2.2.9 Integer nach Hexadezimal

Name	INTHEX
Funktion	Integerwert in hexadezimale Zeichenreihe umwandeln
Aufruf	JSR INTHEX
Parameter	$63, $62 Eingabewert (L,H) Y : Anzahl der benötigten Stellen (1-4) Ab $0100 : Ergebnis als Zeichenreihe Abschluß mit 0-Code
Adressbereich	$C65D - $C685 = 50781 - 50821

```
C65D          INTHEX:
C65D  A900      LDA #0
C65F  990001    STA PUFFER,Y ;ENDEMARKIERUNG
C662  88        DEY
C663          INTHEX1:
C663  A563      LDA $63
C665  290F      AND #$0F     ;4 BIT VON $63
C667  C90A      CMP #$0A
C669  9002      BCC INTHEX2  ;KLEINER 10
C66B  6906      ADC #$06     ;CARRY GESETZT !
C66D          INTHEX2:
C66D  6930      ADC #$30     ;CARRY CLEAR !
C66F  990001    STA PUFFER,Y ;EINTRAGEN
C672  4662      LSR $62      ;$63,$62 DURCH
C674  6663      ROR $63      ;16 TEILEN
C676  4662      LSR $62
C678  6663      ROR $63
C67A  4662      LSR $62
C67C  6663      ROR $63
C67E  4662      LSR $62
C680  6663      ROR $63
C682  88        DEY
C683  10DE      BPL INTHEX1  ;SCHLEIFE
C685  60        RTS
C686            ;
```

Beschreibung Zunächst wird der 0-Code an den Schluß des Ergebnis-Strings im Pufferbereich ab $0100 geschrieben. Dann werden die hintersten 4 Bit des Wertes in eine hexadezimale Zahl umgewandelt, und diese im Puffer abgelegt und dann die hexadezimale Zahl um 4 Bit nach rechts verschoben.

2.2.2.10 Integer nach Binär

Name	INTBIN
Funktion	Integerzahl in binäre Zeichenreihe umwandeln
Aufruf	JSR INTBIN
Parameter	$63, $62 Eingabewert (L,H) Y : Anzahl der benötigten Stellen (1-4) Ab $0100 : Ergebnis als Zeichenreihe Abschluß mit 0-Code
Adressbereich	$C686 - $C69B = 50822 - 50843

```
C686          INTBIN:
C686  A900       LDA #0
C688  990001     STA PUFFER,Y ;ENDEMARKIERUNG
C68B  88         DEY
C68C          INTBIN1:
C68C  4662       LSR $62      ;BIT RAUSSCHIEBEN
C68E  6663       ROR $63
C690  A900       LDA #0
C692  6930       ADC #$30     ;$30 = ASC("0")
C694  990001     STA PUFFER,Y ;EINTRAGEN
C697  88         DEY
C698  10F2       BPL INTBIN1  ;SCHLEIFE
C69A  60         RTS
C69B             ;
```

Beschreibung Zunächst wird der Pufferbereich mit 0 abgeschlossen, danach ein Bit nach dem anderen aus dem Eingabeparameter herausrotiert und in eine binäre Ziffer umgewandelt.

2.2.2.11 Rahmenprogramme zur Nutzung der Zahlkonvertierungen als Basicerweiterungen

Die in den Kapitel 2.2.2.1 bis 2.2.2.10 beschriebenen Routinen sind allgemeine Konvertierungs-Routinen, für die die Parameter extra noch gesetzt werden müssen. Deshalb sind in diesem Kapitel sämtliche Rahmenprogramme vorgestellt, die bewerkstelligen, daß bei Aufruf der Zahlkonvertierungen als Basic-Funktion die Parameter entsprechend gesetzt und verarbeitet werden.

Adressbereich: $C69B - $C738 = 50843 - 51000

```
C69B          HEXCONV:
C69B  207300    JSR CHRGET  ;1. ZEICHEN ANPEILEN
C69E  A57A      LDA BASBZ   ;BASIC-BEFEHLS-ZEIGER
C6A0  8522      STA $22     ;-> STRINGZEIGER
C6A2  A57B      LDA BASBZ+1
C6A4  8523      STA $23
C6A6  A004      LDY #4      ;MAX. 4 HEXZIFFERN
C6A8  2006C6    JSR HEXINT  ;UMWANDLUNG
C6AB  20FBA8    JSR $A8FB   ;BAS.BZ. UM Y ERH.
C6AE  A000      LDY #0
C6B0  840D      STY $0D     ;TYPFLAG NUMERISCH
C6B2  20E4C5    JSR INTFLP  ;ERGEBNIS -> FAC
C6B5  60        RTS         ;FERTIG
C6B6            ;
C6B6          BINCONV:
C6B6  207300    JSR CHRGET  ;1. ZEICHEN ANPEILEN
C6B9  A57A      LDA BASBZ   ;BASIC-BEFEHLS-ZEIGER
C6BB  8522      STA $22     ;-> STRINGZEIGER
C6BD  A57B      LDA BASBZ+1
C6BF  8523      STA $23
C6C1  A010      LDY #16     ;MAX. 16 BINZIFFERN
C6C3  203DC6    JSR BININT  ;UMWANDLUNG
C6C6  20FBA8    JSR $A8FB   ;BAS.BZ. UM Y ERH.
C6C9  A000      LDY #0
C6CB  840D      STY $0D     ;TYPFLAG NUMERISCH
C6CD  20E4C5    JSR INTFLP  ;ERGEBNIS -> FAC
C6D0  60        RTS         ;FERTIG
C6D1            ;
C6D1          VAL:
C6D1  207300    JSR CHRGET  ;VAL-CODE UEBERLESEN
C6D4  20F1AE    JSR FRMKLA  ;WERT IN () HOLEN
C6D7  2082B7    JSR $B782   ;FRESTR / TYPFLAG
C6DA  8561      STA $61     ;STRINGLAENGE
C6DC  A000      LDY #0
C6DE  B122      LDA ($22),Y ;ERSTES ZEICHEN
C6E0  C924      CMP #$24    ;"$"
C6E2  F013      BEQ HEXVAL  ;VAL VON HEXZAHL
C6E4  C925      CMP #$25    ;"%"
C6E6  F018      BEQ BINVAL  ;VAL VON BINAERZAHL
C6E8  A561      LDA $61     ;STRINGLAENGE
C6EA  4CB0B7    JMP $B7B0   ;ZUR VAL-FUNKTION
C6ED            ;
C6ED          VALPAR:       ;HILFSROUTINE
C6ED  A461      LDY $61     ;STRINGLAENGE
C6EF  88        DEY         ;MINUS 1
C6F0  E622      INC $22     ;STRINGADRESSE
C6F2  D002      BNE VALPAR1 ;PLUS 1
C6F4  E623      INC $23
```

```
C6F6          VALPAR1:
C6F6  60        RTS
C6F7            ;
C6F7          HEXVAL:
C6F7  20EDC6    JSR VALPAR  ;STRING VERKUERZEN
C6FA  2006C6    JSR HEXINT  ;UMWANDLUNG
C6FD  4CE4C5    JMP INTFLP  ;ERGEBNIS -> FAC
C700            ;
C700          BINVAL:
C700  20EDC6    JSR VALPAR  ;STRING VERKUERZEN
C703  203DC6    JSR BININT  ;UMWANDLUNG
C706  4CE4C5    JMP INTFLP  ;ERGEBNIS -> FAC
C709            ;
C709          INTPAR:       ;HILFSROUTINE
C709  20F1AE    JSR FRMKLA  ;AUSDRUCK IN () HOLEN
C70C  208DAD    JSR CHNUM   ;NUMERISCH PRUEFEN
C70F  20EAC5    JSR FLPINT  ;-> INTEGER
C712  60        RTS
C713            ;
C713          HEXSTR:
C713  2009C7    JSR INTPAR    ;WERT HOLEN
C716  A024      LDY #$24      ;"$"
C718  84FF      STY PUFFER-1  ;ERSTES ZEICHEN
C71A  A004      LDY #4        ;4 STELLEN
C71C  205DC6    JSR INTHEX    ;UMWANDLUNG
C71F  A9FF      LDA #$FF
C721  A000      LDY #$00      ;STRINGADR.=$00FF
C723  4C87B4    JMP $B487     ;WEITER WIE STR$
C726            ;
C726          BINSTR:
C726  2009C7    JSR INTPAR    ;WERT HOLEN
C729  A025      LDY #$25      ;"%"
C72B  84FF      STY PUFFER-1  ;ERSTES ZEICHEN
C72D  A010      LDY #16       ;16 STELLEN
C72F  2086C6    JSR INTBIN    ;UMWANDLUNG
C732  A9FF      LDA #$FF
C734  A000      LDY #$00      ;STRINGADR.=$00FF
C736  4C87B4    JMP $B487     ;WEITER WIE STR$
C739            ;
```

In Kapitel 2.2.1.5 werden folgende Labels angesprungen:

HEXCONV, wenn im Basictext ein $-Zeichen auftrat
BINCONV, wenn im Basictext ein %-Zeichen auftrat
VAL, wenn die VAL-Funktion aufgerufen wird
HEXSTR, wenn die Funktion HEX$ aufgerufen wird
BINSTR für die Funktion BIN$.

In diesem Abschnitt ist auch noch die kleine Routine INTPAR enthalten, die einen Integer-Parameter in Klammern holt.

Diese Routinen müssen unbedingt verwendet werden, wenn eine der Basic-Erweiterungen $, %, HEX$, BIN$ oder die erweiterte VAL-Funktion verwendet werden sollen. Auf eine genaue Beschreibung soll verzichtet werden, da ja hier nur die Parameter für die oben beschriebenen Routinen gesetzt werden. Lediglich die Routine für die erweiterte VAL-Funktion soll etwas näher erläutert werden.

Zunächst wird der Code für VAL überlesen (JSR CHRGET), dann wird der Parameter ausgewertet (JSR FRMKLA und JSR $B782). Die String-Länge wird in $61 abgelegt und das erste Zeichen des Strings untersucht. Ist es ein $-Zeichen, wird zu HEXVAL verzweigt, ist es ein %-Zeichen, zu BINVAL. Sonst wird einfach die String-Länge in den Akku zurückgeholt und die normale VAL-Funktion angesprungen.

Die kleine Routine VALPAR setzt die Stringlänge um eins zurück und die Stringadresse um eins hoch, da das erste Zeichen ja bereits bekannt ist.

2.2.3 Erweiterte PEEK- und POKE-Funktionen

Die Befehle PEEK und POKE sind der Schlüssel für viele Programmiertricks. Jedoch kann man damit nicht den gesamtem RAM-Bereich nutzen außerdem ist die Behandlung von 16-Bit-Werten umständlich. Um diesen Mangel zu beheben, stellen wir Ihnen im folgenden die nötigen Befehle und Funktionen vor.

Zusammen mit den Zahlkonvertierungen aus Kapitel 2.2.2 bilden sie ein wertvolles Hilfsmittel, auf das Sie beim Testen von Programmen sicher immer zurückgreifen werden.

2.2.3.1 PEEK ins RAM unter ROM (REEK) und PEEK in den Zeichengenerator (CHEEK)

Name	REEK bzw. CHEEK
Funktion	REEK liest immer aus einer Speicherzelle des RAM CHEEK liest im Bereich $D000 bis $DFFF aus dem Zeichengenerator-ROM.

Syntax	REEK(Adresse) bzw. CHEEK(Adresse)
	(keine Befehle sondern Funktionen)
Parameter	Adresse = Wert zwischen 0 und 65535
Beispiel	?PEEK($D000), REEK($D000), CHEEK($D000)
Hilfsroutinen	nur BASIC-ROM
Adressbereich	$C400 - $C436 = 50176 - 50230

```
C400            LADR=$C3       ; FUER LADEADRESSE
C400            SADR=$C1       ; FUER ANFANGSADRESSE
C400            ;
C400  00      PORTMASK:   BYT 0 ;MERKER REEK/CHEEK
C401            ;
C401          REEK:
C401  A9FC      LDA #%11111100 ;LORAM & HIRAM
C403  D002      BNE ERWPEEK    ;UNBED. SPRUNG
C405            ;
C405          CHEEK:
C405  A9FB      LDA #%11111011 ;CHAREN
C407            ;
C407          ERWPEEK:        ;* ERWEITERTES PEEK *
C407  8D00C4    STA PORTMASK ;MERKER SPEICHERN
C40A  20F1AE    JSR FRMKLA   ; PARAMETER HOLEN
C40D  208DAD    JSR CHNUM    ; NUMERISCH PRUEFEN
C410  78        SEI
C411  A514      LDA ADR      ; ADR SICHERN
C413  48        PHA
C414  A515      LDA ADR+1
C416  48        PHA
C417  A501      LDA 1        ; PROZESSOR-PORT SICHERN
C419  48        PHA
C41A  20F7B7    JSR FACADR   ; ADRESSE DES ZU HOLENDEN
C41D  A000      LDY #0       ; BYTES IN ADR LADEN
C41F  A501      LDA 1
C421  2D00C4    AND PORTMASK ; BESTIMMTE ROM-BER.
C424  8501      STA 1        ; AUSSCHALTEN
C426  B114      LDA (ADR),Y  ; BYTE HOLEN
C428  A8        TAY          ; -> Y
C429  68        PLA          ; PROZESSOR-PORT RESTAURIEREN
C42A  8501      STA 1
C42C  68        PLA          ; ADR RESTAURIEREN
C42D  8515      STA ADR+1
C42F  68        PLA
C430  8514      STA ADR
C432  20A2B3    JSR YFLP     ; Y -> FAC
C435  58        CLI          ; INTERRUPTS EIN
C436  60        RTS          ; END ERW.PEEK
C437            ;
```

Beschreibung Da sich die Aufrufe für REEK und CHEEK nur durch andere Schaltung der Prozessorport-Leitungen unterscheiden, wird eine Hilfszelle PORTMASK definiert, die speichert, welche Port-Bits auf 0 gesetzt werden sollen, während die Speicherzelle ausgelesen wird. Die übrige Leseroutine gleicht im wesentlichen der im ROM eingebauten PEEK-Routine; vor dem eigentlichen Lesen des Speicherinhaltes wird lediglich der Prozessorport umbeschrieben. Außerdem muß das Interrupt-Disable-Flag gesetzt werden, bevor dies geschieht.

2.2.3.2 16-Bit-PEEK (DEEK)

Name	DEEK
Funktion	lies 16-Bit-Wert aus Speicherzellenpaar
Syntax	DEEK(Adresse) (Funktion)
Parameter	Adresse = Adresse der ersten Speicherzelle eines Speicherzellenpaares, das einen 16-Bit-Wert in dem Format LOW-Byte, HIGH-Byte enthält.
Beispiel	PRINT DEEK(55)
Hilfsroutinen	FLPINT, INTFLP
Adressbereich	DEEK: $C437 - $C456 = 50231 - 50262 FLPINT: $C5EA - $C5FD = 50666 - 50685 INTFLP: $C5E4 - $C5E9 = 50660 - 50665

```
C437          DEEK:
C437  20F1AE    JSR FRMKLA    ; AUSDRUCK IN () HOLEN
C43A  208DAD    JSR CHNUM     ; NUMERISCH PRUEFEN
C43D  20EAC5    JSR FLPINT    ; -> INTEGER
C440  A562      LDA $62       ; ZEIGER NACH $64,$65
C442  8565      STA $65
C444  A563      LDA $63
C446  8564      STA $64
C448  A000      LDY #0
C44A  B164      LDA ($64),Y   ;1. BYTE
C44C  8563      STA $63       ;(LOW-BYTE)
C44E  C8        INY
C44F  B164      LDA ($64),Y   ;2. BYTE
C451  8562      STA $62       ;(HIGH-BYTE)
C453  20E4C5    JSR INTFLP    ;ERGEBNIS -> FAC
C456  60        RTS           ;ENDE DEEK
C457            ;
```

Beschreibung Mit Hilfe von FRMKLA, CHNUM und FLPINT wird der Adreßparameter in die Speicherzellen $62,$63 geholt, wonach er in die Zellen $64,$65 kopiert wird, die als Zeiger auf das zu lesende Byte dienen. Die beiden zu lesenden Bytes werden nacheinander geholt und in die Zellen $63 und $62 abgelegt und anschließend mit Hilfe von INTFLP nach Fließkomma zurückgewandelt.

2.2.3.3 16-Bit-POKE (DOKE)

Name	DOKE
Funktion	schreibe 16-Bit-Wert in Zellenpaar
Syntax	DOKE Adresse, Wert oder SYS(50263) Adresse,Wert
Parameter	Adresse = Adresse der ersten Speicherzelle eines Speicherzellenpaares, welches eine 16-Bit-Wert in der Reihenfolge L,H aufnehmen soll. Wert = vorzeichenloser ganzzahliger Wert von 0 - 65535
Beispiel	DOKE $D400,2000
Hilfsroutinen	FLPINT
Adressbereich	FLPINT: $C5EA - $C5FD = 50666 - 50685 DOKE: $C457 - $C471 = 50263 - 50289

```
C457          DOKE:
C457  208AAD    JSR FRMNUM    ;ADRESSE HOLEN
C45A  20F7B7    JSR FACADR    ;-> ADR,ADR+1
C45D  20FDAE    JSR CHKCOM    ;KOMMA PRUEFEN
C460  208AAD    JSR FRMNUM    ;WERT HOLEN
C463  20EAC5    JSR FLPINT    ;-> INTEGER
C466  A000      LDY #0
C468  A563      LDA $63       ;LOW-BYTE
C46A  9114      STA (ADR),Y   ;SPEICHERN
C46C  C8        INY
C46D  A562      LDA $62       ;HIGH-BYTE
C46F  9114      STA (ADR),Y   ;SPEICHERN
C471  60        RTS           ;ENDE DOKE
C472            ;
```

Beschreibung Die Adresse wird mit FRMNUM und FACADR in das Zellenpaar ADR,ADR+1 geschafft, der Wert mit Hilfe von FRMNUM und FLPINT nach $63 und $62, woraus er dann in die Zieladressen gespeichert wird.

2.2.3.4 16-Bit-PEEK in RAM unter ROM (DREEK)

Name	DREEK
Funktion	lies 16-Bit-Wert aus RAM-Zellenpaar
Syntax	DREEK (Adresse) (Funktion)
Parameter	Adresse = Adresse der ersten Zelle eines RAM-Zellenpaares, in dem ein 16-Bit-Wert (L,H) abgelegt ist.
Beispiel	PRINT DEEK($A000)
Hilfsroutinen	FLPINT, INTFLP
Adressbereich	FLPINT: $C5EA - $C5FD = 50666 - 50685
	INTFLP: $C5E4 - $C5E9 = 50660 - 50665
	DREEK: $C472 - $C49D = 50290 - 50333

```
C472          DREEK:
C472  20F1AE    JSR FRMKLA    ; ADRESSE IN ()
C475  208DAD    JSR CHNUM     ; NUMERISCH PRUEFEN
C478  20EAC5    JSR FLPINT    ; -> INTEGER
C47B  A562      LDA $62       ; ADR. NACH $64,$65
C47D  8565      STA $65
C47F  A563      LDA $63
C481  8564      STA $64
C483  A000      LDY #0
C485  A501      LDA 1         ; PROZESSOR-PORT SICHERN
C487  48        PHA           ; SICHERN
C488  78        SEI           ; INTERRUPTS AUS
C489  25FC      AND %11111100 ; BASIC,KERNAL,I/O
C48B  8501      STA 1         ; AUSSCHALTEN
C48D  B164      LDA ($64),Y   ; 1. BYTE
C48F  8563      STA $63       ; (LOW-BYTE)
C491  C8        INY
C492  B164      LDA ($64),Y   ; 2. BYTE
C494  8562      STA $62       ; (HIGH-BYTE)
C496  68        PLA           ; PROZESSOR-PORT
C497  8501      STA 1         ; RESTAURIEREN
C499  58        CLI           ; INTERRUPTS EIN
C49A  20E4C5    JSR INTFLP    ; ERGEBNIS -> FAC
C49D  60        RTS           ; ENDE DREEK
C49E            ;
```

Beschreibung Diese Routine stellt eine Kombination von REEK und DEEK dar, so daß die Beschreibung aus den Kapiteln 2.2.3.1 und 2.2.3.2 hervorgeht.

2.2.3.5 POKE in RAM unter I/O (ROKE)

Name	ROKE
Funktion	schreibe 8-Bit-Wert in RAM-Zelle
Syntax	ROKE Adresse, Wert bzw. SYS(50334) Adresse,Wert
Parameter	Adresse = beliebige Adresse zwischen 0 und 65535, in welcher der Byte-Wert abgelegt werden soll Wert = Zahl zwischen 0 und 255
Beispiel	ROKE $D000, 123
Hilfsroutinen	keine
Adressbereich	ROKE: $C49E - $C4B2 = 50334 - 50354

```
C49E          ROKE:
C49E  20EBB7    JSR GETAB     ; ADRESSE UND WERT
C4A1  78        SEI           ; INTERRUPTS AUS
C4A2  A501      LDA 1         ; PROZESSORPORT
C4A4  48        PHA           ; SICHERN
C4A5  29F9      AND #%11111001;CHAREN & HIRAM
C4A7  8501      STA 1
C4A9  A000      LDY #0
C4AB  8A        TXA
C4AC  9114      STA (ADR),Y   ; BYTE SPEICHERN
C4AE  68        PLA           ; PROZESSORPORT
C4AF  8501      STA 1         ; RESTAURIEREN
C4B1  58        CLI           ; INTERRUPTS EIN
C4B2  60        RTS           ; ENDE ROKE
C4B3            ;
```

Beschreibung Diese Routine gleicht im wesentlichen dem Basic-POKE-Befehl, jedoch wird vorher der Prozessorport so umgeschaltet, daß im Bereich $D000 bis $DFFF für Schreibzugriffe RAM erscheint.

2.2.4 Zeichengenerator ins RAM kopieren (CHARKOP)

Name	CHARKOP
Funktion	kopiere Zeichengenerator-ROM in das darunterliegende RAM
Syntax	CHARKOP oder SYS$C4B3 bzw. SYS50355
Parameter	keine

Beispiel

```
CHARKOP : POKE56567,148 : POKE53272,53
POKE648,204 : PRINT CHR$(147)
ROKE $D100,1
```

Hilfsroutinen	keine
Adressbereich	$C4B3 - $C4D7 = 50355 - 50391

```
C4B3            CHARGEN=$D000 ;ADRESSE CHARACTERS
C4B3            ;
C4B3          CHARKOP:
C4B3  78        SEI           ;INTERRUPTS AUS
C4B4  A900      LDA #<CHARGEN ;ZEIGER AUF CHARGEN
C4B6  8514      STA ADR
C4B8  A9D0      LDA #>CHARGEN
C4BA  8515      STA ADR+1
C4BC  A501      LDA 1         ;PROZESSORPORT
C4BE  48        PHA           ;SICHERN
C4BF  29F9      AND #%11111001;CHAREN & HIRAM
C4C1  8501      STA 1
C4C3  A000      LDY #0
C4C5  A210      LDX #16       ;ZAEHLER FUER PAGES
C4C7          CHARKOP1:
C4C7  B114      LDA (ADR),Y   ;BYTE AUS CHARGEN.
C4C9  9114      STA (ADR),Y   ;IN RAM KOPIEREN
C4CB  C8        INY
C4CC  D0F9      BNE CHARKOP1  ;SCHLEIFE
C4CE  E615      INC ADR+1     ;NAECHSTE PAGE
C4D0  CA        DEX           ;PAGE-ZAEHLER
C4D1  D0F4      BNE CHARKOP1  ;SCHLEIFE
C4D3  68        PLA           ;PROZESSORPORT
C4D4  8501      STA 1         ;RESTAURIEREN
C4D6  58        CLI           ;INTERRUPTS EIN
C4D7  60        RTS           ;ENDE CHARKOP
C4D8            ;
```

Beschreibung

Nach Abschalten der Interrupts wird ein Zeiger (ADR, ADR+1) auf den Anfang des Zeichengenerators gerichtet, dann der Prozessorport so geschaltet, daß aus dem Zei-

chengenerator gelesen werden kann und in das darunter liegende RAM geschrieben werden kann und anschließend die gesamten 4 KByte kopiert.

Zur Anwendung: dieser Befehl kopiert lediglich das Zeichengenerator-ROM in das darunter liegende RAM, so daß die übrigen Arbeiten, wie das Umbesetzen der Zeiger und des Videokontroller-Adressbereiches im Basic-Programm durchgeführt werden müssen. Im Beispiel-Programm ist dies bereits erledigt. Notwendig ist unter anderem die Verlegung des Video-RAM's in einen Bereich zwischen $C000 und $CFFF, da der gesamte Videokontroller-Bereich in die letzten 16 KB verlegt werden muß. Sie können den Zeichengenerator auch ohne weiteres in einen anderen RAM-Bereich legen, jedoch müssen Sie dann den CHARKOP-Befehl mit entsprechenden SWAP-Befehlen umrahmen.

2.2.5 Austausch von RAM-Bereichen (SWAP)

Wir wollen zunächst das eigentliche Maschinenprogramm zum Vertauschen zweier Speicherbereiche beschreiben und dann das Rahmenprogramm, das die Parameter aus dem Basic-Text übernimmt und für die Routine SW aufbereitet.

```
Name            SW
Funktion        Vertauschen von RAM-Bereichen
Aufruf          JSR SW   bzw. JSR $C4DC (50396)
Parameter       $C3, $C4 = Startadresse des ersten Spei-
                           cherbereichs (L,H)
                $C1, $C2 = Startadresse des zweiten Spei-
                           cherbereichs (L,H)
                ZAHL, ZAHL+1 = Anzahl der zu vertauschenden
                               Bytes (L,H); Adresse von
                               ZAHL ist hier $C4D8 (50392)
Hilfsroutinen   keine
Adressbereich   $C4D8 - $C513  =  50392 - 50451
```

```
C4D8            ;
C4D8  0000    ZAHL:      WOR 0 ;ANZAHL BYTES
C4DA  00      ZAEHL:     BYT 0 ;PAGE-ZAEHLER
C4DB  00      PORTALT:   BYT 0 ;FUER PORT
C4DC            ;
```

```
C4DC          SW:
C4DC  A501      LDA 1           ;PROZESSORPORT
C4DE  8DDBC4    STA PORTALT     ;SICHERN
C4E1  08        PHP             ;FLAGS SICHERN
C4E2  78        SEI             ;INTERRUPTS AUS
C4E3  29FC      AND #%11111100;LORAM & HIRAM
C4E5  8501      STA 1
C4E7  A000      LDY #0
C4E9  8CDAC4    STY ZAEHL       ;ZAEHLER AUF NULL
C4EC          S1:
C4EC  B1C3      LDA (LADR),Y    ;1. BYTE HOLEN
C4EE  AA        TAX             ;SICHERN
C4EF  B1C1      LDA (SADR),Y    ;2. BYTE HOLEN
C4F1  91C3      STA (LADR),Y    ;SPEICHERN
C4F3  8A        TXA
C4F4  91C1      STA (SADR),Y    ;1. BYTE SPEICHERN
C4F6  C8        INY
C4F7  D007      BNE S2          ;KEINE PAGE-GRENZE
C4F9  E6C4      INC LADR+1      ;HIGH-BYTES ER-
C4FB  E6C2      INC SADR+1      ;HOEHEN
C4FD  EEDAC4    INC ZAEHL       ;PAGE-ZAEHLER
C500          S2:
C500  ADDAC4    LDA ZAEHL       ;PAGEZAHL
C503  CDD9C4    CMP ZAHL+1      ;MIT SOLL VERGL.
C506  90E4      BCC S1          ;SCHLEIFE
C508  CCD8C4    CPY ZAHL        ;LOW-BYTES VERGL.
C50B  90DF      BCC S1          ;SCHLEIFE
C50D  ADDBC4    LDA PORTALT     ;PROZESSORPORT
C510  8501      STA 1           ;RESTAURIEREN
C512  28        PLP             ;FLAGS RESTAURIEREN
C513  60        RTS             ;ENDE SW
C514            ;
```

Beschreibung Während des Vertauschvorganges werden die Interrupts verhindert und der Prozessorport so geschaltet, daß nur RAM sichtbar ist. Dadurch können auch die RAM-Bereiche unter dem Basic und unter dem KERNAL benützt werden. Als Zeiger auf Start- und Zieladresse werden die Zellenpaare LADR, LADR+1=$C3, $C4 und SADR,SADR+1 = $C1,$C2 benützt.

Name SWAP
Funktion Vertauschen von RAM-Bereichen mit Bereitstellung der Parameter
Syntax SWAP ADR1, ADR2, ZAHL oder
SYS($C518) ADR1,ADR2,ZAHL

bzw. SWAP ADR1,ADR2 oder
SYS($C518) ADR1,ADR2
bzw. SWAP oder
SYS($C518)

Parameter ADR1 = Startadresse des 1. Speicherbereichs
ADR2 = Startadresse des 2. Speicherbereichs
ZAHL = Anzahl der zu vertauschenden Bytes

Werden Parameter weggelassen, so gelten dafür die beim letzten SWAP-Befehl eingestellten Werte. Wurde seit dem Laden der Basic-Erweiterungen noch kein SWAP-Befehl gegeben, so lautet die Voreinstellung: ADR1=$D000, ADR2=$D000,ZAHL=0

Beispiel SWAP 1024, 40960, 1000 : PAUSE1 : SWAP

Hilfsroutinen SW

Adressbereich SWAP: $C514 - $C574 = 50452 - 50548
SW: $C4D8 - $C513 = 50392 - 50451

```
C514  00D0    SWADR1:  WOR $D000
C516  00D0    SWADR2:  WOR $D000
C518             ;
C518           SWAP:
C518  207900     JSR CHRGOT     ;LETZTES ZEICHEN
C51B  F040       BEQ SWAP1      ;TRENNZEICHEN
C51D  208AAD     JSR FRMNUM     ;1.ADRESSE HOLEN
C520  20F7B7     JSR FACADR     ;-> ADR,ADR+1
C523  A514       LDA ADR
C525  8D14C5     STA SWADR1     ;-> SWADR1,SWADR1+1
C528  A515       LDA ADR+1
C52A  8D15C5     STA SWADR1+1
C52D  207900     JSR CHRGOT     ;LETZTES ZEICHEN
C530  F02B       BEQ SWAP1      ;TRENNZEICHEN
C532  20FDAE     JSR CHKCOM     ;KOMMA PRUEFEN
C535  208AAD     JSR FRMNUM     ;2.ADRESSE HOLEN
C538  20F7B7     JSR FACADR     ;-> ADR,ADR+1
C53B  A514       LDA ADR
C53D  8D16C5     STA SWADR2     ;-> SWADR2,SWADR2+1
C540  A515       LDA ADR+1
C542  8D17C5     STA SWADR2+1
C545  207900     JSR CHRGOT     ;LETZTES ZEICHEN
C548  F013       BEQ SWAP1      ;TRENNZEICHEN
C54A  20FDAE     JSR CHKCOM     ;KOMMA PRUEFEN
C54D  208AAD     JSR FRMNUM     ;ANZAHL HOLEN
C550  20F7B7     JSR FACADR     ;-> ADR,ADR+1
C553  A514       LDA ADR
C555  8DD8C4     STA ZAHL       ;-> ZAHL,ZAHL+1
C558  A515       LDA ADR+1
C55A  8DD9C4     STA ZAHL+1
```

```
C55D          SWAP1:
C55D  AD14C5    LDA SWADR1     ;SWADR1 -> LADR
C560  85C3      STA LADR
C562  AD15C5    LDA SWADR1+1
C565  85C4      STA LADR+1
C567  AD16C5    LDA SWADR2     ;SWADR2 -> SADR
C56A  85C1      STA SADR
C56C  AD17C5    LDA SWADR2+1
C56F  85C2      STA SADR+1
C571  20DCC4    JSR SW         ;SW-ROUTINE
C574  60        RTS            ;ENDE SWAP
C575            ;
```

Beschreibung Die Routine CHRGOT setzt das Zero-Flag, wenn ein Trennzeichen, also Doppelpunkt oder Zeilenende im Basic-Text folgt. Dadurch wird überprüft, ob nach dem SWAP-Befehl noch Parameter folgen. Ist dies der Fall, werden sie entsprechend in die vorgesehenen Speicherzellen übernommen und am Schluß die Routine SW aufgerufen.

2.2.6 Laden ohne Veränderung der Basic-Zeiger

Name	LAD
Funktion	Setzen der Parameter und Aufruf der KERNAL-Routine LOAD
Syntax	LAD ZADR, Name, GA, SA oder SYS($C577), ZADR, Name, GA, SA
Parameter	ZADR : Zieladresse für den zu ladenden Bereich, wenn die Sekundäradresse weggelassen wird oder 0 ist. Name : Name der zu ladenden Datei GA : Geräteadresse SA : Sekundäradresse Ist die Sekundäradresse 1, so wird die Zieladresse nicht durch ZADR bestimmt, sondern aus der Datei gelesen.
Beispiel	LAD $E000,"Grafik",8 LAD,"PROFI.OBJ",8,1
Hilfsroutinen	keine
Adressbereich	$C575 - $C5A6 = 50549 - 50598

```
C575  0000    ZADR:   WOR 0     ;ZIELADRESSE
C577          LAD:
C577  208AAD    JSR FRMNUM      ;ZIELADRESSE HOLEN
C57A  20F7B7    JSR FACADR      ;-> ADR,ADR+1
C57D  A514      LDA ADR
C57F  8D75C5    STA ZADR        ;-> ZADR,ZADR+1
C582  A515      LDA ADR+1
C584  8D76C5    STA ZADR+1
C587  20FDAE    JSR CHKCOM      ;KOMMA PRUEFEN
C58A  20D4E1    JSR PARLOSA     ;PARAMETER HOLEN
C58D  A900      LDA #0          ;LOAD (NICHT VERIFY)
C58F  AE75C5    LDX ZADR        ;ALTERNATIVE LADE-
C592  AC76C5    LDY ZADR+1      ;ADRESSE
C595  20D5FF    JSR LOAD        ;LOAD-ROUTINE
C598  B008      BCS LFEHLER     ;FEHLER PRUEFEN
C59A  20B7FF    JSR READST      ;STATUS LESEN
C59D  29BF      AND #$BF        ;EOF AUSBLENDEN
C59F  D001      BNE LFEHLER
C5A1  60        RTS             ;ENDE LAD
C5A2            ;
C5A2          LFEHLER:
C5A2  A21D      LDX #$1D        ;NR.F. 'LOAD ERROR'
C5A4  4C37A4    JMP FEHLER      ;FEHLER AUSGEBEN
C5A7            ;
```

Beschreibung — Die Zieladresse wird aus dem Basic-Text geholt und in die Zellen ZADR, ZADR+1 zwischengespeichert. Dann werden die Parameter für den LOAD-Befehl geholt und anschließend die LOAD-Routine aufgerufen, wonach zu einem Fehler verzweigt wird, wenn das Carry-Flag gesetzt ist oder im Status-Byte ein Fehler angezeigt wird.

Bemerkung — Mit diesem Befehl kann der Bereich $D000 bis $DFFF nicht direkt beschrieben werden, jedoch können Sie durch eine geeignete Folge von SWAP und LAD-Befehlen dies bewerkstelligen.

2.2.7 Speichern von RAM-Bereichen

Name — SPEI
Funktion — Setzen der Parameter für und Aufrufen der

KERNAL-Routine SAVE

Syntax SPEI SADR,EADR,NAME,GA,SA oder
SYS(50599) SADR,EADR,NAME,GA,SA

Parameter SADR = Startadresse des abzuspeichernden Bereichs
EDADR = Endadresse + 1 des abzuspeichernden Bereichs
Name, GA, SA : wie beim Basic-Befehl SAVE

Beispiel SPEI 1024,2023,"Bildschirm",8

Hilfsroutinen keine

Adressbereich $C5A7 - $C5E3 = 50599 - 50659

```
C5A7          SPEI:
C5A7  208AAD    JSR FRMNUM     ;STARTADRESSE HOLEN
C5AA  20F7B7    JSR FACADR     ;-> ADR,ADR+1
C5AD  A514      LDA ADR
C5AF  85C1      STA SADR       ;-> SADR,SADR+1
C5B1  A515      LDA ADR+1
C5B3  85C2      STA SADR+1
C5B5  20FDAE    JSR CHKCOM     ;KOMMA PRUEFEN
C5B8  208AAD    JSR FRMNUM     ;ENDADRESSE HOLEN
C5BB  20F7B7    JSR FACADR     ;-> ADR,ADR+1
C5BE  A514      LDA ADR
C5C0  8D75C5    STA ZADR       ;-> ZADR,ZADR+1
C5C3  A515      LDA ADR+1
C5C5  8D76C5    STA ZADR+1
C5C8  20FDAE    JSR CHKCOM     ;KOMMA PRUEFEN
C5CB  20D4E1    JSR PARLOSA    ;PARAMETER HOLEN
C5CE  A501      LDA 1          ;PROZESSORPORT
C5D0  48        PHA            ;SICHERN
C5D1  29FE      AND #%11111110;LORAM
C5D3  8501      STA 1
C5D5  A9C1      LDA #SADR      ;ZEIGER AUF SADR
C5D7  AE75C5    LDX ZADR       ;ENDADRESSE
C5DA  AC76C5    LDY ZADR+1
C5DD  20D8FF    JSR SAVE       ;SAVE-ROUTINE
C5E0  68        PLA            ;PROZESSORPORT
C5E1  8501      STA 1          ;RESTAURIEREN
C5E3  60        RTS            ;ENDE SPEI
C5E4            ;
```

Beschreibung Die Startadresse wird in die Zero-Page-Zellen SADR, SADR+1 geschrieben, die Endadresse in die Zellen ZADR, ZADR+1. Vor dem Aufruf der SAVE-Routine wird noch das Basic-ROM ausgeschaltet, damit auch in diesem Bereich das RAM abgespeichert werden kann. Eine Behandlung von eventuell auftauchenden Fehlern während der SAVE-Routine erfolgt hier nicht.

Bemerkung Das Abspeichern von Bereichen nach $D000 muß mit einer entsprechenden Kombination von SWAP-Befehlen und dem SPEI-Befehl erfolgen.

2.2.8 Index-Funktion

Name	INDEX
Funktion	Funktion zur Bestimmung der Position einer Zeichenreihe in einer anderen
Syntax	INDEX(Stringausruck1, Stringausdruck2) (Funktion, kein Befehl)
Parameter	Stringausdruck1 = String, der gesucht werden soll Stringausdruck2 = String, in dem gesucht werden soll
Beispiel	A$="WE" : PRINT INDEX(A$,"KUWEIT")
Hilfsroutinen	keine
Adressbereich	$C739 - $C784 = 51001 - 51076

```
C739            STRADR1=$A7    ;ADRESSE 1. STRING
C739            STRADR=$22     ;ADRESSE 2. STRING
C739            STRLEN1=$A9    ;LAENGE 1. STRING
C739            STRLEN=$AA     ;LAENGE 2. STRING
C739            ANZ=$61        ;ANZAHL DURCHLAEUFE
C739            ;
C739          INDEX:
C739  20FAAE    JSR CHKKLA     ;KLAMMER AUF PRUEFEN
C73C  209EAD    JSR FRMEVL     ;AUSDRUCK HOLEN
C73F  20A3B6    JSR FRESTR     ;STRINGPARAMETER
C742  85A9      STA STRLEN1    ;LAENGE SPEICHERN
C744  A522      LDA STRADR     ;ADRESSE SICHERN
C746  85A7      STA STRADR1
C748  A523      LDA STRADR+1   ;HIGH-BYTES
C74A  85A8      STA STRADR1+1
C74C  20FDAE    JSR CHKCOM     ;KOMMA PRUEFEN
C74F  209EAD    JSR FRMEVL
C752  20A3B6    JSR FRESTR
C755  85AA      STA STRLEN
C757  20F7AE    JSR CHKKLZ     ;KLAMMER ZU PRUEFEN
C75A  A201      LDX #1         ;ERGEBNIS = 1
C75C  38        SEC
C75D  A5AA      LDA STRLEN
C75F  E5A9      SBC STRLEN1    ;2.LAENGE-1.LAENGE
C761  901A      BCC INDEX8     ;1. STRING LAENGER
```

```
C763  6900      ADC #0        ; +1
C765  8561      STA ANZ       ;ALS SCHLEIFENZAEHLER
C767          INDEX1:
C767  A4A9      LDY STRLEN1   ;VERGL. BEGINNT HINTEN
C769          INDEX2:
C769  88        DEY           ;NAECHSTES ZEICHEN
C76A  3013      BMI INDEX9    ;ALLE ZEICHEN VERGL.
C76C  B1A7      LDA (STRADR1),Y
C76E  D122      CMP (STRADR),Y ;1 ZEICHEN VERGL.
C770  F0F7      BEQ INDEX2    ;WENN OK, NAECHSTES Z.
C772  E622      INC STRADR    ;NAECHSTE POSITION
C774  D002      BNE INDEX3    ;KEIN UEBERTRAG
C776  E623      INC STRADR+1
C778          INDEX3:
C778  E8        INX           ;ERGEBNIS ERHOEHEN
C779  C661      DEC ANZ       ;LAUFVARIABLE
C77B  D0EA      BNE INDEX1    ;SCHLEIFENENDE N. ERR.
C77D          INDEX8:
C77D  A200      LDX #0        ;ERGEBNIS = 0
C77F          INDEX9:
C77F  8A        TXA
C780  A8        TAY
C781  20A2B3    JSR YFLP      ;NACH FLIESSK. WANDELN
C784  60        RTS           ;ENDE INDEX
C785            ;
```

Beschreibung

Die Stringparameter werden mit Hilfe von FRMEVL und FRESTR ausgewertet und in Zellen in der Zero-Page zwischengespeichert.

Die beiden Strings werden jeweils mit FRMEVL und FRESTR eingelesen. Die Deskriptoren verteilen sich auf die Speicherzellen STRLEN1, STRADR1, STRADR1+1 für den ersten String und STRLEN, STRADR, STRADR+1 für den zweiten String. Die Anzahl der Schleifendurchläufe berechnet sich aus STRLEN - STRLEN1 + 1 und wird in ANZ gespeichert. Für ein negatives oder verschwindendes Ergebnis wird sofort zum Ende verzweigt, wo die gefundene Position auf Null gesetzt wird. Die äußere Schleife arbeitet mit zwei Laufvariablen, die gegeneinander laufen (ANZ zählt rückwärts, das X-Register aufwärts); abgebrochen wird die Schleife, wenn ANZ Null ist oder der String gefunden wurde. Das X-Register enthält zunächst die gefundene Position, die mit Hilfe der Routine YFLP in einen Fließkommawert gewandelt wird.

Bemerkung Die Funktion wird immer funktionieren, bis auf einen speziellen Fall, der durch die Stringverwaltung des Rechners gegeben ist: im Direktmodus bei Verwendung von zwei Konstanten als Stringparameter. Dieser Fall hat keine konkrete Anwendung, taucht aber auf, wenn Sie die Funktion testen wollen.

2.2.9 Floppy-Routinen

In diesem Kapitel wollen wir eine Gruppe von Befehlen und Routinen besprechen, die zur Behandlung der Disketten-Station nützlich sind. Es ist zum einen das Einlesen des Disk-Status zum anderen die Möglichkeit, aus Text- oder Programmdateien im Direktmodus zu lesen und dadurch unmittelbar Dateien ansehen zu können.

Vorher müssen wir noch einige Hilfsroutinen beschreiben, die sich u.a. mit der Stringvariablenbehandlung des Basic befassen.

Ab $9E00 (40448) befindet sich eine Sprungtabelle auf die benötigten Befehle DISKS, GETPZ, GETTZ und VIEW. Sofern Sie die Befehle als Basic-Erweiterung nutzen möchten, muß diese auch ins RAM geladen werden.

```
9E00            ;** SPRUNGTABELLE **
9E00  4CAD9E    JMP DISKS
9E03  4C459F    JMP GETTZ
9E06  4CEE9E    JMP GETPZ
9E09  4C6B9F    JMP VIEW
9E0C            ;****************************
9E0C            ;* VARIABLE UND HILFSZELLEN *
9E0C            ;****************************
9E0C  0000    PZNR:  WOR 0 ; ZEILENNUMMER
9E0E            ;DER PROGRAMMZEILE
9E0E  0000    PZVP:  WOR 0 ; VORWAERTS-
9E10            ;ZEIGER DER PROGRAMMZEILE
9E10            ;
```

2.2.9.1 Hilfsroutinen

```
Name            STRPARL
Funktionen      schreibt Länge einer String-Variablen
                in Zelle $AA, Startadresse in $22, $23
Parameter       $47, $48 = Variablen-Adresse
Adressbereich   $9E10 - $9E20  =  40464 - 40480
```

```
9E10            ;** STRINGPARAMETER LESEN **
9E10          STRPARL:
9E10  A002      LDY #2
9E12  B147      LDA (VARADL),Y
9E14  8523      STA STRADR+1
9E16  88        DEY
9E17  B147      LDA (VARADL),Y
9E19  8522      STA STRADR
9E1B  88        DEY
9E1C  B147      LDA (VARADL),Y
9E1E  85AA      STA STRLEN
9E20  60        RTS
9E21            ;
```

```
Name            STRPARS
Funktionen      schreibe Länge des Strings aus Zelle
                $AA in Variable und Startadresse aus
                Zelle $22, $23 in Variable
Parameter       $47, $48 = Variablen-Adresse
Adressbereich   $9E21 - $9E31  =  40481 - 40497
```

```
9E21            ;** STRINGPARAMETER SCHREIBEN **
9E21          STRPARS:
9E21  A002      LDY #2
9E23  A523      LDA STRADR+1
9E25  9147      STA (VARADL),Y
9E27  88        DEY
9E28  A522      LDA STRADR
9E2A  9147      STA (VARADL),Y
9E2C  88        DEY
9E2D  A5AA      LDA STRLEN
9E2F  9147      STA (VARADL),Y
9E31  60        RTS
9E32            ;
```

Name STRZUW
Funktionen weise String aus Eingabepuffer Platz im Stringbereich zu
Parameter ZLEN = Länge des Strings im Puffer
$AA = Länge des alten Strings
STRADR, STRADR+1 = Startadresse des alten Strings
Eingabepuffer liegt ab $A000
Adressbereich $9E32 - $9E5B = 40498 - 40538

```
9E32            ;** STRING-ZUWEISUNG **
9E32          STRZUW:
9E32  A5AB      LDA ZLEN        ;ZEILENLAENGE
9E34  C5AA      CMP STRLEN      ;ALTE STRINGLAENGE
9E36  F009      BEQ STRZU1      ;GLEICH
9E38  9007      BCC STRZU1      ;KLEINER
9E3A  20F4B4    JSR NEUSTR      ;NEUEN STRING
9E3D  8622      STX STRADR      ;NEUE STRINGADRESSE
9E3F  8423      STY STRADR+1
9E41          STRZU1:
9E41  85AA      STA STRLEN      ;STRINGL.=ZEILENL.
9E43  A8        TAY             ;ALS ZAEHLER
9E44  F014      BEQ STRZU3      ;ZEILENLAENGE NULL
9E46  A501      LDA 1           ;PROZESSORPORT
9E48  48        PHA             ;MERKEN
9E49  29FE      AND #%11111110  ;LORAM
9E4B  8501      STA 1
9E4D          STRZU2:
9E4D  88        DEY
9E4E  B900A0    LDA INPUFF,Y    ;ZEICHEN AUS PUFFER
9E51  9122      STA (STRADR),Y  ;IN STRING UEBERTR.
9E53  C000      CPY #0          ;ZAEHLER = 0
9E55  D0F6      BNE STRZU2      ;SCHLEIFE
9E57  68        PLA             ;PROZESSORPORT
9E58  8501      STA 1           ;RESTAURIEREN
9E5A          STRZU3:
9E5A  60        RTS             ;ENDE STRZUW
```

Beschreibung Ist der neue String kleiner oder gleich lang wie der alte String, kann der gleiche Bereich zur Speicherung verwendet werden. Auf alle Fälle wird der String aus dem Puffer in den Stringbereich übertragen.

Dekodieren von Basic-Code

Name PZCODE
Funktionen Wandle Basic-Code entsprechend der Befehlstabelle in Klartext um
Aufruf JSR PZCODE
Parameter Akkumulator : Basic-Code

```
                ZLEN        : Zeiger in Puffer
                INPUFF...   : Puffer für Klartext
                Akkumulator : letztes umgewandeltes Zeichen
Hilfsroutinen   keine
Adressbereich   $9E5B - $9E94  =  40539 - 40596

9E5B            ;****************************
9E5B            ;* BASIC-CODE UMWANDELN     *
9E5B            ;****************************
9E5B          PZCODE:
9E5B  AA        TAX          ;CODE MERKEN
9E5C  102A      BPL PZCODE6  ;KEIN BASIC-TOKEN
9E5E  C9FF      CMP #$FF     ;CODE FUER PI
9E60  F026      BEQ PZCODE6  ;JA
9E62  240F      BIT $0F      ;HOCHKOMMA-MODUS
9E64  3022      BMI PZCODE6  ;JA
9E66  38        SEC
9E67  E97F      SBC #$7F     ;127 ABZIEHEN
9E69  AA        TAX          ;ALS ZAEHLER
9E6A  A0FF      LDY #$FF
9E6C          PZCODE2:
9E6C  CA        DEX          ;CODEZAEHLER
9E6D  F008      BEQ PZCODE4  ;CODE GEFUNDEN
9E6F          PZCODE3:
9E6F  C8        INY          ;ZEIGER IN KTAB
9E70  B99EA0    LDA KTAB,Y   ;ZEICHEN AUS KTAB
9E73  10FA      BPL PZCODE3  ;IM WORT
9E75  30F5      BMI PZCODE2  ;WORTENDE
9E77          PZCODE4:
9E77  C8        INY          ;NAECHSTES ZEICHEN
9E78  B99EA0    LDA KTAB,Y   ;VON CODE-WORT
9E7B  3009      BMI PZCODE5  ;WORTENDE
9E7D  A6AB      LDX ZLEN     ;ZEILENLAENGE
9E7F  E6AB      INC ZLEN     ;ERHOEHEN
9E81  9D00A0    STA INPUFF,X;ZEICHEN ABLEGEN
9E84  D0F1      BNE PZCODE4  ;UNBED. SPRUNG
9E86          PZCODE5:
9E86  297F      AND #$7F     ;MSB AUSBLENDEN
9E88          PZCODE6:
9E88  C922      CMP #$22     ;HOCHKOMMA
9E8A  D008      BNE PZCODE7  ;NEIN
9E8C  A50F      LDA $0F      ;HOCHKOMMAFLAG
9E8E  49FF      EOR #$FF     ;UMDREHEN
9E90  850F      STA $0F
9E92  A922      LDA #$22     ;KODE VON HOCHKOMMA
9E94          PZCODE7:
9E94  60        RTS          ;ENDE PZCODE
9E95            ;
```

Beschreibung Handelt es sich nicht um einen Basic-Code (Akku kleiner 128), wird nur auf Anführungszeichen geprüft und dann das Unterprogramm verlassen, ebenso wenn der Hochkomma-Modus (in Zelle $0F gespeichert) aktiv ist. Im anderen Fall wird das zur Codenummer gehörige Befehlswort aus der im Basic-Interpreter enthaltenen Befehlswort-tabelle herausgesucht und bis auf das letzte Zeichen in den Puffer übertragen. Das letzte Zeichen wird im Akku zurückgegeben.

Dateinummer bestimmen

Name	GETFILN
Funktion	Hole Dateinummer aus Basic-Text und setze Eingabekanal
Aufruf	JSR GETFILN
Parameter	Zelle $13 = Nummer der Eingabedatei
Hilfsroutinen	keine
Adressbereich	$9E95 - $9EAC = 40597 - 40620

```
9E95          GETFILN:
9E95  A200      LDX #0        ;DEFAULT-WERT
9E97  207900    JSR CHRGOT    ;LETZTES ZEICHEN
9E9A  C923      CMP #$23      ;"#"
9E9C  D00C      BNE GETFILN9  ;NEIN
9E9E  207300    JSR CHRGET    ;"#" UEBERLESEN
9EA1  209EB7    JSR GETBYT    ;DATEINUMMER HOLEN
9EA4  201EE1    JSR BCHKIN    ;ALS EINGABEKANAL
9EA7  20FDAE    JSR CHKCOM    ;KOMMA UEBERLESEN
9EAA          GETFILN9:
9EAA  8613      STX AKTIO     ;DATEINUMMER MERKEN
9EAC  60        RTS           ;ENDE GETFILN
9EAD            ;
```

Beschreibung Folgt dem Basic-Text kein '#', so wird statt der Dateinummer eine Null in der Zelle $13 zurückgegeben. Im anderen Fall wird die Dateinummer mit GETBYT eingelesen, der Eingabekanal auf die angegebene Datei gelegt und das folgende Komma überlesen.

2.2.9.2 Disk-Status lesen

Name	DISKS
Funktion	lies Diskstatuszeile von Floppy in Variable DS$ ein; unmittelbare Bildschirmausgabe im Direktmodus
Syntax	DISK bzw. SYS40621
Parameter	Variable DS$ = Diskstatus
Beispiel	DISK : ? DS$
Hilfsroutinen	STPARL, STPARS, STRZUW
Adressbereich	STRPARL : $9E10 - $9E20 = 40464 - 40480
	STRPARS : $9E21 - $9E31 = 40481 - 40497
	STRZUW : $9E32 - $9E5A = 40498 - 40538
	DISKS : $9EAD - $9EED = 40621 - 40685

```
9EAD          DISKS:
9EAD  A900      LDA #0
9EAF  85AB      STA ZLEN      ;ZEILENLAENGE=0
9EB1  A908      LDA #8        ;GERAET NR. 8
9EB3  20B4FF    JSR TALK      ;TALK-KOMMANDO
9EB6  A96F      LDA # 15+$60  ;SEK. ADR. 15
9EB8  2096FF    JSR TKSA      ;SEK.ADR. SENDEN
9EBB          DISKS1:
9EBB  20A5FF    JSR ACPTR     ;BYTE LESEN
9EBE  A4AB      LDY ZLEN
9EC0  9900A0    STA INPUFF,Y  ;ABLEGEN
9EC3  E6AB      INC ZLEN      ;ZEILENLAENGE ERH.
9EC5  249D      BIT $9D       ;MODUS-FLAG
9EC7  1003      BPL DISKS2    ;PROGRAMMMODUS
9EC9  20D2FF    JSR CHROUT    ;ZEICHEN AUSGEBEN
9ECC          DISKS2:
9ECC  C90D      CMP #$0D      ;CARRIAGE-RETURN
9ECE  D0EB      BNE DISKS1    ;SCHLEIFE
9ED0  20ABFF    JSR UNTALK    ;UNTALK SENDEN
9ED3  C6AB      DEC ZLEN      ;ZEILENLAENGE - 1
9ED5  A944      LDA #$44      ;"D"
9ED7  8545      STA VARNAM    ;VARIABLENNAME
9ED9  A9D3      LDA #$53+$80  ;"S" + STRINGFLAG
9EDB  8546      STA VARNAM+1
9EDD  A9FF      LDA #$FF      ;STRINGFLAG
9EDF  850D      STA $0D       ;SETZEN
9EE1  20E7B0    JSR NAMSUC    ;VARIABLE SUCHEN
9EE4  20109E    JSR STRPARL   ;ALTE PARAM. LESEN
9EE7  20329E    JSR STRZUW    ;STRING UEBERTRAGEN
9EEA  20219E    JSR STRPARS   ;NEUE PARAM. SETZEN
9EED  60        RTS           ;ENDE DISKS
9EEE            ;
```

Beschreibung Zunächst wird ein TALK-Kommando mit Geräteadresse 8 und Sekundäradresse 15 über den seriellen Bus geschickt und anschließend in einer Schleife der String eingelesen, bis ein Carriage-Return auftritt. Dabei wird jedes Zeichen in einem Zwischenspeicher abgelegt und anschließend die Variable DS$ gesucht und der String an diese Variable übergeben.

2.2.9.3 Programmzeile lesen

Name GETPZ
Funktion lies eine Programmzeile aus Floppy-Datei in Variable ein
Syntax LESP#DN, VAR oder SYS(40686)#DN, VAR
Parameter DN = logische Dateinummer der zu lesenden Programmdatei
VAR = Stringvariable, an die die eingelesene Zeile zugewiesen wird
$9E0C,$9E0D = eingelesene Zeilennummer (L,H)
$9E0E,$9E0F = eingelesener Vorwärtszeiger
Beispiel
```
10 OPEN 2,8,2,"Programm"
20 GET#2,A1$, A2$: REM Startadr. überl.
30 LESP#2,A$:PRINTA$:IFDEEK($9E0E)THEN30
```
Hilfsroutinen GETFILN, STRPARL, STRZUW, STRPARS, PZCODE
Adressbereich
```
STRPARL : $9E10 - $9E20  =  40464 - 40480
STRPARS : $9E21 - $9E31  =  40481 - 40497
STRZUW  : $9E32 - $9E5A  =  40498 - 40538
PZCODE  : $9E5B - $9E94  =  40539 - 40596
GETFILN : $9E95 - $9EAC  =  40597 - 40620
GETPZ   : $9EEE - $9F44  =  40686 - 40772
```

```
9EEE            ;*****************************
9EEE            ;* EINE PROGRAMMZEILE LESEN *
9EEE            ;*****************************
9EEE          GETPZ:
9EEE  A900      LDA #0
9EF0  850F      STA $0F       ;HOCHKOMMAMODUS AUS
```

```
9EF2  85AB     STA ZLEN      ;ZEILENLAENGE = 0
9EF4  20959E   JSR GETFILN   ;DATEINR. FESTLEGEN
9EF7  208BB0   JSR VARSUC    ;VARIABLE SUCHEN
9EFA  20109E   JSR STRPARL   ;ALTE PARAMETER LESEN
9EFD  2024E1   JSR BGETIN    ;BYTE LESEN
9F00  8D0E9E   STA PZVP      ;VORWAERTSZEIGER LOW
9F03  2024E1   JSR BGETIN
9F06  8D0F9E   STA PZVP+1    ;VORW.Z. HIGH
9F09  A8       TAY           ;FLAGS SETZEN
9F0A  F027     BEQ GETPZ2    ;VORW.Z. = 0
9F0C  2024E1   JSR BGETIN    ;BYTE LESEN
9F0F  8D0C9E   STA PZNR      ;ZEILENNUMMER LOW
9F12  2024E1   JSR BGETIN
9F15  8D0D9E   STA PZNR+1    ;ZEILENNR. HIGH
9F18         GETPZ1:
9F18  2024E1   JSR BGETIN    ;ZEICHEN LESEN
9F1B  205B9E   JSR PZCODE    ;UMWANDELN
9F1E  A4AB     LDY ZLEN      ;ZEILENLAENGE
9F20  9900A0   STA INPUFF,Y  ;ZEICHEN ABLEGEN
9F23  A8       TAY           ;FLAGS SETZEN
9F24  F00D     BEQ GETPZ2    ;ZEILENENDE
9F26  A590     LDA STATUS    ;STATUS-BYTE LESEN
9F28  D009     BNE GETPZ2    ;UNGLEICH NULL
9F2A  E6AB     INC ZLEN      ;ZEILENLAENGE ERH.
9F2C  D0EA     BNE GETPZ1    ;SCHLEIFE
9F2E  A217     LDX #23       ;STRING TOO LONG
9F30  4C37A4   JMP FEHLER    ;FEHLER AUSGEBEN
9F33           ;
9F33         GETPZ2:
9F33  20329E   JSR STRZUW    ;STRING ZUWEISEN
9F36  20219E   JSR STRPARS   ;STRINGPARAM. SETZEN
9F39  A613     LDX AKTIO     ;EINGABEDATEI
9F3B  F007     BEQ GETPZE    ;TASTATUR
9F3D  20CCFF   JSR CLRCHN    ;KANAELE SCHLIESSEN
9F40  A200     LDX #0
9F42  8613     STX AKTIO     ;MERKER RUECKSETZEN
9F44         GETPZE:
9F44  60       RTS           ;ENDE GETPZ / GETTZ
9F45           ;
```

Beschreibung Nach Auswahl der Eingabe-Datei und Suchen der Variablen werden aus der Datei zunächst der Vorwärtszeiger und die Zeilennummer eingelesen und gespeichert. Dann wird in einer Schleife jeweils ein Zeichen gelesen, dekodiert und im Puffer abgelegt. Am Ende der Schleife wird der Variablen der Ergebnis-String zugewiesen und der Eingabekanal zurückgesetzt. Endekriterium einer Schleife ist das Auftauchen des 0-Codes.

Bemerkung Da der Basic-Eingabepuffer nicht benutzt wird, kann dieses Kommando im Direktmodus gegeben werden.

2.2.9.4 Textzeile lesen

Name GETTZ
Funktion lies eine Textzeile in Variable (ähnlich INPUT)
Syntax LEST #DN,VAR oder SYS(40773)#DN,VAR
LEST VAR oder SYS(40773) VAR
Achtung: vor LEST vom Bildschirm muß POKE 144,0 eingegeben werden
Parameter DN = logische Nummer der Eingabedatei
VAR = Variable, die einen eingelesenen String aufnehmen soll
Beispiel 10 OPEN 2,8,2,"SEQDATEI"
20 LESP#2,A$:PRINTA$:IFST=0THEN20
Hilfsroutinen GETFILN, STRPARL, STRZUW, STRPARS
Adressbereich STRPARL : $9E10 - $9E20 = 40464 - 40480
STRPARS : $9E21 - $9E31 = 40481 - 40497
STRZUW : $9E32 - $9E5A = 40498 - 40538
GETFILN : $9E95 - $9EAC = 40597 - 40620
GETTZ : $9F33 - $9F6A = 40755 - 40810

```
9F45          GETTZ:
9F45  A900      LDA #0
9F47  85AB      STA ZLEN      ;ZEILENLAENGE = 0
9F49  20959E    JSR GETFILN   ;DATEINR. FESTLEGEN
9F4C  208BB0    JSR VARSUC    ;VARIABLE SUCHEN
9F4F  20109E    JSR STRPARL   ;ALTE PARAM. LESEN
9F52          GETTZ1:
9F52  2012E1    JSR BASIN     ;ZEICHEN LESEN
9F55  A4AB      LDY ZLEN      ;ZEILENLAENGE
9F57  9900A0    STA INPUFF,Y  ;ZEICHEN ABLEGEN
9F5A  C90D      CMP #$0D      ;ZEILENENDE
9F5C  F0D5      BEQ GETPZ2    ;JA, WEITER BEI GETPZ2
9F5E  A590      LDA STATUS    ;STATUS LESEN
9F60  D0D1      BNE GETPZ2    ;NICHT NULL
9F62  E6AB      INC ZLEN      ;ZEILENLAENGE ERH.
9F64  D0EC      BNE GETTZ1    ;SCHLEIFE
9F66  A217      LDX #23       ;STRING TOO LONG
9F68  4C37A4    JMP FEHLER    ;FEHLER AUSGEBEN
9F6B            ;
```

Beschreibung Nach Auswählen der Eingabedatei und Suchen der Variablen wird in einer Schleife ein Zeichen nach dem anderen aus der Datei eingelesen und in den Puffer abgelegt, bis ein Carriage-Return auftritt. Die Abschlußbehandlung ist die gleiche wie im vorigen Kapitel.

2.2.9.5 Directory / Programmdatei anzeigen (VIEW)

Name	VIEW
Funktion	Liste Programmdatei direkt von Floppy auf Bildschirm
Syntax	VIEW"$",GA oder SYS(40825)"$",GA VIEW PN$,GA oder SYS(40825)PN$,GA
Parameter	PN$ = Name der zu listenden Programmdatei GA = Geräteadresse
Beispiel	VIEW"$:A*",8 : REM zeigt alle Dateien, die mit 'A' beginnen
Hilfsroutinen	PZCODE
Adressbereich	PZCODE : $9E5B - $9E94 = 40539 - 40596 VIEW : $9F6B - $9FEC = 40814 - 40941

```
9F6B          VIEW:
9F6B  20D4E1    JSR PARLOSA ;PARAMETER HOLEN
9F6E  A960      LDA #0 +$60 ;SEK.ADR. 0
9F70  85B9      STA $B9     ;MERKEN
9F72  A5B7      LDA $B7     ;FILENAMENLAENGE
9F74  F073      BEQ VIEWE   ;NULL
9F76  20D5F3    JSR $F3D5   ;DATEI OEFFNEN
9F79  A5BA      LDA $BA     ;GERAETEADR.
9F7B  20B4FF    JSR TALK    ;TALK-KOMMANDO
9F7E  A5B9      LDA $B9     ;SEK. ADR.
9F80  2096FF    JSR TKSA    ;SENDEN
9F83  A003      LDY #3      ;3*2 BYTE
9F85  8461      STY $61     ;ZAEHLER
9F87          VIEW2:
9F87  20A5FF    JSR ACPTR   ;1. BYTE HOLEN
9F8A  8562      STA $62     ;MERKEN
9F8C  A490      LDY STATUS  ;STATUS TESTEN
9F8E  D059      BNE VIEWE   ;NICHT NULL
9F90  20A5FF    JSR ACPTR   ;2. BYTE HOLEN
9F93  A490      LDY STATUS  ;STATUS TESTEN
9F95  D052      BNE VIEWE   ;NICHT NULL
9F97  C661      DEC $61     ;ZAEHLER VERMINDERN
9F99  D0EC      BNE VIEW2   ;SCHLEIFE
```

```
9F9B  A662     LDX $62      ;AKKU/X = HIGH/LOW
9F9D  20CDBD   JSR ADROUT   ;VON ZEILENNUMMER
9FA0  203FAB   JSR SPOUT    ;UND LEERZ. AUSGEBEN
9FA3  A900     LDA #0
9FA5  85AB     STA ZLEN     ;ZEILENLAENGE = 0
9FA7  850F     STA $0F      ;HOCHKOMMAMODUS AUS
9FA9         VIEW3:
9FA9  20A5FF   JSR ACPTR    ;BYTE LESEN
9FAC  A490     LDY STATUS   ;STATUS TESTEN
9FAE  D039     BNE VIEWE    ;NICHT NULL
9FB0  AA       TAX          ;FLAGS SETZEN
9FB1  F00C     BEQ VIEW4    ;ZEILENENDE
9FB3  205B9E   JSR PZCODE   ;KODE WANDELN
9FB6  A6AB     LDX ZLEN     ;ZEILENLAENGE
9FB8  E6AB     INC ZLEN     ;ERHOEHEN
9FBA  9D00A0   STA INPUFF,X ;ZEICHEN ABLEGEN
9FBD  D0EA     BNE VIEW3    ;SCHLEIFE
9FBF         VIEW4:
9FBF  A501     LDA 1        ;PROZESSORPORT
9FC1  48       PHA          ;SICHERN
9FC2  29FE     AND #%11111110 ;LORAM
9FC4  8501     STA 1
9FC6  A000     LDY #0
9FC8         VIEW5:
9FC8  8461     STY $61       ;ZAEHLER
9FCA  E661     INC $61       ;ERHOEHEN
9FCC  B900A0   LDA INPUFF,Y ;ZEICHEN AUS PUFFER
9FCF  20D2FF   JSR CHROUT    ;AUSGEBEN
9FD2  A461     LDY $61
9FD4  C4AB     CPY ZLEN      ;ZEILENLAENGE
9FD6  D0F0     BNE VIEW5     ;SCHLEIFE
9FD8  68       PLA           ;PROZESSORSTATUS
9FD9  8501     STA 1         ;RESTAURIEREN
9FDB  A90D     LDA #$0D      ;ZEILENVORSCHUB
9FDD  20D2FF   JSR CHROUT    ;AUSGEBEN
9FE0  A002     LDY #2        ;2*2 BYTE
9FE2  8461     STY $61
9FE4  20E1FF   JSR STOP      ;STOPTASTE ABFRAGEN
9FE7  D09E     BNE VIEW2     ;SCHLEIFE
9FE9         VIEWE:
9FE9  2042F6   JSR $F642     ;CLOSE
9FEC  60       RTS           ;ENDE VIEW
9FED           ;
```

Beschreibung Zunächst werden die Dateiparameter geholt und die Datei direkt geöffnet. Dann werden 3 x 2 Byte gelesen und davon die letzten beiden als Zeilennummer ausgegeben sowie die Zeile in den Puffer übernommen, bis ein '0-Code' auftritt. Anschließend wird die

Zeile aus dem Puffer ausgegeben und die nächste Zeile geholt, bis der Status von null verschieden ist. Zum Abschluß wird die Datei direkt geschlossen.

2.2.10 Integer-Division

Name	IDIV
Funktion	bilde ganzzahligen Quotienten und Rest von zwei 16 Bit-Zahlen
Aufruf	JSR IDIV bzw. SYS(36608)
Parameter	DIVIDEND,DIVIDEND+1 = Dividend (L,H) und Quotient (L,H) DIVISOR,DIVISOR+1 = Divisor (L,H) REST, REST+1 = Rest (L,H)
Beispiel	siehe Kapitel 2.2.1.8
Hilfsroutinen	keine
Adressbreich	$8F00 - $BF4A = 36608 - 36682

```
8F00          IDIV:             ;GANZZAHLIGE DIVISION
8F00  A200      LDX #$00
8F02  8E498F    STX REST
8F05  8E4A8F    STX REST+1        ;REST ANNULLIEREN
8F08  AD458F    LDA DIVISOR
8F0B  0D468F    ORA DIVISOR+1
8F0E  F02E      BEQ DIVNULL       ;WENN DIVISOR = 0
8F10  A210      LDX #16           ;BITZAEHLER AUF 16
8F12          IDIV1:
8F12  2E478F    ROL DIVIDEND      ;CARRY VON SBC IN ER-
8F15  2E488F    ROL DIVIDEND+1    ;GEBNIS UND NAECHSTES
8F18  2E498F    ROL REST          ;BIT DES DIVIDENDEN
8F1B  2E4A8F    ROL REST+1        ;IN REST SCHIEBEN
8F1E  38        SEC               ;FUER SUBTRAKTION
8F1F  AD498F    LDA REST          ;DIVISOR VON REST
8F22  ED458F    SBC DIVISOR       ;ABZIEHEN
8F25  A8        TAY               ;AKKU SICHERN
8F26  AD4A8F    LDA REST+1        ;HIGH-BYTES
8F29  ED468F    SBC DIVISOR+1     ;ABZIEHEN
8F2C  9006      BCC IDIV2         ;WENN REST ZU KLEIN,
8F2E  8C498F    STY REST          ;DANN NICHT SPEICHERN
8F31  8D4A8F    STA REST+1
8F34          IDIV2:
8F34  CA        DEX               ;BITZAEHLER
8F35  D0DB      BNE IDIV1         ;SCHLEIFE
8F37  2E478F    ROL DIVIDEND      ;CARRY IN ERGEBNIS
8F3A  2E488F    ROL DIVIDEND+1    ;HINEINROTIEREN
8F3D  60        RTS               ;FERTIG
```

```
8F3E          DIVNULL:
8F3E  68        PLA             ;STACK BEREINIGEN
8F3F  68        PLA
8F40  A214      LDX #20         ;NR. FUER 'DIV. BY 0'
8F42  6C0003    JMP ($0300)     ;FEHLERMELDUNG
8F45            ;
8F45  0000    DIVISOR:    WOR 0 ;PLATZ FUER DIVISOR
8F47  0000    DIVIDEND:   WOR 0 ;DIVIDEND UND ERGEBNIS
8F49  0000    REST:       WOR 0 ;PLATZ FUER REST
```

Beschreibung Ist der Divisor gleich Null, so wird eine Fehlermeldung ausgegeben. Im Vergleich zu anderen Codierungen des Divisions-Algorithmus ist hier die Analogie zum Verfahren, das man mit Bleistift und Papier durchführt, relativ deutlich zu sehen. Das X-Register dient als Bit-Zähler, das Y-Register nur zum Zwischenspeichern des Akkus.

2.2.11 Zeitbehandlung mit dem CIA

Im folgenden wollen wir kurz auf die Möglichkeiten des CIA bezüglich Timer und Uhr eingehen.

2.2.11.1 Pause mit Timer

Name	PAUSE
Funktion	Halte Programm bis zu 655.35 Sekunden an
Syntax	PAUSE D oder SYS51083 D
Parameter	D = Dauer / Anzahl Sekunden (auf 1/100 genau angebbar)
Beispiel	PAUSE 0.25
Hilfsroutinen	keine
Adressbereich	$C786 - $C7CC = 51078 - 51148

```
C786          FK100:          ;FLIESSK.KONSTANTE 100
C786  87        BYT $87
C787  48        BYT $48
C788  00        BYT $00
C789  00        BYT $00
C78A  00        BYT $00
C78B            ;
C78B            CIA2=$DD00  ;ADRESSE DES CIA2
C78B            ;
```

```
C78B          PAUSE:
C78B  A97A      LDA #<9850  ;TEILRATE A LOW-BYTE
C78D  8D04DD    STA CIA2+4
C790  A926      LDA #>9850  ;TEILRATE A HIGH-BYTE
C792  8D05DD    STA CIA2+5
C795  208AAD    JSR FRMNUM  ;ZEITPARAMETER HOLEN
C798  A986      LDA #<FK100
C79A  A0C7      LDY #>FK100
C79C  2028BA    JSR FMALK   ;MAL KONSTANTE 100
C79F  20F7B7    JSR FACADR  ;-> ADR,ADR+1
C7A2  A514      LDA ADR
C7A4  8D06DD    STA CIA2+6  ;TEILRATE B LOW-BYTE
C7A7  A515      LDA ADR+1
C7A9  8D07DD    STA CIA2+7  ;TEILRATE B HIGH-BYTE
C7AC  AD0EDD    LDA CIA2+14 ;KONTROLLREG. A
C7AF  29D1      AND #%11010001
C7B1  0911      ORA #%00010001
C7B3  8D0EDD    STA CIA2+14
C7B6  AD0FDD    LDA CIA2+15 ;KONTROLLREG. B
C7B9  29D1      AND #%11010001
C7BB  0959      ORA #%01011001
C7BD  8D0FDD    STA CIA2+15
C7C0          PAULOOP:
C7C0  AD0DDD    LDA CIA2+13 ;INTERRUPT-FLAG-REG.
C7C3  2902      AND #%00000010
C7C5  D005      BNE PAUEND
C7C7  20E1FF    JSR STOP    ;STOPTASTE ABFRAGEN
C7CA  D0F4      BNE PAULOOP ;SCHLEIFE
C7CC          PAUEND:
C7CC  60        RTS         ;ENDE PAUSE
```

Beschreibung Es werden die beiden Timer im CIA2 benutzt. Timer A wird so programmiert, daß seine Unterlauffrequenz genau 100 Hertz beträgt, Timer B arbeitet im one-shot-Betrieb. Bevor der Parameter in High-Byte und Low-Byte zerlegt werden kann, muß er mit 100 multipliziert werden. Dann werden die Timer gestartet und gewartet, bis im Timer B ein Unterlauf auftritt, was im Interrupt-Kontroll-Register angezeigt wird.

2.2.11.2 Uhrzeit setzen

Name SETTIME
Funktion Setze Uhrzeit oder Alarmzeit im CIA

Syntax SETTIME Uhrnr, Zeit oder
SYS (51968) Uhrnr, Zeit

Parameter Uhrnr = Nummer der zu stellenden Uhr
1 = Uhrzeit im CIA1
2 = Uhrzeit im CIA2
3 = Alarmzeit im CIA1
4 = Alarmzeit im CIA2
Zeit = Zeichenreihe der Länge 10 im Format
"HH:MM:SS:Z"

Beispiel SETTIME 1,"12:10:00:0"

Hilfsroutinen keine

Adressbereich SETTIME: $CB00 - $CB8A = 51968 - 52106

```
CB00            ;*****************************
CB00            ;* UHRZEIT IM CIA SETZEN     *
CB00            ;*****************************
CB00            CLOCKNR=$AB
CB00            ;
CB00          SETTIME:
CB00  209EB7    JSR GETBYT      ;UHRNUMMER HOLEN
CB03  CA        DEX             ;EINS ABZIEHEN
CB04  86AB      STX CLOCKNR     ;= CLOCKNR.
CB06  20FDAE    JSR CHKCOM      ;KOMMA
CB09  209EAD    JSR FRMEVL      ;STRING HOLEN
CB0C  20A3B6    JSR FRESTR
CB0F  C90A      CMP #10         ;10 ZEICHEN
CB11  F003      BEQ SETTIM1     ;OK
CB13  4C48B2    JMP ILLQERR     ;FEHLER
CB16          SETTIM1:
CB16  A5AB      LDA CLOCKNR     ;0,1,2,3
CB18  4A        LSR A           ;HALBIEREN
CB19  48        PHA             ;AKKU SICHERN
CB1A  A900      LDA #0
CB1C  69DC      ADC #$DC        ;HIGH-BYTE VON CIA1
CB1E  85A8      STA $A8
CB20  A200      LDX #0          ;X=0 !
CB22  86A7      STX $A7
CB24  A00E      LDY #14
CB26  B1A7      LDA ($A7),Y     ;KONTROLLREG. A
CB28  0980      ORA #%10000000 ;50HZ-FLAG SETZEN
CB2A  91A7      STA ($A7),Y
CB2C  C8        INY
CB2D  68        PLA             ;CLOCKNR. / 2
CB2E  4A        LSR A           ;0-BIT INS CARRY
CB2F  08        PHP             ;CARRY SICHERN
CB30  B1A7      LDA ($A7),Y     ;KONTROLLREG. B
CB32  2A        ROL A
CB33  28        PLP             ;CARRY VON OBEN
CB34  6A        ROR A
CB35  91A7      STA ($A7),Y     ;ALARMZEIT/UHRZEIT
```

```
CB37  2065CB    JSR AUSW2       ;2 ZIFFERN AUSW.
CB3A  F8        SED             ;BCD-ARITHMETIK
CB3B  C912      CMP #$12        ;12 UHR
CB3D  F006      BEQ SETT2       ;PM-FLAG AUTOM.
CB3F  9004      BCC SETT2       ;VOR 12 UHR
CB41  E912      SBC #$12        ;12 ABZIEHEN
CB43          SETTPM:
CB43  0980      ORA #%10000000  ;PM-FLAG
CB45          SETT2:
CB45  D8        CLD             ;BCD AUS
CB46  A00B      LDY #11
CB48  91A7      STA ($A7),Y     ;STUNDEN SETZEN
CB4A  2065CB    JSR AUSW2       ;2 ZIFFERN AUSW.
CB4D  88        DEY
CB4E  91A7      STA ($A7),Y     ;MINUTEN SETZEN
CB50  2065CB    JSR AUSW2       ;2 ZIFFERN AUSW.
CB53  88        DEY
CB54  91A7      STA ($A7),Y     ;SEKUNDEN SETZEN
CB56  A122      LDA (STRADR,X)  ;1/10 SEKUNDEN
CB58  290F      AND #$0F
CB5A  88        DEY
CB5B  91A7      STA ($A7),Y     ;SETZEN
CB5D  60        RTS
CB5E            ;
CB5E          INCSA:
CB5E  E622      INC STRADR      ;STRADR UM EINS
CB60  D002      BNE INCSA1      ;ERHOEHEN
CB62  E623      INC STRADR+1
CB64          INCSA1:
CB64  60        RTS
CB65            ;
CB65            ;
CB65          AUSW2:            ;2 ZIFFERN UND ":"
CB65  A122      LDA (STRADR,X)  ;NAECHSTES ZEICHEN
CB67  0A        ASL A           ;4 BIT NACH LINKS
CB68  0A        ASL A
CB69  0A        ASL A
CB6A  0A        ASL A
CB6B  85A9      STA $A9         ;MERKEN
CB6D  205ECB    JSR INCSA
CB70  A122      LDA (STRADR,X)  ;NAECHSTES ZEICHEN
CB72  290F      AND #$0F        ;4 BIT AUSBLENDEN
CB74  05A9      ORA $A9         ;MIT 1.ZIFFER ODERN
CB76  85A9      STA $A9         ;MERKEN
CB78  205ECB    JSR INCSA
CB7B  A122      LDA (STRADR,X)  ;NAECHSTES ZEICHEN
CB7D  C93A      CMP #$3A        ;":"
CB7F  F005      BEQ AUSW21      ;OK
```

```
CB81  68       PLA             ;STACK BEREINIGEN
CB82  68       PLA
CB83  4C48B2   JMP ILLQERR     ;FEHLER
CB86         AUSW21:
CB86  A5A9     LDA $A9         ;ERGEBNIS IN AKKU
CB88  4C5ECB   JMP INCSA       ;ENDE UPRO AUSW2
CB8B           ;
```

Beschreibung Die Uhr-Nummer wird mit GETBYT geholt und um eins vermindert in die Zelle CLOCKNR gespeichert. In den Zellen $A7,$A8 wird ein Zeiger auf die Basisadresse des CIA errichtet. Das High-Byte des Zeigers wird um eins erhöht, wenn der Wert in CLOCKNR ungerade ist, so daß in diesem Fall der Zeiger auf CIA2 zeigt. Im Kontrollregister 1 des CIA wird das Bit 7 gesetzt (50 Hertz). Das Bit 1 von CLOCKNR wird in das Bit 7 des Kontrollregisters 2 des CIA übertragen, womit zwischen Uhrzeit und Alarmzeit gewählt wird. Zwei Hilfsroutinen erleichtern die weitere Verarbeitung: INCSA erhöht den Zeiger auf den String (STRADR,STRADR+1) um eins, AUSW2 wertet zwei Zeichen des Strings aus und prüft, ob ein Doppelpunkt folgt. Damit die Uhr wie im 24-Stunden-Format arbeiten kann, muß von Stundenwert 12 abgezogen werden, wenn dieser größer gleich 12 ist und statt dessen das Nachmittags-Flag gesetzt werden. Im CIA ist die Darstellung von 12 Uhr Mittag nicht eindeutig, man kann sie darstellen als 12 Uhr Vormittag oder 0 Uhr Nachmittags.

2.2.11.3 Uhrzeit lesen

Name	TIME
Funktion	liest Zeit aus CIA-Uhr
Syntax	TIME bzw. TIME2 (Funktion)
Parameter	folgt nach dem Wort TIME eine Ziffer, so gibt diese die Nummer des CIA an, aus dem die Uhrzeit gelesen werden soll

Beispiel PRINT TIME,TIME2
Hilfsroutinen keine
Adressbereich $CB8B - $CBF8 = 52107 - 52216

```
CB8B            ;****************************
CB8B            ;* UHRZEIT IM CIA LESEN     *
CB8B            ;****************************
CB8B          TIME:
CB8B  A201      LDX #1        ;DEFAULTNR. = 1
CB8D  207900    JSR CHRGOT    ;FOLGT ZIFFER
CB90  B003      BCS TIME1     ;NEIN
CB92  209EB7    JSR GETBYT    ;UHRNUMMER HOLEN
CB95          TIME1:
CB95  8A        TXA
CB96  18        CLC
CB97  69DB      ADC #$DC-1    ;UHRNUMMER-1+$DC
CB99  85A8      STA $A8       ;$A7,$A8 AUF CIA-BASIS
CB9B  A900      LDA #0
CB9D  85A7      STA $A7
CB9F  85AB      STA $AB       ;ZAEHLER AUF 0
CBA1  A90A      LDA #10       ;10 ZEICHEN
CBA3  20F4B4    JSR NEUSTR    ;STRING EINRICHTEN
CBA6  8561      STA $61       ;STRINGLAENGE
CBA8  8662      STX $62       ;STRINGADR. LOW
CBAA  8463      STY $63       ;STRINGADR. HIGH
CBAC  A00B      LDY #11
CBAE  B1A7      LDA ($A7),Y ;STUNDEN
CBB0  F8        SED           ;BCD-ARITHMETIK
CBB1  08        PHP           ;FLAG RETTEN
CBB2  297F      AND #%01111111 ;OHNE PM-FLAG
CBB4  C912      CMP #$12      ;12 UHR
CBB6  D002      BNE TIME3
CBB8  A900      LDA #$00      ;12 UHR AM = 0 UHR
CBBA          TIME3:
CBBA  28        PLP           ;FLAG HOLEN
CBBB  1003      BPL TIME4     ;AM
CBBD  18        CLC
CBBE  6912      ADC #$12      ;+ 12 STUNDEN
CBC0          TIME4:
CBC0  D8        CLD           ;BCD AUS
CBC1  20EBCB    JSR TIMEUD    ;-> STRING
CBC4  A00A      LDY #10
CBC6  B1A7      LDA ($A7),Y ;MINUTEN
CBC8  20EBCB    JSR TIMEUD    ;-> STRING
CBCB  A009      LDY #9
CBCD  B1A7      LDA ($A7),Y ;SEKUNDEN
CBCF  20EBCB    JSR TIMEUD    ;-> STRING
CBD2  A008      LDY #8
CBD4  B1A7      LDA ($A7),Y ;1/10 SEKUNDEN
```

```
CBD6  20E0CB   JSR TIMEU2  ;-> STRING
CBD9  4CCAB4   JMP PUSHSTR ;STRING AUF STAPEL
CBDC           ;
CBDC         TIMEU1:       ;HIGH-NYBBLE -> STRING
CBDC  4A       LSR A       ;4 BIT NACH RECHTS
CBDD  4A       LSR A
CBDE  4A       LSR A
CBDF  4A       LSR A
CBE0         TIMEU2:       ;LOW-NYBBLE -> STRING
CBE0  290F     AND #$0F    ;4 BIT MASKIEREN
CBE2  0930     ORA #$30    ;ASC-BITS FUER ZIFFER
CBE4         TIMEU3:       ;AKKU -> STRING
CBE4  A4AB     LDY $AB     ;ZAEHLER IM STRING
CBE6  9162     STA ($62),Y ;AKKU EINTRAGEN
CBE8  E6AB     INC $AB     ;ZAEHLER ERHOEHEN
CBEA  60       RTS
CBEB           ;
CBEB         TIMEUD:       ;BYTE EINTRAGEN
CBEB  48       PHA         ;MERKEN
CBEC  20DCCB   JSR TIMEU1  ;HIGH-NYBBLE
CBEF  68       PLA
CBF0  20E0CB   JSR TIMEU2  ;LOW-NYBBLE
CBF3  A93A     LDA #$3A    ;":"
CBF5  20E4CB   JSR TIMEU3  ;EINTRAGEN
CBF8  60       RTS
```

Beschreibung Folgt eine Ziffer (Carry gleich Null nach CHRGOT), wird die Uhrnummer mit GETBYT geholt. Auf alle Fälle wird in den Zellen $A7,$A8 ein Zeiger auf den CIA errichtet. Anschließend wird ein String für 10 Zeichen eingerichtet. Zur Darstellung der Zeilen im 24-Stunden-Format muß 12 addiert werden, wenn das Nachmittags-Flag gesetzt ist. Zur Vereinfachung der Verarbeitung dienen die Hilfsroutinen TIMEU1, TIMEU2, TIMEU3, die jeweils ein Zeichen in den String eintragen und TIMEUD, die gleich zwei Zeichen und den Doppelpunkt in den String einträgt.

2.2.12 Key-Click

Die folgende kleine Routine läßt in der zweiten Stimme des SID jedesmal einen Klickton erklingen, wenn eine neue Taste gedrückt wird. Zur Aktivierung dieser Routine muß zum

einen die Lautstärke (Adresse $D418) auf einen beliebigen Wert gesetzt werden und außerdem die Routine im Interrupt-Manager aktiviert werden mit POKE$C809, PEEK($C809) OR 2.

Wenn Sie den Interrupt-Manager nicht benützen, können Sie die Routine aufrufen, indem Sie den Interrupt-Vektor auf die Routine richten und das RTS am Schluß der Routine durch einen Sprung auf die Standard-IRQ-Routine ersetzen.

Name	KEYCLICK
Funktion	Klickton bei jedem neuen Tastendruck
Aktivierung	POKE$D418,L : POKE$C809,PEEK($C809) OR 2
Parameter	L = Lautstärke
Hilfsroutinen	Interrupt-Manager
Adressbereich	$C000 bis $C02F = 49152 bis 49199

```
ADR.  OBJ     * QUELLTEXT

C000            %ORG $C000
C000            SID=$D400     ;BASISADRESSE SID
C000          KEYCLICK:
C000  A5CB      LDA $CB       ;GEDR. TASTE
C002  CD2FC0    CMP TALT      ;MIT ALTERT TASTE VGL.
C005  F024      BEQ KEYEND    ;GLEICH, DANN FERTIG
C007  C940      CMP #64       ;KEINE TASTE
C009  F020      BEQ KEYEND    ;KEINE, DANN FERTIG
C00B  A200      LDX #$00      ;
C00D  8E07D4    STX SID+7     ;OSZ. 2 FREQL = 0
C010  8E0BD4    STX SID+11    ;WELLENFORM = 0
C013  A204      LDX #$04
C015  8E0DD4    STX SID+13    ;SUSTAIN=0, RELEASE=4
C018  A28C      LDX #$8C
C01A  8E08D4    STX SID+8     ;FREQH = 140
C01D  A210      LDX #$10
C01F  8E0CD4    STX SID+12    ;ATTACK=1, DECAY=0
C022  A211      LDX #$11
C024  8E0BD4    STX SID+11    ;WELLENFORM DREICK
C027  CA        DEX
C028  8E0BD4    STX SID+11    ;KEYBIT AUS
C02B          KEYEND:
C02B  8D2FC0    STA TALT      ;TASTE MERKEN
C02E  60        RTS           ;FERTIG
C02F            ;
C02F  40      TALT:  BYT 64 ;ALTE TASTE
```

Beschreibung Die momentan gedrückte Taste ist in der Adresse $CB zu finden; sie wird mit den zuletzt gespeicherten Wert verglichen; bei Gleichheit wird die Routine sofort beendet.

Im anderen Fall werden die Register für die zweite Stimme des SID für einen Klickton gesetzt.

2.2.13 Grafikbefehle

Eine der hervorstechenden Eigenschaften des Commodore 64 ist die hochauflösende Grafik, die jedoch von Basic aus nicht einfach und vor allem sehr langsam zu bedienen ist. Verwendet man reine Basic-Routinen zur Grafikverarbeitung, so liegt meistens der Grafikspeicher ab Zelle 8192, wodurch nur etwa 6 KByte für Basic-Programme zur Verfügung stehen. Die im folgenden beschriebenen Erweiterungen legen den Grafikspeicher ab $E000 ab, also unter das KERNAL-ROM, wo er überhaupt keinen sonstigen Platz wegnimmt. Das Video-RAM, also der Platz für die ersten beiden Farben, wird nach $CC00 gelegt.

Mit den Befehlen in diesem Kapitel können Sie die Grafik einschalten, wieder einschalten, löschen, umfärben, ausschalten, einen Punkt setzen und ein Textzeichen in die Grafik positionieren. Damit haben Sie die entscheidenden Zeit- und Speicherplatzprobleme für weitere Basic-Grafik-Routinen gelöst. Aus Platzgründen wollen wir keine weiteren Grafik-Routinen in Maschinensprache beschreiben. Im Anhang finden Sie jedoch Hinweise, wie Sie mit Hilfe des Basic-Loaders oder des Hex-Listings weitere Grafikbefehle realisieren.

Zu Beginn des Grafik-Moduls steht eine Sprungtabelle auf sämtliche Labels, die als Befehle benützt werden können. Danach folgen die Platzreservierungen für Fließkomma-, Word- und Byte-Variable, sowie eine Hilfsroutine, die angesprungen wird, wenn in dem Moment, wo das KERNAL ausgeschaltet ist, die RESTORE-Taste gedrückt wird. Für die Grafik-Routinen sind die folgenden Zeilen (Adreßbereich $9400 - $9469 = 37888 - 37993) unbedingt zu übernehmen.

```
9400            AX     =$A4    ;KOORDINATEN
9400            AY     =$A6    ;ANFANGSPUNKT
9400            EX     =$A7    ;KOORDINATEN
9400            EY     =$A9    ;ENDPUNKT
9400            PBYTE =$AB     ;HILFSZ. FUER PUNKS
9400            PRDL   =$57    ;PRODUKT BZW. DIVIDEND
9400            PRDH=PRDL+1
9400            FAK1L=$59      ;1.FAKTOR BZW. DIVISOR
9400            FAK1H=FAK1L+1
9400            FAK2=$5B       ;2.FAKTOR
9400            QUOTL=$5C      ;QUOTIENT
9400            QUOTH=QUOTL+1
9400            Z=$FD          ; ZEIGER (HILFSVAR.)
9400            GANFH=$E0      ;ANF. GRAPHIK-RAM HIGH
9400            GENDH=$FF      ;ENDE GRAPHIK-RAM HIGH
9400            VANFANG=$CC00;ANFANG VIDEO-RAM
9400            FARBRAM=$D800;ANFANG FARBRAM
9400            VIC=$D000      ;BASISADR. VIDEO-CONTR.
9400            INTL=$65       ;LOW-BYTE INT(FAC)
9400            INTH=$64       ;HIGH-BYTE INT(FAC)
9400            ;
9400            F1     =$B9BC ;FLIESSK.KONST. 1
9400            F05    =$BF11 ;FLIESSK.KONST. 0.5
9400            F2PI   =$E309 ;FLIESSK.KONST. 2*PI
9400            ;
9400            ILQERR=$B248 ; ILL. QUANTITY ERROR
9400            DIZERR=$BB8A ; ZERODIVIDE ERROR
9400            ;
9400            FLDFACK=$BBA2; KONSTANTE NACH FAC
9400            FPLUSK=$B867 ; FAC = FAC+KONSTANTE
9400            FSIN=$E26B     ; FAC = SIN(FAC)
9400            FCOS=$E264     ; FAC = COS(FAC)
9400            FCHS=$BFD4     ; FAC = -FAC
9400            FKMINUS=$B850; FAC = KONSTANTE-FAC
9400            ;FMALK=$BA28   ; FAC = FAC*KONSTANTE
9400            FKDIV=$BB0F   ; FAC = KONSTANTE/FAC
9400            FSGN=$BC2B     ;  A = SGN(FAC)
9400            FLPSGI=$BC9B ; FAC NACH INT
9400            ;
9400            ;          ** SPRUNGLEISTE **
9400  4C6A94    JMP GREIN
9403  4C0695    JMP GRAUS
9406  4C9D94    JMP LOESCH
9409  4CBC94    JMP FARBE
940C  4C7494    JMP GRAFEIN
940F  4C2795    JMP PUNKT
9412  4C3396    JMP CHAR
9415  4CD496    JMP LINIE
```

```
9418  4C5298    JMP RECHT
941B  4C0099    JMP BLOCK
941E  4C6A99    JMP ELLIPSE
9421  4C499A    JMP RADIUS
9424  4CD19A    JMP ZEICHEN
9427  4C7F9B    JMP SCROLL
942A  4CBE9C    JMP GRAFSPEI
942D  4C5E9D    JMP GRAFLAD
9430  4C129D    JMP GRAFMIS
9433            ;
9433            ;       *** FLIESSKOMMAVARIABLE ***
9433            ;
9433          RX:             ;RADIUS X-RICHTUNG
9437  00        %DUP 5, BYT 0
9438          RY:             ;RADIUS Y-RICHTUNG
943C  00        %DUP 5, BYT 0
943D          LAUF:           ;LAUFVARIABLE
9441  00        %DUP 5, BYT 0
9442          STEP:           ;SCHRITTWEITE
9446  00        %DUP 5, BYT 0
9447          WINK:           ;WINKEL
944B  00        %DUP 5, BYT 0
944C            ;
944C            ;BYTE- UND WORD-VARAIBLE:
944C            ;
944C  0000    XDIFF:     WOR 0 ; DIFFERENZ X-KOORD.
944E  00      YDISGN:    BYT 0 ; SGN. DIFF. Y-KOORD.
944F  00      YDIABS:    BYT 0 ; ABS. DIFF. Y-KOORD.
9450  0000    XLAUF:    WOR 0  ; LAUFVARIABLE
9452  00      YLAUF:    BYT 0  ; LAUFVARIABLE
9453  0000    OX:    WOR 0     ; KOORDINATEN
9455  00      OY:    BYT 0     ;  OBERER PUNKT
9456  0000    UX:    WOR 0     ; KOORDINATEN
9458  00      UY:    BYT 0     ;  UNTERER PUNKT
9459  0000    KX:    WOR 0     ; KOORDINATEN
945B  00      KY:    BYT 0     ;  KREISMITTELPUNKT
945C  00      PFARBE:    BYT 0 ;PUNKTFARBE:
945D  00      MULTI:     BYT 0 ;MULTI-COLOUR-MODUS
945E            ;
945E          NMIHILF:     ;** NMI-HILFSROUTINE **
945E            ;
945E  48        PHA             ;AKKU SICHERN
945F  A501      LDA 1
9461  0907      ORA #%00000111 ;STANDARDWERTE
9463  8501      STA 1           ;IN PROZESSORPORT
9465  68        PLA
9466  58        CLI             ;INTERRUPTS ERMOEGL.
9467  6CFAFF    JMP ($FFFA)     ;SPRUNG UEBER ROM-
946A            ;               ;NMI-VEKTOR
```

2.2.13.1 Grafik einschalten

Hier werden vier kleine Routinen vorgestellt, die die Befehle GRLOE (Grafik löschen), FARBE (Farbe definieren und ändern), GRAFEIN (Grafik wieder einschalten) und GREIN (Grafik einschalten und Grafik löschen) darstellen. Der letzte der vier Befehle ist einfach eine Kombination der vorherigen drei.

Name	LOESCH
Funktion	Lösche Grafikspeicher
Syntax	GRLOE oder SYS$9406 bzw. SYS37894
Parameter	keine
Hilfsroutinen	keine
Adressbereich	$949D - $94BB = 38045 - 38075

```
949D          LOESCH:        ;** GRAFIK LOESCHEN **
949D  A9E0      LDA #GANFH
949F  85FE      STA Z+1
94A1  A900      LDA #0
94A3  85FD      STA Z          ;Z = GANF
94A5          LOE1:
94A5  A8        TAY            ;Y = 0
94A6          LOE2:
94A6  91FD      STA (Z),Y
94A8  C8        INY
94A9  D0FB      BNE LOE2       ;SCHLEIFE LAUFVAR. Y
94AB  E6FE      INC Z+1
94AD  A4FE      LDY Z+1
94AF  C0FF      CPY #GENDH     ;ENDEKRITERIUM
94B1  D0F2      BNE LOE1       ;SCHLEIFE LAUFVAR Z+1
94B3  A8        TAY            ;Y=0
94B4          LOE3:          ;RESTLICHE
94B4  91FD      STA (Z),Y      ;64 BYTES
94B6  C8        INY            ;LOESCHEN
94B7  C040      CPY #64        ;8000=$FF64
94B9  D0F9      BNE LOE3
94BB  60        RTS     ;ENDE UPRO GRAPHIK LOESCHEN
94BC            ;
```

Beschreibung Als Zeiger auf die momentan zu löschende Speicherzelle dient das Zellenpaar Z,Z+1, wobei immer gleich pageweise gelöscht wird. Die letzten 64 Bytes werden extra in einer Schleife gelöscht.

```
Name            FARBE
Funktion        Beschreibe Video-RAM und gegebenenfalls
                Farb-RAM und setze Multi-Flag
Syntax          FARBE F1,F2 oder FARBE F1,F2,F3 oder
                SYS($9409) F1,F2 oder SYS($9409) F1,F2,F3
                bzw. SYS(37897) F1,F2 oder
                SYS(37897) F1,F2,F3
Parameter       F1 = 1. Farbe
                F2 = 2. Farbe
                F3 = eventuell 3. Farbe
Hilfsroutinen   keine
Adressbereich   $94BC - $9505  =  38076 - 38149
```

```
94BC            FARBE:      ;** FARBEN LESEN/SETZEN
94BC  209EB7      JSR GETBYT ; ZEICHENFARBE -> X
94BF  8A          TXA
94C0  0A          ASL A
94C1  0A          ASL A
94C2  0A          ASL A
94C3  0A          ASL A           ;A = ZF * 16
94C4  8D5C94      STA PFARBE
94C7  20F1B7      JSR GETBYTC     ;HI.GR.FARBE -> X
94CA  8A          TXA
94CB  0D5C94      ORA PFARBE      ;A = ZF * 16 + HF
94CE  A000        LDY #0          ;SCHLEIFENZAE.=0
94D0  8C5D94      STY MULTI       ;MULTI-COLOUR AUS
94D3            FARB1:
94D3  9900CC      STA VANFANG,Y     ;BYTE 0-249
94D6  99FACC      STA VANFANG+250,Y ;BYTE 250-499
94D9  99F4CD      STA VANFANG+500,Y ;BYTE 500-749
94DC  99EECE      STA VANFANG+750,Y ;BYTE 750-999
94DF  C8          INY
94E0  C0FA        CPY #250
94E2  D0EF        BNE FARB1       ;SCHLEIFE BIS Y=250
94E4  207900      JSR CHRGOT      ;LETZTES ZEICHEN
94E7  F01C        BEQ FARBEND     ;TRENNZEICHEN
94E9  20F1B7      JSR GETBYTC     ;3. FARBE LESEN
94EC  A9FF        LDA #$FF
94EE  8D5D94      STA MULTI       ;MULTI-FLAG SETZEN
94F1  8A          TXA
94F2  A000        LDY #0
94F4            FARB2:
94F4  9900D8      STA FARBRAM,Y     ;FARBRAM 1.TEIL
94F7  99FAD8      STA FARBRAM+250,Y ;FARBRAM 2.TEIL
94FA  99F4D9      STA FARBRAM+500,Y ;FARBRAM 3.TEIL
94FD  99EEDA      STA FARBRAM+750,Y ;FARBRAM 4.TEIL
9500  C8          INY
9501  C0FA        CPY #250          ;JE 250 BYTE
9503  D0EF        BNE FARB2         ;SCHLEIFE
9505            FARBEND:
9505  60          RTS       ;ENDE UPRO FARBEN SETZEN
```

Beschreibung Die beiden ersten Farben werden mit GETBYT geholt, zu einem Byte zusammengefaßt und damit der gesamte Video-RAM beschrieben. Dann wird geprüft, ob ein Komma folgt und in diesem Fall die dritte Farbe geholt. Das Multi-Flag wird dann auf $FF gesetzt und das Farb-RAM mit der dritten Farbe beschrieben.

Name	GRAFEIN
Funktion	schalte auf Grafik-Modus um, auch auf Multi-Modus, wenn Multi-Flag = $FF.
Syntax	GRAFEIN oder SYS($940C) oder SYS(37900)
Parameter	keine
Hilfsroutine	keine
Adressbereich	$9474 - $949C = 38004 - 38044

```
9474            GRAFEIN:
9474  A95E        LDA #<NMIHILF;ADRESSE VON HILFS-
9476  8DFAFF      STA $FFFA    ;NMI-ROUTINE ALS
9479  A994        LDA #>NMIHILF;NMI-VEKTOR IM RAM
947B  8DFBFF      STA $FFFB
947E  A994        LDA #%10010100
9480  8D00DD      STA $DD00    ;VIDEOBERICH AB $C000
9483  A93B        LDA #%00111011
9485  8D11D0      STA VIC+17   ;HI-RES EIN
9488  A93B        LDA #%00111011 ;VIDEO-RAM AB $CC00
948A  8D18D0      STA VIC+24  ;GRAPHIK-RAM BEI $E000
948D  AD16D0      LDA VIC+22
9490  29EF        AND #%11101111 ;MULTI-COLOUR AUS
9492  2C5D94      BIT MULTI
9495  1002        BPL GRAFEIN1
9497  0910        ORA #%00010000 ;MULTI-COLOUR EIN
9499            GRAFEIN1:
9499  8D16D0      STA VIC+22
949C  60          RTS            ;FERTIG (GREIN)
949D              ;
```

Beschreibung Im VIC-Register 17 wird auf HIRES umgeschaltet, im Register 24 werden die Zeiger auf die Bit-Map und den Video-RAM gesetzt, im Register 22 im Bedarfsfall das Multi-Bit gesetzt und im CIA2 der Arbeitsbereich des VIC verlegt.

```
Name            GREIN
Funktion        kombinierter Aufruf von LOESCH, FARBE,
                GRAFEIN
Syntax          GREIN F1, F2 bzw. GREIN F1, F2, F3 oder
                SYS($9400) F1,F2 bzw. SYS(37888) F1,F2
                SYS($9400) F1,F2,F3 bzw.
                SYS(37888) F1,F2,F3
Parameter       F1 = 1. Farbe
                F2 = 2. Farbe
                F3 = eventuell 3. Farbe
Hilfsroutinen   LOESCH, FARBE, GRAFEIN
Adressbereich   LOESCH :$949D - $94BB  =  38045 - 38075
                FARBE  :$94BC - $9505  =  38076 - 38149
                GRAFEIN:$9474 - $949C  =  38004 - 38044
                GREIN  :$946A - $9473  =  37994 - 38003
```

```
946A          GREIN:
946A  209D94    JSR LOESCH
946D  20BC94    JSR FARBE
9470  207494    JSR GRAFEIN
9473  60        RTS
9474            ;
```

Beschreibung Hier werden die drei oben genannten Routinen nacheinander aufgerufen.

2.2.13.2 Grafik ausschalten

```
Name            GRAUS
Funktion        Grafik ausschalten
Syntax          GRAUS oder SYS($9403) oder SYS(37891)
Parameter       keine
Hilfsroutine    keine
Adressbereich   $9506 - 951D  =  38150 - 38173
```

```
9506          GRAUS:            ;** GRAFIK AUS **
9506  A997      LDA #%10010111  ;STANDARDWERTE
9508  8D00DD    STA $DD00       ;CIA2 PORT A
950B  A91B      LDA #%00011011
950D  8D11D0    STA VIC+17
9510  A915      LDA #%00010101  ;NORMALWERTE IM VIC
9512  8D18D0    STA VIC+24      ;WIEDERHERSTELLEN
9515  AD16D0    LDA VIC+22
```

```
9518  29EF      AND #%11101111
951A  8D16D0    STA VIC+22      ;MULTI-COLOR AUS
951D  60        RTS             ;FERTIG (GRAUS)
951E            :
```

Beschreibung Diese Routine macht die Änderungen der Routine GRAFEIN rückgängig und setzt damit den Video-RAM zurück nach $0400 und schaltet die entsprechenden anderen Bits im Video-Kontroller und im CIA2 zurück.

2.2.13.3 Punkt setzen

Zunächst soll die eigentliche Routine zum Setzen eines Punktes beschrieben werden, danach die dazu notwendigen Routinen zum Lesen der Punktfarbe und zum Multiplizieren.

Die Punkt-Routine hat zwei Einsprungstellen, die eine (PUNKT) zur Verwendung als Basic-Befehl, die andere (PUNKS) zur Verwendung für weitere Maschinen-Routinen (Aufruf:JSR PUNKS), wobei in den Zellen ADR,ADR+1 die X-Koordinate, im X-Register die Y-Koordinate und in der Zelle PFARBE die Farbe ($FF,0,1,2,3,4) übergeben werden muß.

```
Name            PUNKT bzw. PUNKS
Funktion        setze, lösche oder invertiere einen Punkt
Syntax          PUNKT PF,PX,PY oder SYS (37903)PF,PX,PY
Parameter       PF = Punktfarbe
                     normal:-1 = Invertieren
                             0 = Löschen
                             1 = Setzen
                     Multi : 0 = Löschen
                             1 = 1. Farbe
                             2 = 2. Farbe
                             3 = 3. Farbe
                PX = X-Koordinate des Punktes
                     normal: Wert zwischen 0 und 319
                     Multi : Wert zwischen 0 und 159
                PY = Y-Koordinate des Punktes (0-199)
Hilfsroutinen   MULT,PFL
Adressbereich   MULT :$9605 - $9628  =  38405 - 38440
```

```
                PUNKT:$951E - $95EE  =  38174 - 38302
                PFL  :$95EF - $9604  =  38383 - 38404

951E           PUNERR:       ;** FEHLER BEI PUNKS **
951E  A501       LDA 1       ;PROZESSORPORT
9520  0907       ORA #%00000111 ;STANDARDWERTE
9522  8501       STA 1       ;SETZEN
9524  4C48B2     JMP ILQERR ;"ILLEGAL QUANTITY"
9527             ;
9527           PUNKT:        ;PUNKT ALS BASIC-BEFEHL
9527  20EF95     JSR PFL     ;PUNKTFARBE LESEN
952A  20FDAE     JSR CHKCOM ;KOMMA UEBERLESEN
952D  20EBB7     JSR GETAB   ;KOORDINATEN LESEN
9530             ;
9530           PUNKS:        ;** PUNKT SETZEN **
9530             ;X-KOORD. IN ADR,ADR+1
9530             ;Y-KOORD. IM X-REGISTER
9530             ;
9530  E0C8       CPX #200
9532  B0EA       BCS PUNERR ;Y-KOORD. > 199
9534  2C5D94     BIT MULTI
9537  1003       BPL FORTS   ;WENN MINUSFLAG=1
9539  4CC495     JMP PUNKM   ;DANN SPRUNG NACH PUNKM
953C           FORTS:        ;SONST FORTSETZUNG
953C  A515       LDA ADR+1
953E  F00A       BEQ PUN1    ;X-KOORD. < 256
9540  C901       CMP #1
9542  D0DA       BNE PUNERR ;X-KOORD. > 511
9544  A514       LDA ADR
9546  C940       CMP #64
9548  B0D4       BCS PUNERR ;X-KOORD. > 319
954A           PUN1:
954A  A514       LDA ADR
954C  2907       AND #$07
954E  A8         TAY         ; Y = X AND 7
954F  38         SEC         ; ZUM HINEINROTIEREN
9550  A900       LDA #$00
9552           PUN2:
9552  6A         ROR A
9553  88         DEY         ; ZAEHLER
9554  10FC       BPL PUN2
9556             ;             A=2^(7-(XK AND 7))
9556  85AB       STA PBYTE   ; WERT SICHERN
9558             ;
9558  A514       LDA ADR
955A  29F8       AND #$F8
955C  8514       STA ADR     ; ADR = XK AND $FFF8
955E             ;
955E           PUN3:
955E  08         PHP         ;PROZ.STATUS SICHERN
```

```
955F  78       SEI          ;INTERRUPT VERHINDERN
9560  A501     LDA 1
9562  48       PHA          ;PORTWERT MERKEN
9563  29FD     AND #%11111101 ;HIRAM = 0
9565  8501     STA 1        ;KERNAL AUSBLENDEN
9567  8A       TXA
9568  2907     AND #$07     ; A = YK AND 7
956A  18       CLC
956B  6514     ADC ADR      ; A = ADR + (YK AND 7)
956D  8514     STA ADR
956F  A515     LDA ADR+1
9571  69E0     ADC #GANFH
9573  8515     STA ADR+1
9575           ;ADR= GANF+(XK AND $FFF8)+(YK AND 7)
9575  8A       TXA
9576  29F8     AND #$F8
9578  8559     STA FAK1L
957A  A900     LDA #$00
957C  855A     STA FAK1H  ;FAK1 = YK AND $00F8
957E  A928     LDA #40
9580  855B     STA FAK2   ;FAK2 = 40
9582  200596   JSR MULT   ;PRD = FAK1 * FAK2
9585  18       CLC
9586  A514     LDA ADR
9588  6557     ADC PRDL
958A  8514     STA ADR
958C  A515     LDA ADR+1
958E  6558     ADC PRDH
9590  8515     STA ADR+1  ;ADR = ADR + PRD
9592           ; ADR= GANF+(XK AND $FFF8) +
9592           ;  + (YK AND 7) + 40 * (YK AND $F8)
9592  A5AB     LDA PBYTE  ; ZU SETZENDES BIT
9594  A000     LDY #$00
9596  2C5D94   BIT MULTI
9599  AE5C94   LDX PFARBE
959C  7012     BVS PUN4   ; SPRUNG, WENN MULTI
959E  1004     BPL PUN5
95A0  5114     EOR (ADR),Y ;PUNKT INVERTIEREN
95A2  5019     BVC PUNE    ;UNBED. SPRUNG
95A4         PUN5:
95A4  F004     BEQ PUN6
95A6  1114     ORA (ADR),Y ;PUNKT SETZEN
95A8  5013     BVC PUNE
95AA         PUN6:
95AA  49FF     EOR #$FF
95AC  3114     AND (ADR),Y ;PUNKT LOESCHEN
95AE  500D     BVC PUNE
95B0         PUN4:
95B0  48       PHA
95B1  49FF     EOR #$FF
95B3  3114     AND (ADR),Y ;BITPAAR LOESCHEN
```

```
95B5  85AB      STA PBYTE
95B7  68        PLA
95B8  3DEB95    AND PMASK,X
95BB  05AB      ORA PBYTE    ;EVTL. BITS SETZEN
95BD          PUNE:
95BD  9114      STA (ADR),Y ;UND SPEICHERN
95BF  68        PLA         ;PROZESSORPORT
95C0  8501      STA 1       ;RESTAURIEREN
95C2  28        PLP         ;PROZ.STATUS RESTAU.
95C3  60        RTS         ;ENDE PUNKT SETZEN
95C4            ;
95C4          PUNKM:        ;PUNKT MULTI-COLOUR
95C4  A515      LDA ADR+1
95C6  F003      BEQ PUNKM1  ;PX < 256
95C8          PUNKM0:
95C8  4C1E95    JMP PUNERR  ;FEHLER
95CB          PUNKM1:
95CB  A514      LDA ADR
95CD  C9A0      CMP #160
95CF  B0F7      BCS PUNKM0  ;PX > 159
95D1  2903      AND #%00000011 ;PX AND 3
95D3  A8        TAY
95D4  B9E795    LDA POSMASK,Y  ;ENTSPR. BIT-MASKE
95D7  85AB      STA PBYTE      ;MERKEN  (3*BX)
95D9  A514      LDA ADR        ;PX AND $FC
95DB  29FC      AND #%11111100
95DD  0A        ASL A          ;(PX AND $FC) * 2
95DE  8514      STA ADR        ;ALS NEUE X-KOORD.
95E0  9002      BCC PUNKM2
95E2  E615      INC ADR+1
95E4          PUNKM2:
95E4  4C5E95    JMP PUN3       ;WEITER WIE OBEN
95E7            ;
95E7          POSMASK:
95E7  C0        BYT %11000000
95E8  30        BYT %00110000
95E9  0C        BYT %00001100
95EA  03        BYT %00000011
95EB          PMASK:
95EB  00        BYT %00000000
95EC  55        BYT %01010101
95ED  AA        BYT %10101010
95EE  FF        BYT %11111111
95EF            ;
```

Beschreibung Das Label PUNERR wird angesprungen, wenn die Koordinaten nicht den Bedingungen entsprechen. Dort muß der Prozessor-Port auf die Standardwerte zurückgesetzt werden, da

sonst keine Fehlermeldung ausgegeben werden könnte.

Vor dem eigentlichen Setzen des Punktes werden die Punktfarbe und die Koordinaten gelesen. Dann werden die Koordinaten auf Plausibilität geprüft und je nach Modus (Multi ja/ nein) verzweigt. Die Behandlung eines Punktes im normalen Modus geschieht zwischen den Labeln FORTS und PUN3, die Behandlung des Multi-Color-Modus ab dem Label PUNKM. Im normalen Modus wird lediglich die Maske für das zu setzende Bit in PBYTE berechnet und in der X-Koordinate die letzten 3 Bit annulliert. Im Multi-Color-Modus folgt die Berechnung der Bitmaske durch Lesen aus einer Tabelle. Für beide Modes gemeinsam ist das Programmstück ab dem Label PUN3. Hier muß zunächst der alte Prozessorstatus (Interrupt-Flag) gesichert werden, anschließend wird das KERNAL ausgeschaltet. Dann wird die zu ändernde Speicherzelle berechnet und dort die Änderungen je nach Punktfarbe verschieden vorgenommen. Am Schluß werden der Prozessor-Port und der Prozessor-Status restauriert.

Name	PFL
Funktion	Lies Punktfarbe
Aufruf	JSR PFL
Parameter	Im Basic-Text folgt die Farbnummer (siehe PUNKT)
Hilfsroutinen	keine
Adressbereich	$95EF - $9604 = 38383 - 38404

```
95EF           PFL:          ;** PUNKTFARBE LESEN **
95EF  2C5D94     BIT MULTI   ;MULTI-MODUS
95F2  300A       BMI PFL1
95F4  208AAD     JSR FRMNUM  ;WERT HOLEN
95F7  202BBC     JSR FSGN    ;VORZEICHEN BESTIMMEN
95FA  8D5C94     STA PFARBE  ;ALS FARBE SPEICHERN
95FD  60         RTS
95FE           PFL1:
95FE  209EB7     JSR GETBYT  ;WERT HOLEN
9601  8E5C94     STX PFARBE  ;SPEICHERN
9604  60         RTS         ;FERTIG
9605             ;
```

Beschreibung Im normalen Modus wird lediglich das Vorzeichen des Farbparameters als Punktfarbe gespeichert, im Multi-Modus wird die Farbzahl mit GETBYT geholt.

Name	MULT
Funktion	Multipliziere 16-Bit-Wert mit 8-Bit-Wert
Aufruf	JSR MULT
Parameter	FAK1L, FAK1H = 1. Faktor
	FAK2 = 2. Faktor
	PRDL, PRDH = Produkt
Hilfsroutinen	keine
Adressbereich	$9605 - $9628 = 38405 - 38440

```
9605            MULT:
9605  A900        LDA #$00
9607  8557        STA PRDL
9609  8558        STA PRDH     ;PRD = 0
960B  A008        LDY #$08     ;SCHLEIFENZAEHLER
960D  18          CLC
960E            MULT1:
960E  265B        ROL FAK2
9610  900D        BCC MULT2    ;NICHTS ADDIEREN
9612  18          CLC
9613  A557        LDA PRDL
9615  6559        ADC FAK1L
9617  8557        STA PRDL
9619  A558        LDA PRDH
961B  655A        ADC FAK1H
961D  8558        STA PRDH     ;PRD = PRD + FAK1
961F            MULT2:
961F  88          DEY
9620  F006        BEQ MULEND   ;8 BIT VERARBEITET
9622  2657        ROL PRDL
9624  2658        ROL PRDH     ;PRD VERDOPPELN
9626  90E6        BCC MULT1    ;CARRY MUSS 0 SEIN
9628            MULEND:
9628  60          RTS          ;ENDE UPRO MULT
9629              ;
```

Beschreibung Der Multiplikations-Algorithmus ist angelehnt an das Verfahren, das man mit Bleistift und Papier ausführt. Zunächst wird das Produkt annulliert, dann der zweite Faktor um ein Bit nach links verschoben, dann wird untersucht, ob zum Produkt der erste Faktor addiert werden muß, in jedem Fall aber das Produkt verdoppelt. Als Bit-Zähler dient das Y-Register. Eigentlich

benötigt das Ergebnis von einer 16-Bit mal 8-Bit Multiplikation 24 Bit, jedoch sind bei unserer Aufgabenstellung die obersten 8 Bit 0, so daß diese weggelassen werden.

2.2.13.4 Textzeichen in Grafik

Name	CHAR
Funktion	Positioniere Textzeichen in Grafik
Syntax	CHAR PF,PX,PY,BC oder SYS (37906) PF,PX,PY,BC
Parameter	PF = Punktfarbe (siehe PUNKT) PX = X-Koordinate PY = Y-Koordinate BC = Bildschirmcode des zu druckenden Zeichens: 0 - 127 : normale Zeichen des Großschriftzeichensatzes 128 - 255 : Revers-Zeichen des Großschriftzeichensatzes 256 - 383 : normale Zeichen des Kleinschriftzeichensatzes 384 - 511 : Revers-Zeichen des Kleinschriftzeichensatzes
Hilfsroutinen	PUNKS,PFL
Adressbereich	PUNKS:$951E - $95EE = 38174 - 38302 CHAR :$9629 - $96A2 = 38441 - 38562 PFL :$95EF - $9604 = 38383 - 38404

```
9629            BITTAB:
9629  80          BYT %10000000
962A  40          BYT %01000000
962B  20          BYT %00100000
962C  10          BYT %00010000
962D  08          BYT %00001000
962E  04          BYT %00000100
962F  02          BYT %00000010
9630  01          BYT %00000001
9631              ;
9631  00        BITZAE:    BYT 0 ;BIT-ZAEHLER
9632  00        BYTEZAE:    BYT 0;BYTE-ZAEHLER
9633              ;
9633            CHAR:             ;*** CHAR-BEFEHL ***
9633  20EF95      JSR PFL         ;PUNKTFARBE LESEN
9636  20FDAE      JSR CHKCOM      ;KOMMA PRUEFEN
9639  20EBB7      JSR GETAB       ;KOORD. HOLEN
963C  86A6        STX AY          ;Y-KOORD.
963E  A514        LDA ADR         ;X-KOORD.
```

```
9640  85A4      STA AX
9642  A515      LDA ADR+1
9644  85A5      STA AX+1
9646  20FDAE    JSR CHKCOM    ;KOMMA PRUEFEN
9649  208AAD    JSR FRMNUM    ;BS-CODE LESEN
964C  E661      INC $61
964E  E661      INC $61
9650  E661      INC $61       ;MAL 8 (EXP. +3)
9652  20F7B7    JSR FACADR    ;-> ADRESSFORMAT
9655  84FD      STY Z         ;LOW-BYTE
9657  18        CLC
9658  69D0      ADC #$D0      ;$D000 ADDIEREN
965A  85FE      STA Z+1       ;HIGH-BYTE
965C            ;
965C          CHARS:
965C  78        SEI
965D  A501      LDA 1         ;PROZESSORPORT
965F  29FB      AND #%11111011; CHAREN=0
9661  8501      STA 1
9663  A907      LDA #7        ;
9665  8D3296    STA BYTEZAE   ;ZAEHLER AUF 7
9668          CHARS1:
9668  A907      LDA #7
966A  8D3196    STA BITZAE    ;BITZAEHLER AUF 7
966D          CHARS2:
966D  AE3196    LDX BITZAE    ;BITNUMMER
9670  AC3296    LDY BYTEZAE   ;BYTE-NUMMER (ZEILE)
9673  B1FD      LDA (Z),Y     ;ZEICHEN AUS CHARGEN.
9675  3D2996    AND BITTAB,X ;MASKE F. 1 BIT
9678  F017      BEQ CHARS3    ;NICHTS TUN
967A  18        CLC
967B  A5A4      LDA AX        ;X-KOORD. BERECHNEN
967D  6D3196    ADC BITZAE    ;XK = AX + BITZAE
9680  8514      STA ADR       ;PARAMETER F. PUNKS
9682  A5A5      LDA AX+1      ;HIGH-BYTE
9684  6900      ADC #0
9686  8515      STA ADR+1
9688  A5A6      LDA AY        ;Y-KOORD. BERECHNEN
968A  6D3296    ADC BYTEZAE   ;YK = AY + BYTEZAE
968D  AA        TAX           ;PARAMETER F. PUNKS
968E  203095    JSR PUNKS     ;PUNKT SETZEN
9691          CHARS3:
9691  CE3196    DEC BITZAE    ;BITZAEHLER - 1
9694  10D7      BPL CHARS2    ;SCHLEIFE
9696  CE3296    DEC BYTEZAE   ;BYTEZAEHLER - 1
9699  10CD      BPL CHARS1    ;SCHLEIFE
969B  A501      LDA 1         ;PROZESSORPORT
969D  0904      ORA #%00000100; CHAREN=1
969F  8501      STA 1         ;RESTAURIEREN
96A1  58        CLI
96A2  60        RTS           ;ENDE CHAR
```

Beschreibung Ab dem Label CHAR werden zunächst die Punktfarbe und die Punktkoordinaten eingelesen, dann mit FRMNUM der Bildschirmcode gelesen. Durch Erhöhung des Exponenten ($61) um 3 erfolgt eine Multiplikation mit 8, wonach das Ergebnis in Integer umgewandelt und um $D000 vermehrt in die Zellen Z,Z+1 gespeichert wird. Damit sind die Parameter für die Routine CHARS gesetzt, die auch vom Maschinenprogramm aus aufgerufen werden könnte.

Zunächst werden die Interrupts verhindert und der Zeichengenerator lesbar geschaltet. Als Zeilenzähler dient die Zelle BYTEZAE, als Spaltenzähler BITZAE. Ist das entsprechende Bit im Zeichen nicht gesetzt, so wird ohne Aktion auf die nächste Punktposition übergegangen. Im anderen Fall werden die Parameter für die Routine PUNKS gesetzt und diese aufgerufen.

Die beiden Schleifen werden erst beendet, wenn beide Zähler null sind; dann wird noch der Prozessorport auf den alten Wert gesetzt und Interrupts wieder ermöglicht.

2.2.14 Funktionstasten programmieren

Diese Routine wird - wie der Keyclick - vom Interrupt-Manager aufgerufen. Die Routine ist ohne Interrupt-Manager verwendbar, wenn Sie den Interrupt-Vektor auf die unten abgebildete Routine stellen und das RTS am Ende durch ein JMP $EA31 ersetzen. Die Tabelle der Tasten-Belegungen befindet sich im Bereich $C100 bis $C2FF. Wie Sie die Belegung nachträglich ändern und speichern können ist nachfolgend in einem kleinen Basic-Programm gezeigt. Dieses Basic-Programm können Sie auch verwenden, um die aktuelle Belegung der Funktionstasten festzustellen.

Name	FTAST
Funktion	Funktionstasten abfragen; gegebenenfalls Befehlswort in Tastaturpuffer legen
Aktivierung	POKE$C809, PEEK($C809) OR 4
Hilfsroutinen	Interrupt-Manager (Kapitel 2.2.1.6)

```
Adressbereich  FTAST:$C030 - $C09B  =  49200 - 49307
               FTAB :$C100 - $C2FF  =  43408 - 49919

C030            ;
C030            FTAB =$C100 ;BELEGUNGEN AB $C100
C030            ;
C030          FTAST:
C030  A5D4      LDA $D4      ;"-FLAG
C032  D059      BNE FTAST9
C034  8D8EC0    STA FTNR     ;NUMMER ANNULLIEREN
C037  A6CB      LDX $CB      ;GEDRUECKTE TASTE
C039  CA        DEX
C03A  CA        DEX
C03B  CA        DEX          ;MINUS 3
C03C  E004      CPX #4       ;TASTENCODE
C03E  B048      BCS FTAST8   ;GROESSER GLEICH 7
C040  BD90C0    LDA FTASTTAB,X ;NUMMER VON F
C043  AE8D02    LDX $028D       ;FLAG SH.,C=,CTRL
C046  7D94C0    ADC FSHTAB,X    ;SHIFT-WERT ADD.
C049  8D8EC0    STA FTNR     ;ERGEBNIS MERKEN
C04C            ;
C04C  A5CC      LDA $CC      ;BLINKT CURSOR
C04E  D03D      BNE FTAST9   ;NEIN
C050  A5CB      LDA $CB      ;GEDR. TASTE
C052  CD8FC0    CMP FTALT    ;ALTE GEDR. TASTE
C055  F036      BEQ FTAST9   ;GLEICH
C057  A5D8      LDA $D8      ;INSERT-MODUS
C059  D02D      BNE FTAST8   ;JA
C05B            ;
C05B  A5FE      LDA $FE      ;HILFSZEIGER
C05D  48        PHA          ;RETTEN
C05E  A5FD      LDA $FD
C060  48        PHA
C061  AE8EC0    LDX FTNR     ;NUMMER VON FUNKTION
C064  CA        DEX          ;EINS ABZIEHEN
C065  8A        TXA
C066  0A        ASL A
C067  0A        ASL A
C068  0A        ASL A        ;MAL 16
C069  0A        ASL A
C06A  85FD      STA $FD      ;LOW-BYTE ZEIGER
C06C  A9C1      LDA #>FTAB   ;HIGH-BYTE FTAB
C06E  6900      ADC #0       ;CARRY ADDIEREN
C070  85FE      STA $FE      ;ZEIGER AUF STRING
C072  A000      LDY #0       ;ZEICHENZAEHLER
C074          FTAST4:
C074  B1FD      LDA ($FD),Y ;BELEGUNG
C076  F008      BEQ FTAST6   :TRENNZEICHEN
```

```
C078  997702    STA $0277,Y ;TASTATURPUFFER
C07B  C8        INY
C07C  C00F      CPY #15     ;MAX. 15 ZEICHEN
C07E  90F4      BCC FTAST4  ;SCHLEIFE
C080          FTAST6:
C080  84C6      STY $C6     ;ANZ. ZEI. IM TASTPUFF
C082          FTAST7:
C082  68        PLA
C083  85FD      STA $FD     ;HILSZELLEN
C085  68        PLA         ;RESTAURIEREN
C086  85FE      STA $FE
C088          FTAST8:
C088  A5CB      LDA $CB     ;GEDR. TASTE
C08A  8D8FC0    STA FTALT   ;ALTE TASTE
C08D          FTAST9:
C08D  60        RTS         ;FERTIG
C08E            ;
C08E  00      FTNR:   BYT 0 ;NUMMER VON F-TASTE
C08F  40      FTALT:  BYT 64;ALTE GEDR. TASTE
C090          FTASTTAB:
C090  07        BYT 7       ;F7
C091  01        BYT 1       ;F1
C092  03        BYT 3       ;F3
C093  05        BYT 5       ;F5
C094          FSHTAB:
C094  00        BYT 0       ;NICHTS (SH.,C=,CTRL)
C095  01        BYT 1       ;SHIFT
C096  08        BYT 8       ;C=
C097  09        BYT 9       ;SHIFT & C=
C098  10        BYT 16      ;CTRL
C099  11        BYT 17      ;CTRL & SHIFT
C09A  18        BYT 24      ;CTRL & C=
C09B  19        BYT 25      ;CTRL & C= & SHIFT
C09C            ;
C0FF  00        %DUP FTAB-$, BYT 0
C100            ;           *** HIER FTAB ***
C101  495354    %ASC "LIST"
C104  0D        BYT 13      ;RETURN
C10F  00        %DUP $C110-$,BYT 0
C110            ;
C119  3A4CC9    %ASC "OP4,4:CM4:LI"
C11C  0D        BYT 13      ;RETURN
C11F  00        %DUP $C120-$,BYT 0
C120            ;
C120  52554E    %ASC "RUN"
C123  0D        BYT 13      ;RETURN
C12F  00        %DUP $C130-$,BYT 0
C130            ;
C137  534534    %ASC "PR4:CLOSE4"
C13A  0D        BYT 13      ;RETURN
```

```
C13F  00        %DUP $C140-$,BYT 0
C140            ;
C141  4F4E54    %ASC "CONT"
C144  0D        BYT 13      ;RETURN
C14F  00        %DUP $C150-$,BYT 0
C150            ;
C15A  353338    %ASC "?FRE(0)+65538"
C15D  0D        BYT 13      ;RETURN
C15F  00        %DUP $C160-$,BYT 0
C160            ;
C160  08        BYT  8      ;SHIFT/C= SPERREN
C161  0E        BYT  14     ;KLEINSCHRIFT
C16F  00        %DUP $C170-$,BYT 0
C170            ;
C170  08        BYT  8      ;SHIFT/C= SPERREN
C171  8E        BYT  142    ;GROSSCHRIFT
C17F  00        %DUP $C180-$,BYT 0
C180            ;
C181  4F4144    %ASC "LOAD"
C184  22        BYT 34      ;"
C185  22        BYT 34      ;2. HOCHKOMMA
C186  14        BYT 20      ;DELETE
C18F  00        %DUP $C190-$,BYT 0
C190            ;
C197  38302C    %ASC "POKE53280,"
C19F  00        %DUP $C1A0-$,BYT 0
C1A0            ;
C1A0  22        BYT 34      ;"
C1A1  2C38      %ASC ",8"
C1A3  0D        BYT 13      ;RETURN
C1AF  00        %DUP $C1B0-$,BYT 0
C1B0            ;
C1B7  38312C    %ASC "POKE53281,"
C1BF  00        %DUP $C1C0-$,BYT 0
C1C0            ;
C1C1  415645    %ASC "SAVE"
C1C4  22        BYT 34      ;"
C1C5  22        BYT 34      ;2. HOCHKOMMA
C1C6  14        BYT 20      ;DELETE
C1CF  00        %DUP $C1D0-$,BYT 0
C1D0            ;
C1D3  494659    %ASC "VERIFY"
C1D6  22        BYT 34      ;"
C1D7  22        BYT 34      ;2. HOCHKOMMA
C1D8  14        BYT 20      ;DELETE
C1DF  00        %DUP $C1E0-$,BYT 0
C1E0            ;
C1E1  415645    %ASC "SAVE"
C1E4  22        BYT 34      ;"
C1E5  22        BYT 34      ;2. HOCHKOMMA
```

```
C1E6  14        BYT 20      ;DELETE
C1E7  403A      %ASC "§:"
C1EF  00        %DUP $C1F0-$,BYT 0
C1F0            ;
C1F3  494659    %ASC "VERIFY"
C1F6  0D        BYT 13      ;RETURN
C1FF  00        %DUP $C200-$,BYT 0
C200            ;
C205  323030    %ASC "SYS51200"
C208  0D        BYT 13      ;RETURN
C20F  00        %DUP $C210-$,BYT 0
C210            ;
C215  313532    %ASC "SYS49152"
C218  0D        BYT 13      ;RETURN
C21F  00        %DUP $C220-$,BYT 0
C220            ;
C221  49534B    %ASC "DISK"
C224  0D        BYT 13      ;RETURN
C22F  00        %DUP $C230-$,BYT 0
C230            ;
C231  494557    %ASC "VIEW"
C234  22        BYT  34     ;"
C235  24        %ASC "$"
C236  22        BYT  34     ;"
C237  2C38      %ASC ",8"
C239  0D        BYT 13      ;RETURN
C23F  00        %DUP $C240-$,BYT 0
C240            ;
C242  44302C    %ASC "LAD0,"
C245  22        BYT  34     ;"
C246  4532      %ASC "E2"
C248  22        BYT  34     ;"
C24A  382C31    %ASC ",8,1"
C24D  0D        BYT 13      ;RETURN
C24F  00        %DUP $C250-$,BYT 0
C250            ;
C257  30392C    %ASC "POKE$C809,"
C25F  00        %DUP $C260-$,BYT 0
C260            ;
C260  463233    %ASC "F23"
C26F  00        %DUP $C270-$,BYT 0
C270            ;
C270  463234    %ASC "F24"
C27F  00        %DUP $C280-$,BYT 0
C280            ;
C280  463235    %ASC "F25"
C28F  00        %DUP $C290-$,BYT 0
C290            ;
C290  463236    %ASC "F26"
C29F  00        %DUP $C2A0-$,BYT 0
```

```
C2A0                ;
C2A0   463237       %ASC "F27"
C2AF   00           %DUP $C2B0-$,BYT 0
C2B0                ;
C2B0   463238       %ASC "F28"
C2BF   00           %DUP $C2C0-$,BYT 0
C2C0                ;
C2C0   463239       %ASC "F29"
C2CF   00           %DUP $C2D0-$,BYT 0
C2D0                ;
C2D0   463330       %ASC "F30"
C2DF   00           %DUP $C2E0-$,BYT 0
C2E0                ;
C2E0   463331       %ASC "F31"
C2EF   00           %DUP $C2F0-$,BYT 0
C2F0                ;
C2F0   463332       %ASC "F32"
C2FF   00           %DUP $C300-$,BYT 0
C300                ;
```

Beschreibung

Ist der Hochkomma-Modus aktiv, so ist die Auswertung von Funktionstasten nicht sinnvoll; deshalb wird in diesem Fall sofort die Routine beendet. Im anderen Fall wird die Nummer der Funktionstaste aus dem Tastencode und den Flags für die Tasten SHIFT, C= und CTRL gebildet. Im Anschluß an diese Beschreibung finden Sie eine Tabelle, die die Zuordnung zwischen Funktionstasten-Nummer und gedrückten Tasten deutlich macht. Die Umwandlung der Funktionstaste in ein Wort ist weiterhin nur sinnvoll, wenn der Cursor gerade blinkt und die alte gedrückte Taste sich von der neuen unterscheidet.

Ist eine der Bedingungen nicht erfüllt, wird zum Ende des Unterprogramms verzweigt. Im anderen Fall werden die Hilfsregister $FD, $FE gerettet und ein Zeiger auf das entsprechende Wort gesetzt. Danach werden bis zu 15 Zeichen in den Tastaturpuffer übertragen. Zur Beachtung: Da der Tastaturpuffer eigentlich nur 10 Zeichen lang ist,

werden die folgenden 4 bis 5 Byte geändert, wenn lange Zeichenreihen zur Funktionstastenbelegung verwendet werden. Für Basic ist dies allerdings ohne Belang.

Hier nun die Tabelle mit den Standardbelegungen der Funktionstasten und der Zuordnung der Funktionstastennummer zu den gedrückten Tasten:

SHIFT	C=	CTRL	Taste	Nr	Belegung
			F1	F1	LIST (RETURN)
X			F1	F2	OPEN4,4:CMD1:LIST (RETURN)
			F3	F3	RUN (RETURN)
X			F3	F4	PRINT#4:CLOSE4 (RETURN)
			F5	F5	CONT (RETURN)
X			F5	F6	PRINT FRE(0)+65538 (RETURN)
			F7	F7	CHR$(8)+CHR$(14)
X			F7	F8	CHR$(8)+CHR$(14)
	X		F1	F9	LOAD"
X	X		F1	F10	POKE 53280,
	X		F3	F11	",8
X	X		F3	F12	POKE 53281,
	X		F5	F13	SAVE"
X	X		F5	F14	VERIFY"
	X		F7	F15	SAVE"(Klammeraffe):
X	X		F7	F16	VERIFY
		X	F1	F17	SYS 51200
X		X	F1	F18	SYS 49152
		X	F3	F19	DISK
X		X	F3	F20	VIEW"$",8
		X	F5	F21	LADO,"E2",8,1
X		X	F5	F22	POKE$C809,
		X	F7	F23	-
X		X	F7	F24	-
	X	X	F1	F25	-
X	X	X	F1	F26	-
	X	X	F3	F27	-
X	X	X	F3	F28	-
	X	X	F5	F29	-
X	X	X	F5	F30	-
	X	X	F7	F31	-
X	X	X	F7	F32	-

An der 'normalen' Belegung der Tasten (Nummern mit und ohne SHIFT) wurde nichts verändert. Für die weitere Numerierung wurde auf ein einfaches Merkschema geachtet:

F 1	-	F 8	Normal
F10	-	F16	Normal + C=
F17	-	F24	Normal + CTRL
F25	-	F32	Normal + C= + CTRL

Eine kurze Erläuterung zu den verwendeten Belegungen:

F 1 - F 6 : häufig benötigte Befehle
F 7 - F 8 : umschalten Groß/Kleinschrift
F 9 - F16 : laden, speichern, Farben setzen
F17 - F22 : Basic-Erweiterungen
F17 : Basic-Erweiterungen einschalten
F18 : Testen von eigenen Programmen
F19 : Disk-Status anzeigen
F20 : Directory auf Bildschirm zeigen
F21 : zweiten Teil der Erweiterungen laden
F22 : Maske des Interrupt-Managers setzen
F23 - F32 : frei für Ihre Zwecke

Zum Abschluß zeigen wir Ihnen noch das Basic-Programm, mit dem Sie die momentane Funktionstastenbelegung feststellen können und gegebenenfalls Änderungen durchführen können. Sie können danach die Änderungen zusammen mit den Basic-Erweiterungen, die ab $C000 liegen gleich als Datei speichern.

```
100 FOR N=1 TO 32                                   <031>
110 PRINT"F";N,CHR$(34);                            <000>
120 FOR I=0 TO 14                                   <045>
130 A=PEEK(49408+(N-1)*16+I)                        <188>
140 IF A=0 THEN 170                                 <121>
150 PRINT CHR$(A);                                  <195>
160 NEXT I                                          <107>
170 PRINT                                           <067>
180 NEXT N                                          <132>
210 N=0:INPUT"NEUE BELEGUNG FUER NR. ";N            <004>
220 IF N=0 THEN 500                                 <211>
230 PRINT"(RETURN = ←) ? ";                         <010>
240 GOSUB 1000                                      <062>
250 PRINT                                           <147>
260 IF LEN(A$)>15 THEN 230                          <092>
270 A$=A$+CHR$(0)                                   <125>
280 FOR I=1 TO LEN(A$)                              <227>
290 B=ASC(MID$(A$,I,1))                             <128>
300 IF B=ASC("←")THEN B=13                          <101>
310 POKE 49408+16*(N-1)+I-1,B                       <076>
320 NEXT                                            <195>
330 GOTO 210                                        <103>
```

```
500 A$=""                                              <080>
510 INPUT"DATEINAME";A$                                <240>
520 SPEI$C000,$CC00,A$,8                               <093>
590 END                                                <208>
1000 A$=""                                             <070>
1010 SYS 65487     :REM $FFCF (CHRIN)                  <104>
1020 IF PEEK(780)=13 THEN RETURN                       <135>
1030 A$=A$+CHR$(PEEK(780))                             <250>
1040 GOTO 1010                                         <095>
```

Beschreibung des Basic-Programmes

Zur Eingabe der Änderung am Bildschirm kann der normale INPUT-Befehl nicht verwendet werden, da die Verwendung von Anführungszeichen, Komma und Doppelpunkt problematisch ist. Statt dessen wird die KERNAL-Routine CHRIN aufgerufen, mit der man diese Einschränkungen umgehen kann. Wie Sie den SPEI-Befehl durch normale Basic-Befehle ersetzen können finden, Sie in Kapitel 4.

Anleitung zur Verwendung des Programmes

Nach dem Starten des Programmes zeigt Ihnen der Rechner die momentane Belegung der Funktionstasten am Bildschirm an. Danach fragt er nach der zu ändernden Belegung (Tastennummer 1-32). Geben Sie '0' ein, fragt der Rechner nach der Datei, in der der Bereich zwischen $C000 und $CBFF gespeichert werden soll. Möglich ist z.B. die Eingabe:

"(Klammeraffe):E1" (mit Anführungszeichen eingeben)

Geben Sie eine Nummer zwischen 1 und 32 ein, können Sie die neue Belegung (direkt) eingeben, wobei ein 'Pfeil nach links' als RETURN-Taste in den Kommando-String mitaufgenommen wird.

2.2.15 Komfortable Tonerzeugung

Daß der Commodore 64 einen professionellen Synthesizer mit überragenden Fähigkeiten integriert hat, wird wohl niemand abstreiten wollen. Mit dem folgenden Programm lassen sich alle drei Tongeneratoren beeinflussen. Dabei können sogar

Filter eingesetzt werden. Ebenso wie die Verwendung der anderen Eigenschaften des Commodore 64 (Sprites, Grafik), ist auch der Einsatz der Tongeneratoren anwendungsabhängig und kein Thema für ein Nachschlagewerk. Trotzdem wollen wir Ihnen hier ein kleines Hilfsprogramm als Ergänzung vorstellen, das in Form eines Basic-Loaders vorliegt.

Das Programm ist mit den anderen Basic-Erweiterungen kompatibel, sofern Sie die Funktionstastenbelegung und den Key-Click nicht verwenden. Die Erweiterungen können Sie durch jeweiliges Aufrufen von SYS 51200 bzw. SYS 49152 ansprechen.

Bedienungsanleitung

Was wäre der C-64 ohne Tongeneratoren? Wahrscheinlich dasselbe wie ein Schotte ohne Dudelsack! Doch was nützt der beste Tongenerator, wenn dieser nur umständlich ansprechbar ist? Um beim Commodore 64 einen einfachen Ton zu erzeugen, müssen mehrere Register (Speicherstellen) beschrieben ('gePOKEd') werden. Sollen zusätzlich weitere Stimmen eingesetzt werden, die eventuell sogar mit Filtern beeinflußt werden sollen, wird das Basic-Programm länger und unübersichtlicher. Abhilfe kann hier eine Befehlserweiterung schaffen, die mit wenig Befehlen möglichst viele Register auf einmal verändert. Wenn Sie dieses Programm (hoffentlich) fehlerlos eingetippt haben, stehen Ihnen sechs weitere Befehle zur Verfügung. Der Interpreter erkennt diese Befehle an einem vorangestellten Punkt.

Es stehen Ihnen folgende Befehle zur Verfügung:

.W ST,WE,(TAh,TAl),A,D,S,R

Dabei ist .W der Befehl, ST ist die Stimme (1-3) und WE die Wellenform (16=Dreieck, 32=Sägezahn, 64=Puls/Rechteck, 128=Rauschen). Haben Sie Puls als Wellenform gewählt, so folgt nach WE das Tastenverhältnis high (TAh) und low (TAl). Die nächsten vier Parameter (A-Attack, D-Decay, S-Sustain, R-Release) nehmen Werte zwischen 0 und 15 an.

A gibt die Zeit an, die benötigt wird, bis die volle Lautstärke erreicht wird (0-6 msec., 15-24 sec.).

D ist die Zeit, die benötigt wird, bis die Lautstärke auf den Sustain-Level zurückfällt.

S ist die Lautstärke, die der Ton hält, bis das Key-Bit gelöscht ist.

R gibt die Zeit an, die benötigt wird, bis der Ton auf die Lautstärke Null absinkt, nachdem das Key-bit gelöscht wurde (0-6 msec., 15-24 sec.)

Diese vier Buchstaben bestimmen zusammen die Hüllkurve, die die Lautstärke beeinflußt, während der Ton gespielt wird. In Abbildung 1 ist dies zum besseren Verständnis nochmals schematisch dargestellt.

Hüllkurvenverlauf eines Tones:

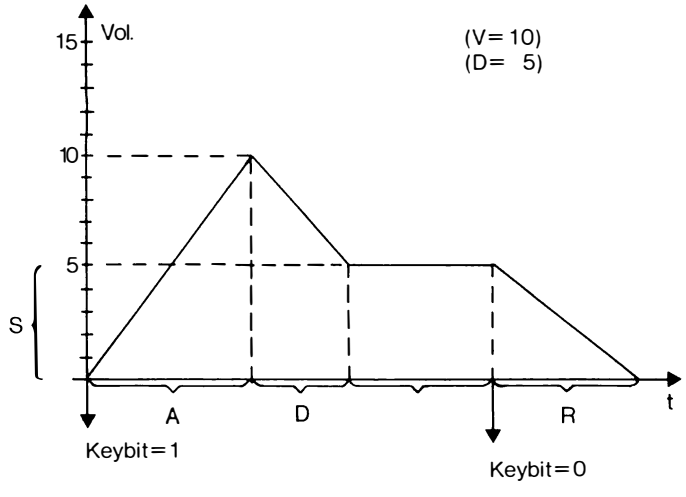


Bild 2.2.15-1 :

.S ST,K,Fh,Fl

Mit diesem Befehl lassen sich die Tonhöhen ändern. ST ist wie immer der Tongenerator, dessen Frequenz verändert werden soll (Stimme).

K hat mehrere Funktionen

- ist K=1, so wird das Keybit gesetzt, d.h. der Ton beginnt zu erklingen. Dabei wird seine Lautstärke durch die Hüllkurve bestimmt (A,D,S,R). Sobald F=0 ist, das Keybit also gelöscht ist, beginnt der Ton in der Zeit R (siehe Befehl .W) auszuklingen. Dieser Vorgang ist mit dem Anschlagen einer Klaviertaste vergleichbar.

- ist K=2, so wird die Stimme mit der vorgehenden synchronisiert (ST 1 mit ST 3)

- ist K=4, so wird die Stimme mit der vorgehenden ringmoduliert (Summe und Differenz der Frequenzen)

Wollen Sie die Funktionen miteinander kombinieren, so addieren Sie einfach die entsprechenden Werte. (z.B. 7=1 (Keybit) + 2 (Synchr.) + 4 (Ringm.)).

Fh nimmt wie Fl einen Wert zwischen 0 und 255 an. Mit diesen zwei Werten läßt sich die Frequenz ändern. Dabei ist Fh für die Grobeinstellung und Fl für die Feineinstellung der Frequenz zuständig (die Frequenzen finden Sie im Anhang).

.F ST,FA,Fh,Fl,RE

Mit Hilfe dieses schönen Befehls können Sie die Stimme 1, 2, 3 (ST=1,2,4), plus eine externe Stimme (ST=8), durch einen Filter beeinflussen; sollen mehrere Stimmen über den Filter geleitet werden, so addieren Sie einfach die entsprechenden Werte.

Mit FA wird die Filterart festgelegt (FA=16 Tiefpass, FA=32 Bandpass, FA=64 Hochpass). Auch hier können Sie die Filterarten kombinieren, indem Sie die Werte addieren.

Mittels Fh (0-255) und Fl (0-7) wird die Filterfrequenz festgelegt. Die Filterfrequenz können Sie folgendermaßen errechnen :

F = (30 + W * 5.8) Hz bzw. W = (F - 30) / 5.8

W bedeutet hierbei die 11-Bit Zahl (Fh,Fl).

1024	512	256	128	64	32	16	8	4	2	1
Fh								Fl		
128	64	32	16	8	4	2	1	4	2	1

Bild 2.2.15-2 : Berechnung der 11-Bit-Zahl W

Zerlegen Sie F in eine Binärzahl. Addieren Sie nun die Werte in der unteren Zeile, bei denen ein Bit gesetzt ist.

RE ist die Resonanzfrequenz (0=gering, 15=hoch). Sie wird verwendet, um bestimmte Abschnitte des Frequenzspektrums hervorzuheben.

.V LS

Diesen Befehl benötigen Sie zur Lautstärkeeinstellung. (LS=0 stumm, LS=15 volle Lautstärke)

.O ST

Mit diesem Befehl (Buchstabe O, nicht null) lassen sich die Stimmen 1 bis 3 (ST=1,2,3) ausschalten.

.P t

Wählen Sie diesen Befehl, so macht der Computer eine Zeitschleife von t/10 Sekunden.

Alle Befehle können Sie im Programm wie auch im Direktmodus verwenden. Selbstverständlich können die Parameter durch Variablen ersetzt werden.

```
10 FOR T=0 TO 317:READ A                                    <002>
11 POKE 49152+T,A:S=S+A                                     <143>
12 NEXT                                                     <142>
13 IF S<>39382 THEN PRINT"{CLR}FEHLER IN DATAS !":
   END                                                      <004>
14 PRINT"{CLR}O.K.":SYS 49152                               <081>
17 DATA 169,11,141,8,3,169,192,141                          <157>
18 DATA 9,3,96,32,115,0,201,46                              <205>
```

```
19 DATA 240,3,76,231,167,32,115,0                          <096>
20 DATA 201,87,208,3,76,69,192,201                         <165>
21 DATA 86,208,3,76,126,192,201,83                         <167>
22 DATA 208,3,76,175,192,201,70,208                        <212>
23 DATA 3,76,211,192,201,79,208,3                          <110>
24 DATA 76,161,192,201,80,208,3,76                         <165>
25 DATA 137,192,76,231,167,32,26,193                       <018>
26 DATA 32,252,192,32,34,193,164,255                       <011>
27 DATA 153,246,0,201,64,208,14,32                         <152>
28 DATA 34,193,166,250,157,3,212,32                        <213>
29 DATA 34,193,157,2,212,32,42,193                         <164>
30 DATA 230,250,32,42,193,198,250,136                      <060>
31 DATA 169,0,153,4,212,185,247,0                          <115>
32 DATA 153,4,212,76,174,167,32,26                         <141>
33 DATA 193,141,24,212,133,253,76,174                      <064>
34 DATA 167,32,26,193,160,100,169,83                       <020>
35 DATA 133,250,198,250,165,250,208,250                    <165>
36 DATA 136,208,243,202,208,238,76,174                     <124>
37 DATA 167,32,26,193,32,252,192,169                       <030>
38 DATA 0,153,4,212,76,174,167,32                          <123>
39 DATA 26,193,32,252,192,32,34,193                        <230>
40 DATA 164,255,89,246,0,164,250,153                       <030>
41 DATA 4,212,32,34,193,164,250,153                        <221>
42 DATA 1,212,32,34,193,153,0,212                          <110>
43 DATA 76,174,167,32,26,193,138,133                       <038>
44 DATA 250,32,34,193,69,253,141,24                        <232>
45 DATA 212,32,34,193,141,22,212,32                        <214>
46 DATA 34,193,141,21,212,32,34,193                        <224>
47 DATA 10,10,10,10,69,250,141,23                          <107>
48 DATA 212,76,174,167,134,255,224,1                       <032>
49 DATA 208,5,160,0,132,250,96,224                         <178>
50 DATA 2,208,5,160,7,132,250,96                           <084>
51 DATA 224,3,208,5,160,14,132,250                         <173>
52 DATA 96,96,32,115,0,32,158,183                          <148>
53 DATA 138,96,32,253,174,32,158,183                       <049>
54 DATA 138,96,32,34,193,10,10,10                          <132>
55 DATA 10,133,251,32,34,193,69,251                        <238>
56 DATA 164,250,153,5,212,96                               <155>
```

Das folgende Programm zeigt eine Demonstration für die Befehlserweiterung, bei der alle Befehle verwendet werden.

```
10 .V15              :REM LAUTST. =15                      <107>
12 .W1,128,11,12,2,7:REM WELLENF. +HUELLKURVE ST1          <117>
14 .W2,16,04,07,2,4 :REM WELLENF. +HUELLKURVE ST2          <072>
16 A=INT(RND(1)*130):IF A<100 THEN 16:REM ZUFALLSTON
   HOEHE D. WELLE                                          <013>
```

```
18 B=INT(RND(1)*050):IF B<020 THEN 18:REM ZUFALLSTON
   LAENGE D. WELLE                                    <088>
20 .F3,16+64,230,7,10:REM FILTER TP+HP+RESONANZ       <164>
22 .S1,1,A,100:GOSUB 30:REM EVENT. MOEWE              <111>
24 .PB         :GOSUB 30:REM EVENT. MOEWE             <248>
26 .S1,0,A,100:GOSUB 30:REM EVENT. MOEWE              <114>
28 GOTO 16                                            <012>
30 C=INT(RND(1)*4):ON C GOTO 32,36,36                 <092>
32 RETURN                                             <142>
34 REM**** MOEWE ****                                 <126>
36 FOR T=90 TO 70 STEP-1                              <164>
38 .S2,1,T,100:REM AENDERN DER TONHOEHE               <110>
40 NEXT T                                             <254>
42 .S2,0,85,100:REM TON AUSKLINGEN                    <058>
44 RETURN                                             <186>
```

3

Ports

3. Ports

Der Commodore 64 hat zum Anschluß externer Geräte sowie zum Experimentieren sechs verschiedene Anschlußmöglichkeiten, die wir nacheinander in den folgenden Kapiteln besprechen wollen. Es ist jeweils eine Skizze mit den Anschlußbelegungen angegeben, anschließend ist eine Liste aufgeführt, aus der Sie entnehmen können, welche Leitungen des Anschlusses mit welchen Anschlüssen im Computer verbunden sind.

Angegeben ist jeweils noch der Typ des Anschlusses, wobei in Frage kommen:

dE = digitaler Eingang
dA = digitaler Ausgang
aA = analoger Ausgang
aE = analoger Eingang
SV = Stromversorgung

Oft können die Anschlüsse sowohl als Eingang als auch als Ausgang programmiert werden; die Datenrichtung, die nicht standardmäßig vorgesehen ist, ist in Klammern angegeben.

Zu bemerken ist, daß ein digitaler Eingang als logisch 1 gelesen wird, wenn der Anschluß offen ist.

Wollen Sie mit diesen Anschlüssen experimentieren, müssen Sie einen entsprechenden Steckkontakt im Fachhandel erwerben. Auf diesen Steckkontakten finden Sie dann die gleichen Nummern wie hier angegeben. Wenn nicht besonders angegeben, gelten folgende elektrische Belastungsgrenzen für die Anschlüsse: Digitale Eingänge vertragen eine Spannung von -0,5 Volt bis +5,5 Volt. Digitale Ausgänge vertragen eine Standard-TTL-Last, d.h. Sie schalten am besten zur Sicherung Ihres Computers einen Treiberbaustein zwischen den Anschluß und Ihre Schaltung. Ein Treiberbaustein für sechs Leitungen kostet zur Zeit unter einer Mark und ist doch wesentlich billiger zu ersetzen als ein hochintegrierter Baustein im Rechner.

3.1 User-Port

Die hardwaremäßig am leichtesten zu handhabende Schnittstelle ist der User-Port. Wir wollen Ihnen im folgenden Kapitel zeigen, was man damit anfangen kann. Die meisten Leitungen am User-Port sind mit dem im 64er verwendeten Ein-/ Ausgabebaustein 6526 verbunden, der im Kapitel 5.1.5 besprochen wird. Wir zeigen hier, wie die Anschlüsse verbunden sind.

Anschlußbelegung des Userports

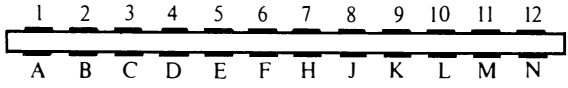


Abbildung 3.1-1 (Steckeranschluß)

Bild 3.1-1 : Steckeranschluß des User-Ports

Mechanisch ist der Userport als Direktstecker mit 24 Kontakten realisiert. Die unteren zwölf Anschlüsse sind völlig analog zu den Anschlüssen der anderen Commodore-Rechner, z.B. Commodore 3032. Vier Anschlüsse führen Masse, und drei Anschlüsse dienen zur Versorgung externer Geräte, so daß nur noch 17 Anschlüsse digitale Funktionen erfüllen. Die Anschlüsse im einzelnen:

Pin	Bezeichnung	Funktion	Typ
1	GND	Masse	SV
2	+5V	Stromversorgung für externe Geräte (100 mA)	SV
3	RESET	entsprechende Leitung des des Prozessors	dA/dE
4	CNT1	CNT-Anschluß des CIA1	dE/dA
5	SP1	SP -Anschluß des CIA1	dA/dE
6	CNT2	CNT-Anschluß des CIA2	dE/dA
7	SP2	SP-Anschluß des CIA2	dA/dE

8	PC2	PC-Anschluß des CIA2 (geht für einen Takt auf Low, wenn ein Schreib- oder Lesezugriff auf das Portregister B ausgeführt wurde)	dA
9	ATN	ATN des seriellen Bus	dA
10	9 V AC	Wechselstrom f. externe Geräte	SV
11	9 V AC	Wechselstrom f. externe Geräte	SV
12	GND	Masse	SV
A	GND	Masse	SV
B	FLAG2	FLAG-Anschluß des CIA2	dE
C	PB0	Port B Bit 0 von CIA 2	dE(dA)
D	PB1	Port B Bit 1 von CIA 2	dE(dA)
E	PB2	Port B Bit 2 von CIA 2	dE(dA)
F	PB3	Port B Bit 3 von CIA 2	dE(dA)
H	PB4	Port B Bit 4 von CIA 2	dE(dA)
J	PB5	Port B Bit 5 von CIA 2	dE(dA)
K	PB6	Port B Bit 6 von CIA 2	dE(dA)
L	PB7	Port B Bit 7 von CIA 2	dE(dA)
M	PA2	Port A Bit 2 von CIA 2	dE(dA)
N	GND	Masse	SV

Fast alle digitalen Anschlüsse sind also mit einem der im Kapitel 5.1.5 besprochenen CIA's verbunden. Zu beachten ist, daß einige Funktionen und Leitungen der CIA's vom 64er bereits für interne Aufgaben benötigt werden und dies bei der Programmierung berücksichtigt werden muß.

Vom CIA 1 sind belegt:

Port A und Port B	dienen zur Tastaturabfrage bzw. zur Abfrage der Joysticks.
FLAG	ist mit dem Eingabeanschluß des Kassettenports verbunden.
Timer A	wird für die Erzeugung des 1/60 Sekunden-Interrupts benötigt.

Die anderen Funktionen bzw. Leitungen des CIA 1 können Sie selbst programmieren. Wenn Sie die Interrupt-Möglichkeiten des CIA 1 ausnützen wollen, müssen Sie in Ihrer IRQ-Routine zuerst abfragen, wodurch der Interrupt ausgelöst wurde. Beachten Sie bitte dabei, daß das Interrupt-Kontrollregister durch einen Lesevorgang gelöscht wird.

CIA 2:

Die Leitungen PA0 und PA1 bestimmen den Arbeitsbereich des Video-Controllers. PA3 bis PA7 werden für den seriellen Bus benötigt. Die freie Leitung PA2 ist an den User-Port

herausgeführt. Der Rest steht zu Ihrer eigenen Programmierung zur Verfügung.

Während eine Übereinstimmung vom Interrupt-Daten- und Maskenregister des CIA 1 einen IRQ-Interrupt auslöst, löst sie im CIA 2 einen NMI aus. Da ein NMI auch durch die RESTORE-Taste ausgelöst werden kann, müssen Sie in Ihrer NMI-Routine eine entsprechende Abfrage einbauen.

Bei Verwendung der RS-232-Schnittstelle wird der CIA 2 fast vollständig belegt, so daß Sie dann im allgemeinen keine Anwenderschaltung mehr anschließen können.

Anschließen von Experimentierschaltungen an den User-Port

Kleinere Schaltungen können bis zu einer Stromstärke bis ca. 100 mA mit der am User-Port zur Verfügung gestellten +5 Volt-Anschluß versorgt werden. Aus den beiden 9 Volt-Wechselstrom-Anschlüssen auch andere Spannungen ableiten. Beachten Sie, daß diese Wechselspannung nicht potentialfrei ist, da sie über einen Gleichrichter mit der 5 V - Gleichspannung gekoppelt ist.

Ein einfaches Beispiel und zugleich eine hilfreiche Schaltung ist das Anschließen von sechs Leuchtdioden mit Hilfe eines einzigen ICs am User-Port. Die Leuchtdioden werden dabei über TTL-Inverter (7406) gesteuert. Der andere Pol der Leuchtdioden liegt an +5 V. Liegt nun an einem Eingang eine logische 1, so liegt am Ausgang eine 0, und die Diode leuchtet. Die verwendeten Widerstände begrenzen den Diodenstrom und können bei Bedarf auch noch variiert werden.

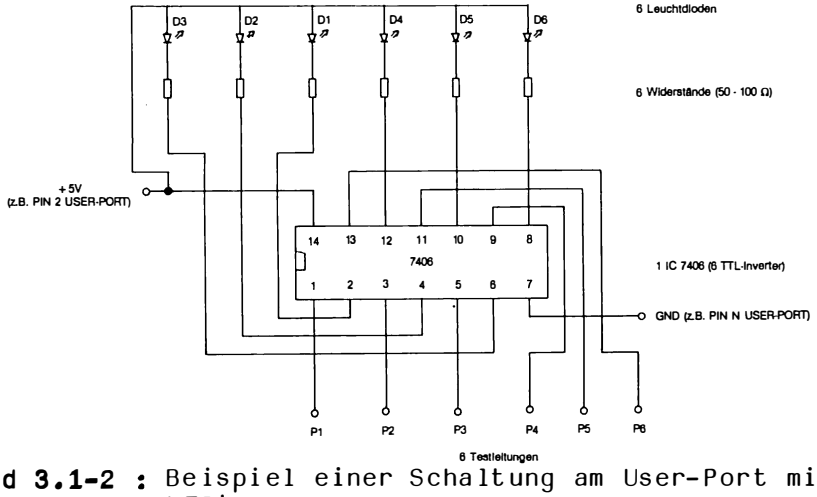


Bild 3.1-2 : Beispiel einer Schaltung am User-Port mit LED's

3.2 Control-Port

Dieser Anschluß wird normalerweise für Joysticks, Paddles und den Light-Pen verwendet. Die Verwendung von Joysticks ist in Kapitel 2.1.1, die Verwendung von Paddles in Kapitel 2.1.2 beschrieben. Jeder der beiden Controlports ist realisiert als ein neunpoliger Stecker:

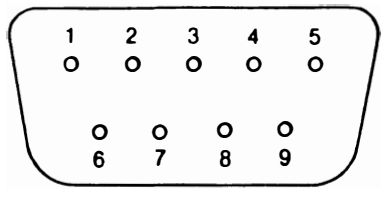


Bild 3.2-1 : Steckeranschluß der Control-Ports

Controlport 1

Pin	Belegung	Typ	Baustein
1	Joystick A oben	dE(dA)	CIA1 PBO
2	Joystick A unten	dE(dA)	CIA1 PB1
3	Joystick A links	dE(dA)	CIA1 PB2
4	Joystick A rechts	dE(dA)	CIA1 PB3
5	Paddle A 1	aE	SID Pin 23 über AS
6	Feuerknopf A / Light-Pen	dE(dA)	CIA1 PB4 und VIC LP
7	+ 5 V (maximal 100 mA)	SV	
8	Masse	SV	
9	Paddle A 2	aE	SID Pin 24 über AS

(AS = Analogschalter)

Controlport 2

Pin	Belegung	Typ	Baustein
1	Joystick B oben	dE(dA)	CIA1 PA0
2	Joystick B unten	dE(dA)	CIA1 PA1
3	Joystick B links	dE(dA)	CIA1 PA2
4	Joystick B rechts	dE(dA)	CIA1 PA3
5	Paddle B 1	aE	SID Pin 23 über AS
6	Feuerknopf B	dE(dA)	CIA1 PA4
7	+ 5 V (maximal 100 mA)	SV	
8	Masse	SV	
9	Paddle B 2	aE	SID Pin 24 über AS

Die Anschlüsse für den Joystick und die Feuerknöpfe sind einfache digitale Anschlüsse, die mit entsprechenden Anschlüssen des CIA 1 verbunden sind. Im Joystick passiert nichts anderes, als daß der entsprechende Kontakt mit Masse kurzgeschlossen wird. Die Anschlüsse für die Paddles werden abgefragt, indem der Wert des Widerstandes, der zwischen +5 V und dem Paddle-Anschluß liegt, gemessen wird. Die Auswertung erfolgt optimal, wenn der Widerstand sich im Bereich von 200 Ohm bis 200 KOhm bewegt. Wichtig ist noch, daß gleichzeitig nur ein Paddle-Paar abgefragt wird. Die beiden Paddlepaare sind über einen Analogschalter geführt, den man im Bedarfsfall umstellen kann (siehe Kapitel 2.1.2).

3.3 Expansion-Port

An diesem Anschluß sind der Adreßbus und der Datenbus des Prozessors direkt herausgeführt, außerdem sind einige andere wichtige Signale verfügbar. Für Hardware-Spezialisten bietet sich mit dieser Schnittstelle ein nahezu unbegrenztes Betätigungsfeld. Schließlich wird die für den Commodore 64 verfügbare Z-80-Karte ja auch an diese Schnittstelle angeschlossen.

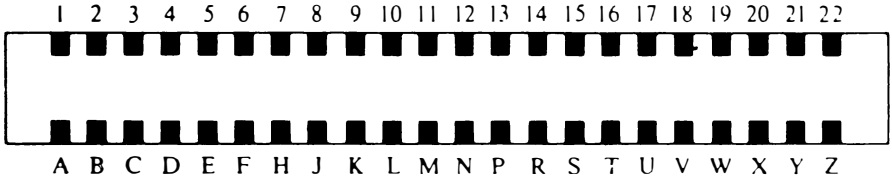


Bild 3.3-1 : Steckeranschluß des Expansion-Ports

Pin	Signal	Bedeutung
1	GND	Masse
2	+ 5 Volt	Versorgungsspannung
3	+ 5 Volt	Versorgungsspannung
4	$\overline{\text{IRQ}}$	IRQ-Interruptleitung des 6510
5	R/$\overline{\text{W}}$	Schreib-/Leseleitung
6	Dot Clock	Dot-Frequenz 7.88 MHz
7	$\overline{\text{I/O 1}}$	Auswahlanschluß für zusätzlichen Peripheriebaustein im Adreßbereich $DE00 bis $DEFF
8	$\overline{\text{GAME}}$	Anschluß zum Adreßmanager; dient u.a. zum Ausblenden des Basic-ROM's.
9	$\overline{\text{EXROM}}$	Wie oben, jedoch zum Einblenden eines externen ROM in den Bereich $8000 bis $A000
10	$\overline{\text{I/O 2}}$	Wie 7, jedoch im Adreßbereich $DF00 bis $DFFF
11	$\overline{\text{ROML}}$	Selektionsbit für externe ROM's im Bereich $8000 bis $9FFF
12	BA	Bus available = Bus verfügbar (für DMA-Betrieb)
13	$\overline{\text{DMA}}$	DMA-Anforderungs-Eingang

14	D7	Datenbus Bit 7
15	D6	Datenbus Bit 6
16	D5	Datenbus Bit 5
17	D4	Datenbus Bit 4
18	D3	Datenbus Bit 3
19	D2	Datenbus Bit 2
20	D1	Datenbus Bit 1
21	DO	Datenbus Bit O
22	GND	Masse
A	GND	Masse
B	$\overline{\text{ROMH}}$	Selektiert externe ROM's im Bereich ab $A000
C	$\overline{\text{RESET}}$	RESET-Leitung des 6510
D	$\overline{\text{NMI}}$	NMI-Interrupt-Anschluß des 6510
E	PHI2	Taktleitung des 6510
F	A15	Adreßbus Bit 15
H	A14	Adreßbus Bit 14
J	A13	Adreßbus Bit 13
K	A12	Adreßbus Bit 12
L	A11	Adreßbus Bit 11
M	A10	Adreßbus Bit 10
N	A9	Adreßbus Bit 9
P	A8	Adreßbus Bit 8
R	A7	Adreßbus Bit 7
S	A6	Adreßbus Bit 6
T	A5	Adreßbus Bit 5
U	A4	Adreßbus Bit 4
V	A3	Adreßbus Bit 3
W	A2	Adreßbus Bit 2
X	A1	Adreßbus Bit 1
Y	AO	Adreßbus Bit O
Z	GND	Masse

3.4 Serial-Port

Über diesen Anschluß verkehren die Floppy, der Drucker und auch andere Peripheriegeräte mit dem Rechner. Alle diese Geräte sind am gleichen Bus angeschlossen. Das Prinzip dieses Busses ist ähnlich dem des IEC-Bus. Letzterer besitzt acht Daten-Leitungen, fünf Management-Leitungen und

drei Handshake-Leitungen. Der serielle Bus des 64er muß aus Kostengründen mit weniger Leitungen auskommen, so daß die Datenübertragung nicht mehr parallel sondern seriell erfolgt und außerdem auf ein paar Management-Leitungen verzichtet werden muß. Die Belegung des Steckanschlusses ist wie folgt:

Pin	Belegung	verbunden mit
1	SRQ (nicht unterstützt)	CIA1 FLAG
2	Masse	Masse-Leitung
3	ATN	CIA2 PA3 über Inverter
4	CLK (in/out)	in: CIA2 PA6
		out: CIA2 PA4 über Inverter
5	DATA (in/out)	in: CIA2 PA7
		out: CIA2 PA5 über Inverter
6	RESET	Reset-Leitung

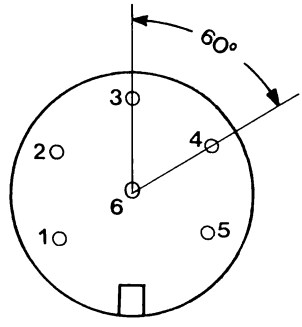


Bild 3.4-1 : Steckanschluß des Serial-Port

Die Ausgänge für DATA, CLK und ATN sind über Inverter geführt, damit der CIA bei einem eventuellen Kurzschluß keinen Schaden leidet. Der Ausgang des Inverters für ATN ist auch am Userport Pin 9 verfügbar. Der Anschluß SRQ ist mit dem Kassettenrekorder-Read-Anschluß verbunden; jedoch wird der SRQ-Anschluß von keinem Commodore-Gerät benützt und auch vom Rechner nicht abgefragt. Beim IEC-Bus ist diese Leitung für den Fall vorgesehen, daß z.B. ein Drucker meldet, daß kein Papier mehr vorhanden ist (Service Request).

Funkionsweise des seriellen Bus (schematisch)

Der Informationsablauf auf dem Bus wird von einem Controller (hier also unserem Rechner) gesteuert. Obwohl der Bus

bidirektional ist (Daten können sowohl in eine als auch in die andere Richtung fließen), darf nur ein Controller am Bus hängen. Jedes Gerät am Bus (außer dem Controller) hat eine eigene Nummer, die sogenannte Primäradresse. Über diese Primäradresse kann der Controller Kommandos an die Geräte verteilen. Die sogenannte Sekundäradresse dient zum Ansprechen von Untereinheiten im angesprochenen Gerät wie z.B. dem Fehlerkanal der Floppy. Mögliche Kommandos sind:

Talk : Aufforderung an Gerät, ab jetzt Daten zu senden.

Untalk : Aufforderung an Gerät, mit dem Senden von Daten aufzuhören.

Listen : Aufforderung an Gerät, Daten zu empfangen

Unlisten: Aufforderung an Gerät, ab nun keine Daten mehr zu empfangen.

Die Kommandos Unlisten und Untalk wirken jeweils auf alle angesprochenen Peripherie-Geräte. Zur Unterscheidung von Kommandos von normalen Daten dient die Leitung ATN. Ist sie 'wahr', werden Kommandos gesendet. Auch die Sekundäradresse wird mit ATN = 'wahr' gesendet. Dabei gibt es folgende Besonderheiten:

1. Sekundäradressen größer als 128 werden nicht als solche gesendet

2. Bei Sekundäradressen nach Talk muß Bit 5 und Bit 6 gesetzt sein

3. Sind nach einem Listenkommando Bit 4 bis 7 der Sekundäradresse gesetzt, erwartet das angesprochene Gerät nachfolgend einen Dateinamen.

4. Ist nach einem Listenkommando in der Sekundäradresse das Bit 7 null, aber Bit 6, Bit 5 und Bit 4 gleich 1, so wird die Datei auf dem Peripherie-Gerät geschlossen.

Mit diesen Angaben dürfte es Ihnen möglich sein, mit Hilfe der Tabelle in Kapitel 5.2.1 unter Benutzung der Routinen ACPTR, CIOUT, LISTEN, READST, SECOND, TALK, TKSA, UNLSN

und UNTALK den seriellen Bus zu bedienen. Sie sollten aber in jedem Fall aufpassen, daß Sie zuerst ein Gerät abmelden, bevor Sie ein neues anmelden.

Die Unlisten- und Untalk-Kommandos sind übrigens nichts anderes als Listenkommandos mit der Geräteadresse 63 bzw. 95.

Außer mit den oben aufgeführten Kernal-Routinen können Sie den seriellen Bus natürlich auch über logische Dateien, also ähnlich wie Basic, oder direkt von Basic aus bedienen.

3.5 Audio-Video-Port

An diesen Anschluß sind 3 Video- und 2 Audio-Anschlüsse zusammengefaßt. Die Steckerbelegung im Handbuch ist falsch, weil dort statt des achtpoligen DIN-Steckers nur ein fünfpoliger abgebildet wurde und die Farbinformation am nicht abgebildeten mittleren Anschluß abzugreifen ist.

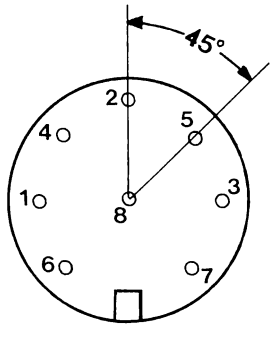


Bild 3.5-1 : Steckanschluß des Audio-Video-Port

Pin	Belegung
1	Luminanz
2	Masse
3	Audio-Ausgang
4	Hilfssignal für Video
5	Audio-Eingang
8	Chrominanz

Für schwarz/weiß-Monitore reicht es, die Video-Information zwischen Pin 1 und Pin 2 abzunehmen.

Der Audio-Ausgang ist der mit einer Transistorstufe verstärkte Ausgang des SID. Der Audio-Eingang ist über einen Kondensator direkt mit dem Eingang des SID verbunden.

3.6 Kassetten-Port

Bei diesem zwölfpoligen Platinen-Direktstecker sind jeweils die oberen Anschlüsse mit den unteren verbunden. Die Belegung ist:

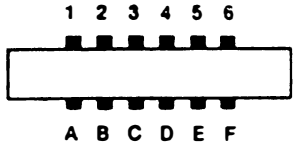


Bild 3.6-1 : Steckanschluß des Kassetten-Port

Pin	Belegung	verbunden mit
A/1	Masse	
B/2	+ 5 Volt	
C/3	Motorsteuerung	Prozessor-Port Bit 5
D/4	Read-Leitung	CIA1 FLAG
E/5	Write-Leitung	Prozessor-Port Bit 3
F/6	Sense-Leitung	Prozessor-Port Bit 4

Zwischen dem Prozessor-Port Bit 5 und dem Anschluß C/3 am Kassetten-Port ist eine Transistorstufe zwischengeschaltet, die die Versorgungsspannung für den Motor des Kassettenrekorders ein- und ausschaltet. Die Sense-Leitung fragt ab, ob eine Taste am Rekorder gedrückt ist. Dem Anschluß C/2 ist ein RC-Glied vorgeschaltet wodurch u.a. verhindert wird, daß durch ein Kurzschluß an diesem Anschluß der Rechner zerstört wird.

4

In der Kürze liegt die Würze — Tips und Tricks

4. In der Kürze liegt die Würze - Tips und Tricks

In dem folgenden Kapitel haben wir für Sie einige interessante Kleinigkeiten zusammengefaßt. In den meisten Fällen sind es Hilfsmittel, die Ihnen den Umgang mit Ihrem Commodore 64 erleichtern sollen. In der Regel sind es hilfreiche PEEK's und POKE's, aber auch SYS-Aufrufe und kleine Programme. Im zweiten Unterkapitel gehen wir kurz auf die Kommunikation mit anderen Commodore-Rechner ein. Der VC 20 wurde dabei ausgespart, weil der Grad der Inkompatibilität zu hoch ist.

4.1 Diverse PEEK's, POKE's und SYS-Aufrufe

Allgemeine Anmerkung: Die folgenden Tips sind nach dem wichtigsten Schlagwort alphabetisch gegliedert, damit ein leichtes Auffinden ermöglicht wird. Der Begriff ist jeweils in Großbuchstaben angegeben.

ASCII-CODE in Bildschirmcode umwandeln

Mit den ersten beiden Zeilen des folgenden Programmes wird ein in A vorliegender ASCII-Code in den Bildschirmcode des Commodore 64 umgewandelt, der Rest stellt ein Testprogramm dar:

```
 4 A(0)=128 : A(1)=32 : A(2)=0 : A(3)=64 : A(4)=192 :
              A(5)=96 : A(6)=64 : A(7)=96
 8 DEF FNB(X)=A(X/32)+(XAND31)+33*(X=255)

10 FOR X=25TC 255
15 PRINT"(CLR)";X,FNB(X),CHR$(34),CHR$(X)
20 POKE 1048,FNB(X)
```

```
25 FOR W=1 TO 200 : NEXT
30 NEXT
```

In der ersten Zeile wird eine Umrechnungsvorgabe in einem Feld festgelegt, die in der zweiten Zeile verwertet wird. Bei der Umrechnung können jeweils Bereiche von 32 Zeichen als ganzes behandelt werden (X AND 31), wenn man von dem letzten Zeichen (PI; ASCII=255 = Bildschirmcode=94) absieht. Näheres ist aus folgender Tabelle ersichtlich:

Bits im ASCII-Code			Bit im Bildschirmcode		
128	64	32	128	64	32
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	0	1	1

Man sieht leicht, daß die Zuordnung nicht umkehrbar eindeutig ist.

BILDSCHIRM-RAM und Zeichensatz verlegen

Das Register 53272 ist für die Position des Bildschirm-RAM und den Zeichensatz zuständig. Dabei gilt

1. Halbbyte (Bits 7-4) * 1024 =
   Startadresse Bildschirm-RAM
2. Halbbyte (Bits 3-0) * 1024 =
   Startadresse Zeichengenerator

Achtung: Bit 0 wird nicht berücksichtigt, so daß sich folgende Möglichkeiten ergeben:

Wert in 53272	Startadresse
xxxx000x	0
xxxx001x	2048
xxxx010x	4096
xxxx011x	6144
xxxx100x	8192
xxxx101x	10240
xxxx110x	12288
xxxx111x	14336

Beispiel: POKE 53272, 18 (=1*16+2) ergibt

Startadresse Bildschirm-RAM : 1024
Startadresse Zeichensatz : 2048

Achtung: für das Bildschirm-RAM muß auch die Adresse 648 geändert werden.

POKE 648, (Startadresse)/256

Relative Adresse zum VIC-Arbeitsbereich, der in 56576 definiert ist.

CURSOR an/aus

```
POKE 204,0              an
POKE 207,0:POKE204,1    aus
```

CURSORGESCHWINDIGKEIT ändern

```
POKE 56325,X            X - gewünschte Geschwindigkeit
```

1 - schnell,..., 255 langsam
0 - Bildschirmausgabe verlangsamen
52 - normaler Zustand, seltene Werte zwischen 51 und 59 in unregelmäßigen Abständen

Achtung: Da sich die Interrupt-Geschwindigkeit ändern kann, geht die TI-Uhr falsch.

Schließen aller offenen DATEIEN

SYS 65511

Die Dateien werden dabei nur im Rechner geschlossen; auf der Floppy können sie weiterhin offen bleiben; dort kann man sie mit

OPEN1,8,15:CLOSE1

schließen.

Anzahl der offenen DATEIEN

PEEK(152)

DAUERFUNKTION für Tasten

POKE 650, x

x = 128 : alle Tasten
x = 0 : normal (Cursor und Leertaste)
x = 64 : alle Tasten ohne Dauerfunktion

Zeilen automatisch löschen - DELETE

Wer nach einem teilweise Umarbeiten eines Programmes viele Zeilen zuviel hat, wird sich sehnlichst eine DELETE-Routine wünschen. Im folgenden stellen wir eine Basic-Routine vor; etwas ausführlicher, weil sie einige Tricks beinhaltet. Sicherlich geht es auch kürzer. Verwendet wird der Tastaturpuffer und der Puffer für den Kassettenrekorder.

Prinzip des Programmes:

- Zeilennummern, die gelöscht werden sollen, nacheinander auf den Bildschirm bringen
- Programm verlassen

- Zeile löschen
- Programm 'mittendrin' wieder neu starten, natürlich mit der nächsten zu löschenden Zeile

Kurz gesagt: den Handbetrieb automatisieren

```
1 GOTO60100
60000 INPUT"VON";VO
60010 INPUT"BIS";B1
60020 INPUT"SCHRITTWEITE";SR
60030 POKE831,VO AND 255
60040 POKE832,VO/256
60050 POKE833,BI AND 255
60060 POKE834,BI/256
60070 POKE835,SR
60100 VO=PEEK(832)*256+PEEK(831)
60110 BI=PEEK(834)*256+PEEK(833)
60120 SR=PEEK(835)
60130 PRINT"(CLR)";VO
60140 IF VO GE BI THEN STOP
60150 VO=VO+SR
60160 POKE831,VO AND 255
60170 POKE832,VO/256
60180 POKE631,145    : REM CURSOR NACH OBEN
60190 POKE632,145    : REM CURSOR NACH OBEN
60200 POKE633,145    : REM CURSOR NACH OBEN
60210 POKE634,13     : REM CHR$(13)
60220 POKE635,82     : REM R
60230 POKE636,85     : REM U
60240 POKE637,78     : REM N
60250 POKE638,13     : REM CHR$(13)
60260 POKE198,8      : REM ZAHL DER ZEICHEN IM PUFFER
```

Zeile 1 wird benötigt, weil der Tastaturpuffer nur zehn Zeichen aufnehmen kann und eine Zeilennummer für ein Programm ziffernweise (genau: zeichenweise) eingegeben werden muß. Die Zeilen von 60000 bis 60070 werden nur zur Erfassung der gewünschten Daten benötigt, wobei die größeren Zahlen (VO, BI) jeweils in der üblichen Form (Low-Byte, High-Byte) in den Kassettenpuffer geschrieben werden.

In Zeile 60100 beginnt die eigentliche 'Programmschleife': Lesen der aktuell zu löschenden Zeile, Endekriterium prüfen, neuen Wert anhand der Schrittweite berechnen, Sichern dieses Wertes, füllen des Tastaturpuffers wie angegeben und Programmabbruch.

Warum die Variableninhalte sichern? Das Löschen einer

Zeile ist eine Programmänderung und löscht somit alle Variablen.

Wenn Sie die Zeile 1 weglassen, können Sie die Programmsequenz mit dem weiter unten beschriebenen Kurz-MERGE einladen, Zeile 1 eintippen und mit RUN 60000 starten. Vielleicht gibt Ihnen das Programm auch ein Tip für Ihre anderen Probleme.

DIRECTORY ohne Programmverlust laden

(siehe auch VIEW in Kapitel 2.2)

```
POKE 44, PEEK (46)+1
LOAD "$",8
POKE 43,1 : POKE 44,8
```

FARBWECHSEL unter dem Cursor

PEEK (647) gibt die Farbe unter dem Cursor an. Dementsprechend kann Sie mit

```
POKE 647, x        x = 0,...,15
```

geändert werden.

FILE-PARAMETER in der Zero-Page

```
PEEK (183)       - Länge des Filenamens
PEEK (184)       - aktuelle Filenummer
PEEK (185)       - aktuelle Sekundär-Adresse
PEEK (186)       - aktuelle Primäradresse
PEEK (187, 188) - Zeiger auf Dateinamen
```

FREIER SPEICHERPLATZ

Die korrekte Ausgabe des freien Speicherplatzes (Basic) erhalten Sie mit

PRINT FRE(X)+65536, wenn FRE(X) negativ

INPUT-Ersatz

Für Textverarbeitungsprogramme und Datenverwaltungsroutinen ist der normale Basic-INPUT-Befehl schlecht geeignet, da Probleme auftauchen, wenn Kommata, Doppelpunkte und Anführungszeichen im Text gemischt auftreten können.

Um diesem Übel abzuhelfen, haben wir den Befehl LEST (Kapitel 2.2.9.4) entwickelt, der sowohl für Eingaben vom Bildschirm als auch von Dateien den INPUT- bzw. INPUT#-Befehl ersetzt, ohne daß die oben genannten Schwierigkeiten auftreten und er auch im Direktmodus verwendet werden kann.

Für die Eingabe vom Bildschirm geht es jedoch auch einfacher mit der folgenden kleine Routine:

```
1000 B$ = ""          : REM Ausgabestring annullieren
1010 SYS 65487        : REM KERNAL-Routine CHRIN ($FFCF)
1020 A=PEEK(780)      : REM Akku = gelesenes Zeichen
1030 IF A=13 THEN RETURN : REM Ende bei 'Return'-Taste
1040 B$=B$+CHR$(A)    : REM in B$ Zeichen sammeln
1050 GOTO 1010        : REM Schleife
```

Diese Routine wird mid GOSUB 1000 aufgerufen und gibt den eingelesenen String in B$ zurück. Hierbei wird kein Fragezeichen - wie bei INPUT üblich - ausgegeben, außerdem steht der Cursor nachher in der alten Zeile. Wenn es notwendig sein sollte, können Sie diesen Mangel aber leicht mit PRINT-Befehlen beseitigen.

Die CHRIN-Routine bewirkt beim ersten Aufruf, daß der Cursor blinkt und die Zeile eingegeben werden kann. Anschließend wird die Routine aufgerufen, um die Zeichen zu holen, wobei das Endekriterium das Auftauchen von CHR$(13) ist.

INVERS-MODUS

```
POKE 199,1          ein
POKE 199,0          aus
```

JOYSTICKSIMULATION über Tastatur

Da Joystick und Tastaturabfrage in einem erledigt werden können, ist es auch möglich, die Joystickeingabe über Tastatur zu simulieren, was allerdings für schnelle Reaktionen einiges Geschick erfordert. Bei untenstehender Tabelle wird auch deutlich, warum die Kollision zwischen Joystick und Tastatur bei Control-Port 2 (Adresse 56320) nicht so gravierend ist: Wer drückt schon im Normalfall die angegebenen Tastenkombinationen?

Controlport 1 (56321)

Leertaste	- Feuer
CTRL-Taste	- links
1	- oben
2	- rechts
LT (Kleinerzeichen)	- unten

Controlport 2 (56320)

CTRL und J	- Feuer
CTRL und D	- links
CTRL und Cursor rechts	- oben
CTRL und G	- rechts
CTRL und A	- unten

Vergleiche auch Kapitel 2.1 und Kapitel 5.8

LÄNGE der momentanen BILDSCHIRMZEILE

Diese kann mit PEEK (213) abgefragt werden

LETZTE ZEILENNUMMER nach Programmabbruch

Haben Sie nach einem Programmabbruch durch die STOP-Taste versehentlich den Bildschirm gelöscht und wollen wissen, welche Zeile die Meldung 'BREAK IN ...' enthalten hat, können Sie diese mit

PRINT PEEK(57) + 256* PEEK(58) bzw. DEEK(57)

erfahren

LISTSCHUTZ ein/aus

POKE 775,200 (ein) POKE 775,167 (aus)

LIST während Programmablauf

```
100 POKE 631,82
110 POKE 632,69
120 POKE 633,212
130 POKE 634,13
140 POKE 198,4
150 LIST
```

```
631 - 633 : Abkürzung für Basic-Befehl RETURN
634       : RETURN-Taste
198       : Anzahl gültiger Zeichen im Tastaturpuffer
```

(vergleiche auch ausführlicheres Beispiel unter DELETE) Das Programmstück ist als Unterprogramm aufzurufen, obwohl kein RETURN am Ende steht. Dies wird durch das Programm selbst übernommen. Sinnvoll ist der Einsatz beim Programmtest.

LOAD ERROR umgehen

```
POKE 45,PEEK(831)
POKE 46,PEEK(832)
CLR
```

oder

```
DOKE 45,DEEK(831):CLR
```

Durch SAVE wird ein Programm zweimal auf Kassette gespeichert. Wird beim Laden eine gewisse Zahl von Fehlern überschritten, tritt ein LOAD ERROR auf, auch wenn das zuerst geladene Programm fehlerfrei ist (z.B. Beschädigung des Bandes, dort wo sich die Kopie befindet). Aber versuchen kann man es immer (vielleicht ist auch ein Fehler in einer Stringkonstanten). Siehe auch Kaltstart, Kapitel 2.1.4.2

Ein einfaches MERGE

- erstes Programm laden
- PRINT PEEK(43), PEEK(44)
- Ergebnis merken (Basic-Anfang)
- POKE 43,(PEEK(45) + 256*PEEK(46)-2) AND255
- POKE 44,(PEEK(45) + 256*PEEK(46)-2) /256
  (Anfangszeiger auf Ende des Programmes setzen)
- zweites Programm laden
- POKE 43, (erste gemerkte Zahl)
- POKE 44, (zweite gemerkte Zahl)
- SYS 42291 (oder SYS$A533) (Programm verbinden)

Achtung: das zweite Programm muß höhere Zeilennummern haben als das erste Programm.

PROGRAMM RETTEN

Folgende Anweisungen funktionieren natürlich nur, wenn Sie ein Programm versehentlich mit NEW oder einem Reset gelöscht haben. In diesem Fall steht das Programm noch im Speicher und nur einige Zeiger wurden durch den Rechner 'umgebogen' (Alle Programmzeilen sind jeweils mit RETURN abzuschließen).

- POKE 46, PEEK(56)-1
- POKE 45, PEEK(55)+247
- CLR
  (Zeiger auf Start der Variablen an Basic-RAM-Ende)
- POKE PEEK(44)*256 + PEEK(43)+1, PEEK(44)
- 63999
- FOR I=PEEK(44)*256+PEEK(43) TO PEEK(46)+256 + PEEK (45):IF PEEK(I) OR PEEK(I+1)ORPEEK(I+2) THEN NEXT
  (! Abkürzungen verwenden ! / Suche nach Programmende: drei aufeinanderfolgende 0-Bytes)
- POKE 45, (I+3)AND 255
- POKE 46, (I+3)/256
- CLR
  (Neues Programmende setzen)

PROGRAMM ZERSTÖREN

Um ein Programm gegen unbefugte Benutzer - etwas - abzusichern, kann man nach einem falsch beantworteten Passwort das Programm sehr leicht mit

```
POKE 776,1
```

zerstören, indem der Vektor für die Basic-Befehlsadresse umgesetzt wird.

Andere Möglichkeit:

```
SYS 64738  (Systemreset)
```

(Siehe auch Listschutz)

RAM-BEREICH SPEICHERN / LADEN

Zum Speichern von Grafiken (sofern diese im zugriffsfähigen RAM liegen) und Maschinenprogrammen ist es oft sinnvoll, RAM-Bereiche direkt zu speichern. Dies geht mit:

```
SYS 57812  "Name",8
POKE 193, (Anfangsadresse Low Byte)
POKE 194, (Anfangsadresse High Byte)
POKE 780, 193
```

```
POKE 781, (Endadresse Low Byte)
POKE 782, (Endadresse High Byte)
SYS 65496
```

Das Gegenstück dazu ist das Laden an eine gewünschte Adresse:

```
SYS 57812 "Name",8
POKE 780,0
POKE 781, (Zieladresse Low Byte)
POKE 782, (Zieladresse High Byte)
SYS 65493
```

Anschließend kann mit

```
PEEK (782)*256 + PEEK(781)
```

die letzte geladene Adresse + 1 abgefragt werden.

Software - RESET

```
SYS 64738
```

bewirkt einen Sprung auf die Initialisierungsroutine, die beim Einschalten angesprungen wird.

Hardware - RESET

a) Pin 1 und Pin 3 am User-Port verbinden (GND und Reset)
b) Pin 2 und Pin 6 am seriellen Bus verbinden (erreichbar am Floppy-Stecker)

Nicht vergessen: Basic-Zeiger sind im Urzustand und müssen für das aktuelle Programm gesetzt werden.

SPEICHERN auf Kassettenrekorder und Floppy VERHINDERN

Kassettenrekorder

```
POKE 0,7     gaukelt dem Benutzer nur das Abspeichern
             vor. Probieren Sie LOAD!
POKE 0,47    Normalzustand
```

Floppy

```
POKE 819,253   ergibt Missing File-Name
POKE 818,253
POKE 808,225   STOP-Taste ausschalten

POKE 819,253   ergibt den den Assembler-Program-
POKE 818,34    mierern hinlänglich bekannten Befehl
POKE 808,225   BIF (Branch in Forest)

POKE 818,237   Normal-Modus
POKE 819,245
POKE 808,237
```

Bemerkung: in 818/819 steht der SAVE-Vektor
in 808/809 steht der STOP-Vektor

STATUSABFRAGE

```
PRINT  ST
```

oder

```
PRINT PEEK(144)
```

gibt den Gerätestatus aus (zu den Bedeutungen vgl. Kapitel 5).

RUN / STOP aus/ein

```
POKE 788,52  (aus)          POKE 788,49   (ein)
```

RUN/STOP-RESTORE aus/ein

```
POKE 808,225   (aus)          POKE 808,237   (ein)
```

STOP-Taste sperren (andere Möglichkeit)

```
POKE 830, 169          LDA
POKE 831, 1                #1
POKE 832, 96           RTS

POKE 808,62
POKE 809,3             (830)

(DOKE 808,830)
```

Das Betriebssystem setzt durch die Stop-Taste das Zero-Flag in einem kleinen Programm. Durch obige Routine wird ein Unterprogramm in den Kassettenpuffer geschrieben, das immer das Zero-Flag löscht. Die Speicherzellen 808 und 809 zeigen auf die Betriebssystem-Routine.

STOP-Taste freigeben

```
POKE 808,237       Anfangsadresse
POKE 809,246       Betriebssystem-Routine eintragen

(DOKE 808,63213)
```

Synthetische STEUERZEICHEN (Finkeln)

Steuerzeichen sind in normalem Text eingebettete Zeichen, die nicht ausgegeben werden, sondern eine andere Aktion veranlassen, wie z.B. Bildschirm löschen (inverses Herz)

oder die Cursor-Bewegungen.

Neben der normalen Tastenfunktion (z.B. durch CLR/HOME) können diese Zeichen aber auch synthetisch erzeugt werden. Für ein vorhandenes Steuerzeichen könnte man dies wie folgt durchführen:

Eingabe	Wirkung
?""	Leere Zeichenreihe ausdrucken. Beachten Sie: Sie sind nicht mehr im "Texteingabemodus".
DEL-Taste	hinteres ""-Zeichen löschen
RVS-ON-Taste	Invers-Modus direkt einschalten
Q	Steuerzeichen für Cursor nach hinten eingeben
RVS OFF-Taste	Invers-Modus direkt ausschalten
RETURN-Taste	Übergabe an den Rechner

Probieren Sie es auch mit der Eingabe 'S' (Cursor in linke, obere Bildschirmecke / HOME

Hier eine Liste von synthetischen Steuerzeichen, die auch mit PRINT CHR$(X) bewirkt werden können:

Zeichen	CHR$-Code	Bedeutung
H	8	Umschaltung Groß/Kleinschrift durch C= und SHIFT wird gesperrt
I	9	Gegenteil zu H
M	13	RETURN; Rest der Zeile wird nicht mehr ausgegeben
N	14	Umschalten auf Kleinschrift
T	20	ersetzt DEL-Taste
SHIFT + M	141	SHIFT RETURN z.B. Drucken des Textes ab diesem Zeichen in der nächsten Zeile
SHIFT + M R		Ausführen der Steuerzeichen auch im Listing, in diesem Fall REVERSE ON. Probieren Sie auch andere Steuerzeichen
SHIFT + N	142	Umschalten auf Großbuchstaben
SHIFT + T	148	ersetzt INST-Taste

Gedrückte TASTE

```
PEEK (203)
```

Characterwert der letzten gedrückten TASTE

```
PEEK (215)
```

Umschalten auf GROSS/KLEINSCHRIFT

```
PRINT CHR$(14)  - Groß/Kleinschrift
PRINT CHR$(142) - Normalmodus
```

WARTESCHLEIFEN

Neben dem bekannten

```
10 GET A$ : IF A$ = "" THEN 10
```

gibt es noch die Version

```
10 GET A$ : ON -(A$="") GOTO 10
```

die den Rest der Zeile (bei knappem Platz) auch noch nutzbar macht, weil die IF-Abfrage wegfällt. Auch möglich

```
WAIT 198,1 : GETA$  (Taste gedrückt)
```

Auf den Joystick wartet

```
WAIT X,127,127   mit X=56320 für Port 2
                 und X=56321 für Port 1
```

ZEICHENFARBE ändern

Wer die CTRL-Kombinationen umgehen möchte, und damit auch die Steuerzeichen im String, kann auch die Adresse 646 benutzen:

```
POKE 646, (Farbcode)
```

ZEILE auf dem Bildschirm LÖSCHEN

```
POKE 781,Z
SYS  59903
```

ZEITERSPARNIS bei aufwendigen Berechnungen

Haben Sie auch schon festgestellt, daß der Bildschirm nichts anzeigt, wenn ein Programm von Kassette geladen wird? Da der Rechner den VIC nicht benötigt, wird er einfach abgeschaltet. Da Eingaben von der Tastatur somit auch sinnlos sind und Timer etc. nicht gebraucht werden, kann man auch den IRQ abschalten. Zusammen ergibt dies:

```
POKE 53265, PEEK(53265) AND 239
```

(VIC ausschalten)

```
POKE 56333,1
```

(IRQ abschalten)

und das ganze zurück:

```
POKE 53265, PEEK(53265) OR 16 (VIC)
POKE 56333,129
```

Ein kleines Problem: Die Tastatur geht nicht mehr.

4.2 Programmaustausch mit anderen Commodore-Rechnern

Im diesem Kapitel sollen Hinweise zur Nutzung des Commodore 64 mit Programmen der Rechner 20xx, 30xx, 80xx gegeben werden. Da Programme der alten Serien 2000/3000 übernommen werden können, und als reine Basic-Programme im Prinzip sofort lauffähig sind, soll hier auf die Übernahme solcher Programme eingegangen werden. Die Übernahme von Basic-Programmen der Serien 3000/4000/ 8000 auf den Commodore 64 ist unproblematisch, da der Commodore 64 die Programme selbst anpaßt. Wegen der anderen Diskettenformate kann eine Übertragung nur mit dem Kassettenrekorder erfolgen.

Serie 2000 auf Commodore 64

Die Programmdateien beim Commodore 64 sind gegenüber der 2000er Serie um ein Byte kürzer. Deshalb ist mit den Anweisungen (im Direkt-Modus nach Laden des Programms)

```
POKE 2051,PEEK(2052)
POKE 2052,PEEK(2053)
POKE 2053,58
```

schon die Lauffähigkeit hergestellt. Dies gilt natürlich nur für reine Basic-Programme. In den ersten beiden Zeilen wird die Anfangsadresse des Basic-Programmes um eine Adresse vorgezogen, und in der dritten Zeile wird das 'frei' gewordene Zeichen in der ersten Programmzeile mit einem ':' besetzt. Diesen Doppelpunkt kann man dann im Basic-Editor nach LIST ganz normal löschen, wenn er stört.

Commodore 64 auf Serien 8000 und 3000

Bei Übernahme von Programmen auf Rechner der Serie 8000 und 3000 sieht die Sache etwas anders aus. Nach dem Laden des Programmes über den Kassettenrekorder ruft man zunächst den eingebauten Monitor mit

```
SYS 1024
```

auf, und läßt sich die Speicherzellen 1024 bis 1030 (dezimal) anzeigen mit

```
.m 0400 0400
```

Anschließend ändert man die Anzeige wie folgt ab:

```
. 0400 00 01 08 00 00 3A 00 aa
```

Damit wird der Programmanfang auch für die 8000/3000er Serie auf das beim Commodore 64 übliche 2. Kilobyte des RAMs gelegt, und eine Zeile 0 eingefügt, die einen ':' enthält.

Dann kann man mit

```
.x
```

den Monitor verlassen und mit

```
0      (Return-Taste)
```

die Zeile 0 gleich wieder löschen. Mit

```
LIST
```

hat man nun das gleiche Programm wie auf dem Commodore 64. Durch die Verschiebung nach vorne (Beginn in Adresse 1024 statt 2048) zeigt der Vektor auf das Ende des Programmes um ein KByte zu weit nach hinten. Bei mehrmaligem übertragen zwischen den Rechner wird das Programm für das Betriebssystem immer größer. Dies kann man umgehen mit:

```
POKE43, PEEK(43)-4
```

indem der Wert für das Higher Byte um vier Blöcke verkleinert wird.

5

Tabellen und Diagramme

5. Tabellen und Diagramme

Im folgenden Kapitel haben wir für Sie die wichtigsten Informationen über Ihren Commodore 64 in tabellarischer Form bzw. mit Abbildungen dargestellt. Aus Gründen der Übersicht und um Ihnen etwas Raum für eigene Notizen zur Verfügung zu stellen, beginnt jeder Abschnitt auf einer neuen Seite.

5.1 Speicher und Register

Da der Anwenderspeicher, seine Aufteilung und Handhabung, sowie die benötigten Register zur Steuerung der Eigenschaften Ihres 64er grundlegende Bedeutung haben, widmet sich der erste Teil ganz diesem Thema.

5.1.1 Zero-Page und weitere wichtige Adressen

Hexadez.	Dezimal	Belegung
00	0	Datenrichtungsregister für Prozessorport
01	1	Prozessorport
02 - 06	2 - 6	frei
07	7	Suchzeichen
08	8	Merker für Hochkomma-Modus
09	9	Speicher für Spalte beim TAB-Befehl
0A	10	Load:0, Verify:1,
0B	11	Zeiger in Eingabepuffer, Anzahl der Dimensionen
0C	12	Merker für DIM
0D	13	Merker $00:numerisch, $FF:String
0E	14	Merker $80:Integer, $00:Fließkomma
0F	15	Merker für Hochkomma-Modus bei LIST
10	16	Merker für FN

Hexadez.	Dezimal	Belegung
11	17	Merker für INPUT:$00, GET:$40, READ:$98
12	18	Vorzeichen bei ATN
13	19	aktives I/O-Gerät
14 - 15	20 - 21	Integer-Adresse, z.B. Zeilennummer
16	22	Zeiger auf Stringstack
17 - 18	23 - 24	Zeiger auf zuletzt verwendeten String
19 - 21	25 - 33	Stringstack
22 - 23	34 - 35	Zeiger auf String nach FRESTR
24 - 25	36 - 37	Zeiger für diverse Zwecke
26 - 2A	38 - 42	Register für Funktionsauswertung und Arithmetik
2B - 2C	43 - 44	Zeiger auf Basic-Programmstart
2D - 2E	45 - 46	Zeiger auf Start der Variablen
2F - 30	47 - 48	Zeiger auf Start der Arrays
31 - 32	49 - 50	Zeiger auf Ende der Arrays
33 - 34	51 - 52	Zeiger auf Beginn der Strings
35 - 36	53 - 54	Hilfszeiger auf String
37 - 38	55 - 56	Zeiger auf BASIC-RAM Ende
39 - 3A	57 - 58	aktuelle BASIC-Zeilennummer
3B - 3C	59 - 60	letzte Basic-Zeilennummer
3D - 3E	61 - 62	Zeiger auf nächsten BASIC-Befehl für CONT
3F - 40	63 - 64	aktuelle Zeilennummer für DATA
41 - 42	65 - 66	Zeiger auf nächstes DATA-Element
43 - 44	67 - 68	Zeiger auf Eingabeelement
45 - 46	69 - 70	aktueller Variablenname
47 - 48	71 - 72	aktuelle Variablenadresse
49 - 48	73 - 74	Zeiger auf Variablenwert
4B - 4C	75 - 76	Zwischenspeicher für Programmzeiger
4D	77	Maske für Vergleichoperationen
4E - 4F	78 - 79	Zeiger für FN
50 - 53	80 - 83	Stringdescriptor
54	84	Konstante $4C JMP für Funktionen
55 - 56	85 - 86	Sprungvektor für Funktionen
57 - 5B	87 - 91	Register für Arithmetik, Akku#3
5C - 60	92 - 96	Register für Arithmetik, Akku#4
61 - 65	97 - 101	Fließkommaakku #1, FAC
66	102	Vorzeichen von FAC
67	103	Zähler für Polynomauswertung
68	104	Rundungsbyte für FAC
69 - 6D	105 - 109	Fließkommaakku #2, ARG
6E	110	Vorzeichen von ARG
6F	111	Vergleichsbyte der Vorzeichen von FAC und ARG
70	112	Rundungsbyte für FAC
71 - 72	113 - 114	Zeiger für Polynomauswertung
73 - 8A	115 - 138	CHRGET-Routine, holt Zeichen aus

Hexadez.	Dezimal	Belegung
		BASIC-Text
79	121	CHRGOT-Routine
7A - 7B	122 - 123	Programmzeiger
8B - 8F	139 - 143	letzter RND-Wert
90	144	Statuswert ST
91	145	Merker für STOP-Taste
92	146	Zeitkonstante für Band
93	147	Merker für LOAD:$00 oder VERIFY:$01
94	148	Merker bei IEC-Ausgabe
95	149	Ausgabepuffer für IEC-Bus
96	150	Merker für EOT vom Band empfangen
97	151	Zwischenspeicher für Register
98	152	Anzahl der offenen Dateien
99	153	aktuelles Eingabegerät
9A	154	aktuelles Ausgabegerät
9B	155	Parität für Band
9C	156	Merker für Byte empfangen
9D	157	Merker für Direkt-Modus:$80, Programm:$00
9E	158	Band Pass 1 Prüfsumme
9F	159	Band Pass 2 Fehlerkorrektur
A0 - A2	160 - 162	Uhr (TI)
A3	163	Bitzähler für serielle Ausgabe
A4	164	Zähler für Band
A5	165	Zähler für Band schreiben
A6	166	Zeiger in Bandpuffer
A7 - A8	167 - 171	Arbeitsspeicher für Bandein/ausgabe
AC - AD	172 - 173	Zeiger für Bandpuffer und Scrolling
AE - AF	174 - 175	Zeiger auf Programmende bei LOAD/SAVE
B0 - B1	176 - 177	Zeitkonstanten für Band-Timing
B2 - B3	178 - 179	Zeiger auf Bandpuffer
B4	180	Bitzähler für Band
B5	181	nächstes Bit für RS 232
B6	182	Puffer für auszugebendes Byte
B7	183	Länge des Dateinamens
B8	184	aktuelle logische Dateinummer
B9	185	aktuelle Sekundäradresse
BA	186	aktuelle Gerätenummer
BB - BC	187 - 188	Zeiger auf Dateinamen
BD	189	serielle Ein/Ausgabe
BE	190	Passzähler für Band
BF	191	Puffer für serielle Ausgabe
C0	192	Merker für Bandmotor
C1 - C2	193 - 194	Startadresse für Ein/Ausgabe vom Bildschirm
C3 - C4	195 - 196	Endadresse für Ein/Ausgabe vom Bildschirm
C5	197	Nummer der gedrückten Taste,

Hexadez.	Dezimal	Belegung
		64:keine Taste
C6	198	Anzahl der gültigen Zeichen im Tastaturpuffer
C7	199	Merker für RVS-Modus
C8	200	Zeilenende für Eingabe
C9	201	Cursorzeile für Eingabe
CA	202	Cursorspalte für Eingabe
CB	203	gedrückte Taste, 64:keine Taste
CC	204	Merker für Cursor, O:Cursor ein, 1:Cursor aus
CD	205	Zähler für Cursor blinken
CE	206	Zeichen unter dem Cursor
CF	207	Merker für Cursor, 1:Ein-Phase, O:Aus-Phase
DO	208	Merker für Eingabe von Tastatur oder Bildschirm
D1 - D2	209 - 210	Zeiger auf Start der aktuellen Bildschirmzeile
D3	211	Cursorspalte.
D4	212	Merker für Hochkommamodus
D5	213	Länge der Bildschirmzeile
D6	214	Cursorzeile
D7	215	für verschiedene Zwecke
D8	216	Anzahl der Inserts
D9 - F2	217 - 242	MSB der Bildschirmzeilenanfänge
F3 - F4	243 - 244	Zeiger in Farb-RAM
F5 - F6	245 - 246	Zeiger auf Tastatur-Dekodiertabelle
F7 - F8	247 - 248	Zeiger auf RS 232 Eingabepuffer
F9 - FA	249 - 250	Zeiger auf RS 232 Ausgabepuffer
FB - FE	251 - 254	frei

Hexadez.	**Dezimal**	**Belegung**
00FF - 010A	255 - 266	Puffer für Umwandlung Fließkomma nach ASCII
0100 - 013E	256 - 318	Speicher für Korrektur bei Bandeingabe
0100 - 01FF	256 - 511	Prozessor Stack
0200 - 0258	512 - 600	BASIC-Eingabepuffer
0259 - 0262	601 - 610	Tabelle der logischen Dateinummern
0263 - 026C	611 - 620	Tabelle der Gerätenummern
026D - 0276	621 - 630	Tabelle der Sekundäradresse
0277 - 0280	631 - 640	Tastaturpuffer
0281 - 0282	641 - 642	Start des BASIC-RAM
0283 - 0284	643 - 644	Ende des BASIC-RAM

Hexadez.	Dezimal	Belegung
0285	645	Timeout-Merker für IEC-Bus
0286	646	aktuelle Farbe
0287	647	Farbe unter dem Cursor
0288	648	High-Byte Video-RAM
0289	649	Länge des Tastaturpuffers
028A	650	Merker für Repeatfunktion aller Tasten
028B	651	Zähler für Repeatgeschwindigkeit
028C	652	Zähler für Repeatverzögerung
028D	653	Merker für Shift, Commodore und CTRL(Bit 0,1 und 2)
028E	654	wie $028D
028F - 0290	655 - 656	Vektor für Tastatur-Dekodierung
0291	657	Merker für SHIFT/Commodore gesperrt
0292	658	Merker für Scrollen
0293	659	RS 232 Kontrollwort
0294	660	RS 232 Befehlswort
0295 - 0296	661 - 662	Bit-Timing
0297	663	RS 232 Status
0298 - 029A	664 - 666	RS 232 Baud-Rate
029B	667	Zeiger auf empfangenes Byte RS 232
029C	668	Zeiger auf Input von RS 232
029D	669	Zeiger auf zu übertragendes Byte RS 232
029E	670	Zeiger auf Ausgabe RS 232
029F - 02A0	671 - 672	Speicher für IRQ während Bandbetrieb
02A1	673	CIA 2 NMI-Merker
02A2	674	CIA 1 Timer A
02A3	675	CIA 1 Interrupt-Merker
02A4	676	CIA 1 Merker für Timer A
02A5	677	aktuelle Bildschirmzeile
02A6	678	Merker für PAL- (1) oder NTSC-Version (0)
02C0 - 02FE	704 - 766	Platz für Sprite (Block 11)
0300 - 0301	768 - 769	$E38B Vektor für BASIC-Warmstart
0302 - 0303	770 - 771	$A483 Vektor für Eingabe einer Zeile
0304 - 0305	772 - 773	$A57C Vektor für Umwandlung in Interpreterkode
0306 - 0307	774 - 775	$A71A Vektor für Umwandlung in Klartext (LIST)
0308 - 0309	776 - 777	$A7E4 Vektor für BASIC-Befehlsadresse holen
030A - 030B	778 - 779	$AEB6 Vektor für Ausdruck auswerten
030C	780	Akku für SYS-Befehl

Hexadez.	Dezimal	Bedeutung
030D	781	X-Register für SYS-Befehl
030E	782	Y-Register für SYS-Befehl
030F	783	Status-Register für SYS-Befehl
0310	784	$4C JMP-Befehl für USR-Funktion
0311 - 0312	785 - 786	USR-Vektor
0314 - 0315	788 - 789	IRQ-Vektor
0316 - 0317	790 - 791	BRK-Vektor
0318 - 0319	792 - 793	NMI-Vektor
031A - 031B	794 - 795	OPEN-Vektor
031C - 031D	796 - 797	CLOSE-Vektor
031E - 031F	798 - 799	CHKIN-Vektor
0320 - 0321	800 - 801	CKOUT-Vektor
0322 - 0323	803 - 803	CLRCH-Vektor
0324 - 0325	804 - 805	INPUT-Vektor
0326 - 0327	806 - 807	OUTPUT-Vektor
0328 - 0329	808 - 809	STOP-Vektor
032A - 032B	810 - 811	GET-Vektor
032C - 032D	812 - 813	CLALL-Vektor
032E - 032F	814 - 815	Warmstart-Vektor
0330 - 0331	816 - 817	LOAD-Vektor
0322 - 0333	818 - 819	SAVE-Vektor
033C - 03FB	828 - 1019	Bandpuffer
0340 - 037E	832 - 894	Platz für Sprite (Block 13)
0380 - 03BE	896 - 958	Platz für Sprite (Block 14)
03C0 - 03FE	960 - 1022	Platz für Sprite (Block 15)

Raum für Notizen:

5.1.2 Speicheraufteilung

Durch POKE 1,x lassen sich beim 64er verschiedene Speicherzustände erreichen, die wir im folgenden an Abbildungen aufzeigen wollen. Da sich die Anzahl der Möglichkeiten potenzieren würde, wenn man die verschiedenen Speicherzustände für den VIC mitberücksichtigt, werden wir diese hier aussparen. In Kapitel 4 wurde z.B. schon auf die Möglichkeit hingewiesen das Bildschirm-RAM und den Zeichengenerator zu verlegen.

Durch Schraffuren wurde jeweils kenntlich gemacht, inwieweit man bei den einzelnen Modi lesend oder schreibend auf den Speicher zugreifen kann.

Abkürzungen:

Abk.	Prozessor-Port	hauptsächliche Verwendung
$\overline{\text{CHAREN}}$	Bit 2	Auslesen des Character-Generators
$\overline{\text{HIRAM}}$	Bit 1	Ausblenden von KERNAL- und Basic-ROM
$\overline{\text{LORAM}}$	Bit 0	Ausblenden des Basic-ROM's

Raum für Notizen:

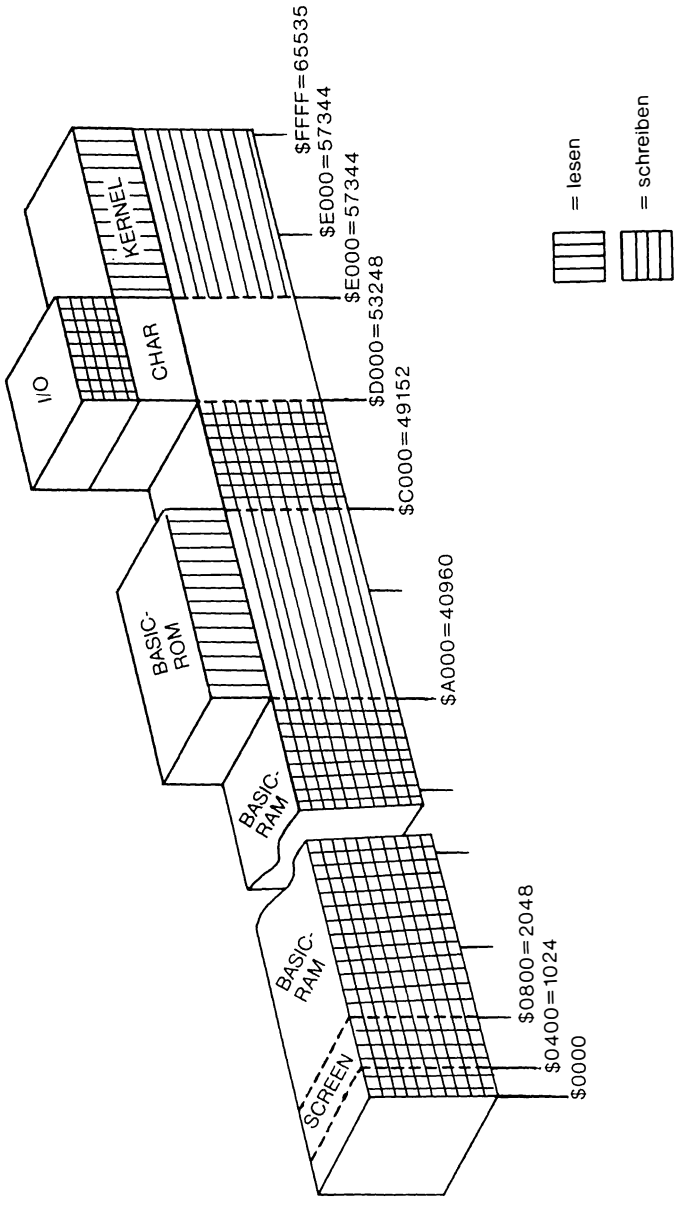


Bild 5.1.2-1 : Einschaltbelegung

$\overline{\text{CHAREN}}$=1; $\overline{\text{HIRAM}}$=1; $\overline{\text{LORAM}}$=1

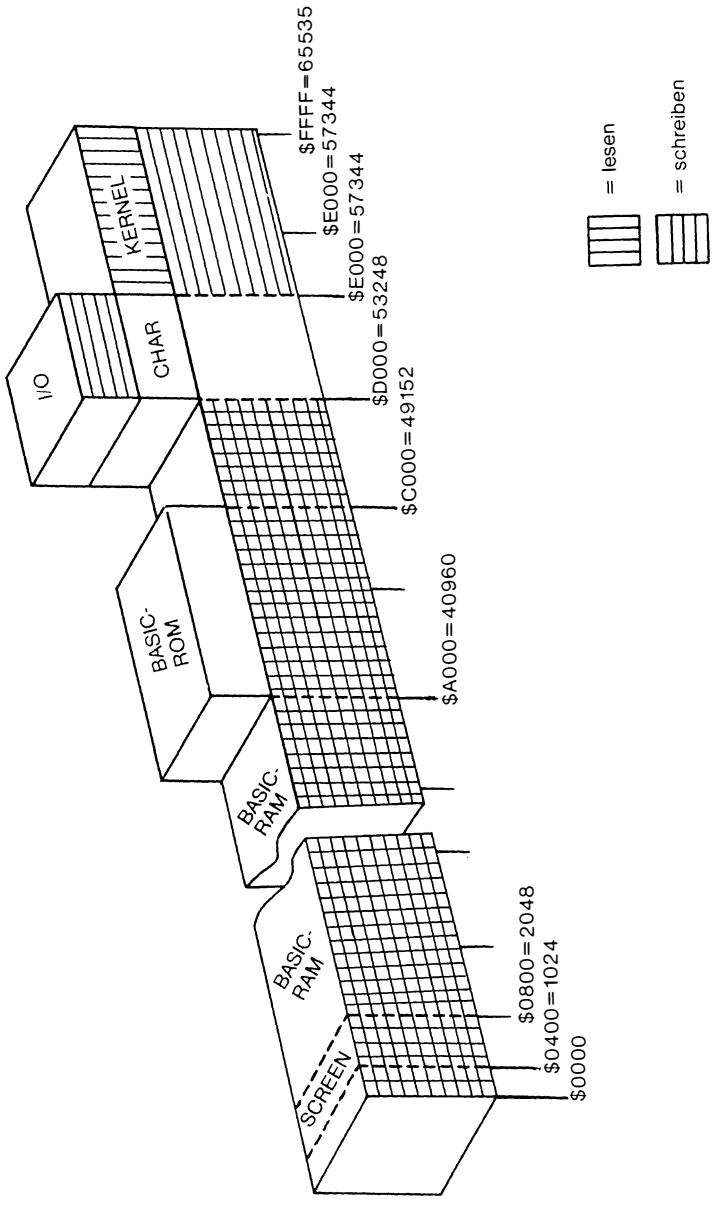


Bild 5.1.2-2 : Basic-ROM ausblenden

$\overline{\text{CHAREN}}$=1; $\overline{\text{HIRAM}}$=1; $\overline{\text{LORAM}}$=0

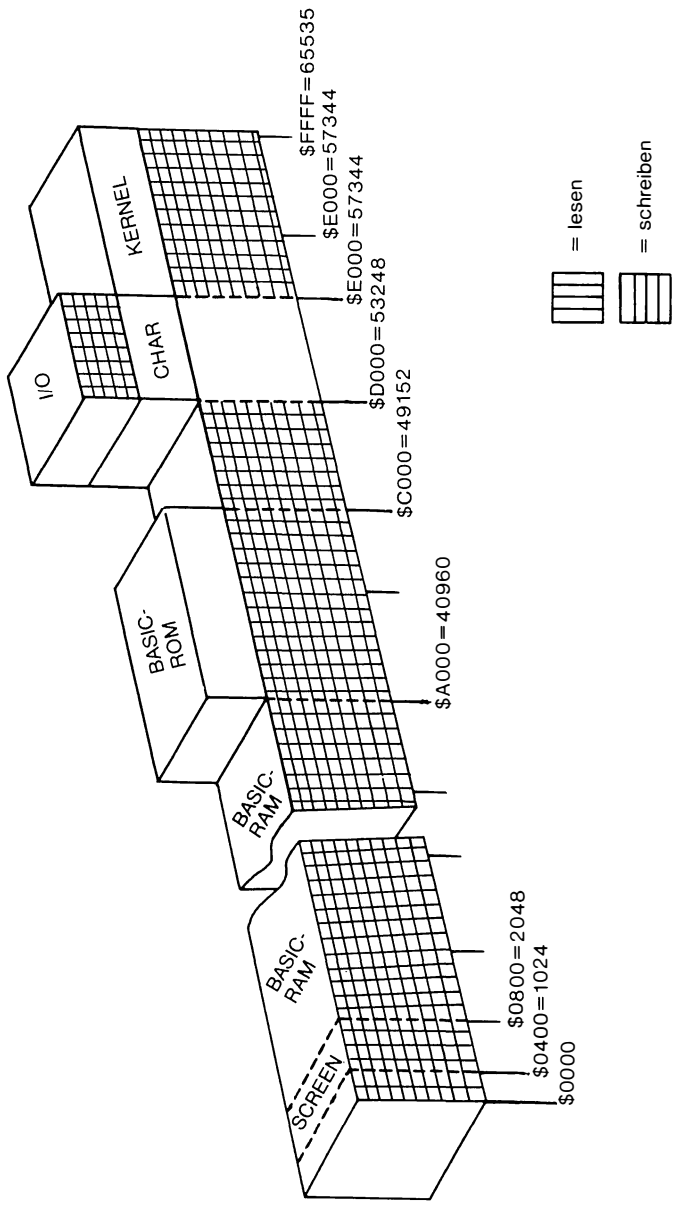


Bild 5.1.2-3 : Basic- und KERNAL-ROM ausblenden

$\overline{\text{CHAREN}}$=1; $\overline{\text{HIRAM}}$=0; $\overline{\text{LORAM}}$=1

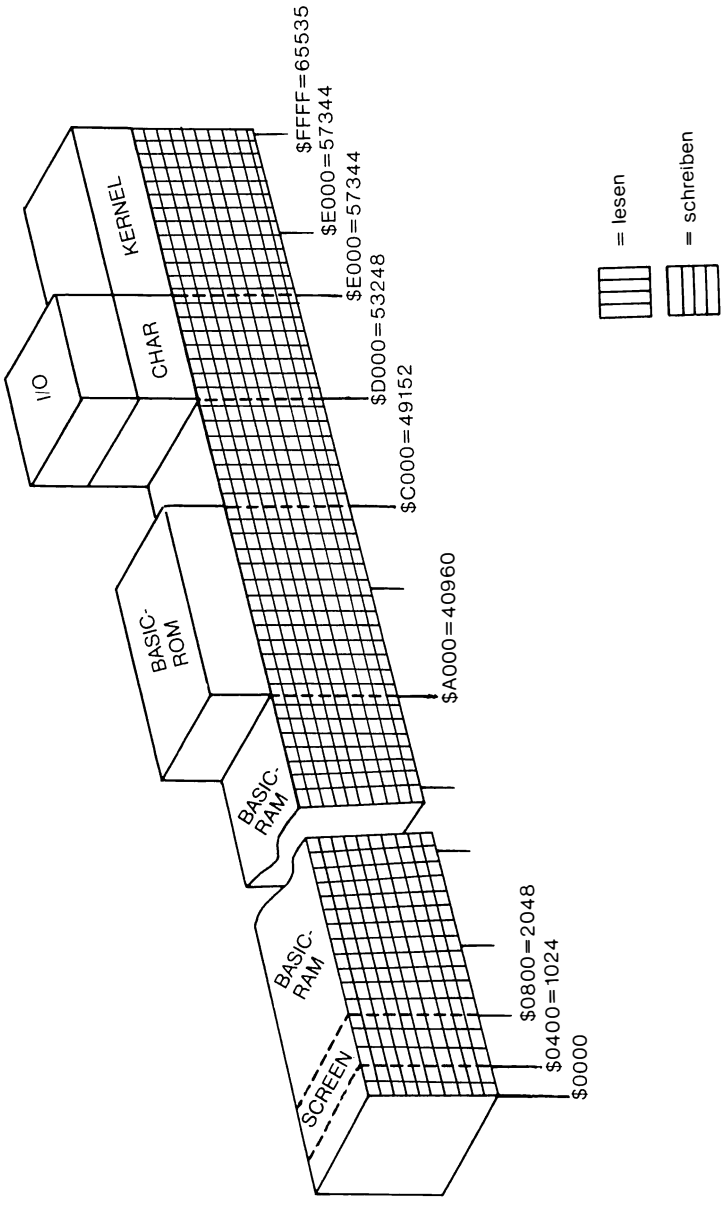


Bild 5.1.2-4 : nur RAM

$\overline{\text{CHAREN}}$=1; $\overline{\text{HIRAM}}$=0; $\overline{\text{LORAM}}$=0

oder $\overline{\text{CHAREN}}$=0; $\overline{\text{HIRAM}}$=0; $\overline{\text{LORAM}}$=0

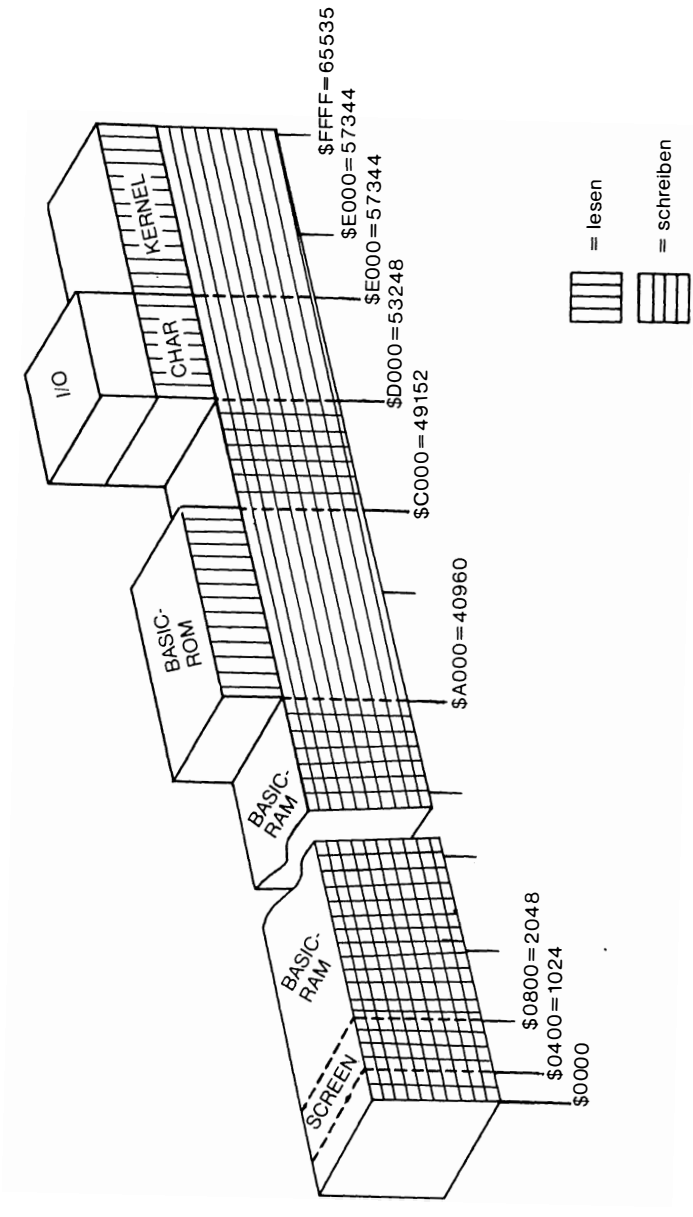


Bild 5.1.2-5 : Zeichengenerator lesen

$\overline{\text{CHAREN}}$=0; $\overline{\text{HIRAM}}$=1; $\overline{\text{LORAM}}$=1

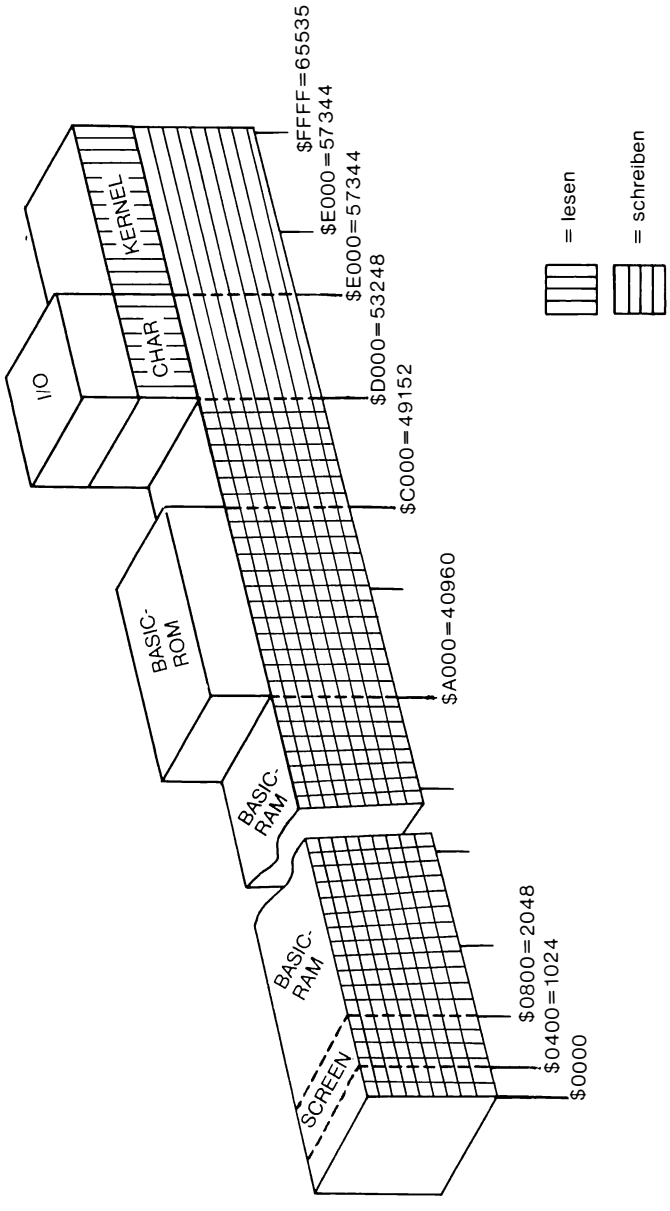


Bild 5.1.2-6 : Zeichengenerator lesen und Basic-ROM ausschalten

$\overline{\text{CHAREN}}$=0; $\overline{\text{HIRAM}}$=1; $\overline{\text{LORAM}}$=0

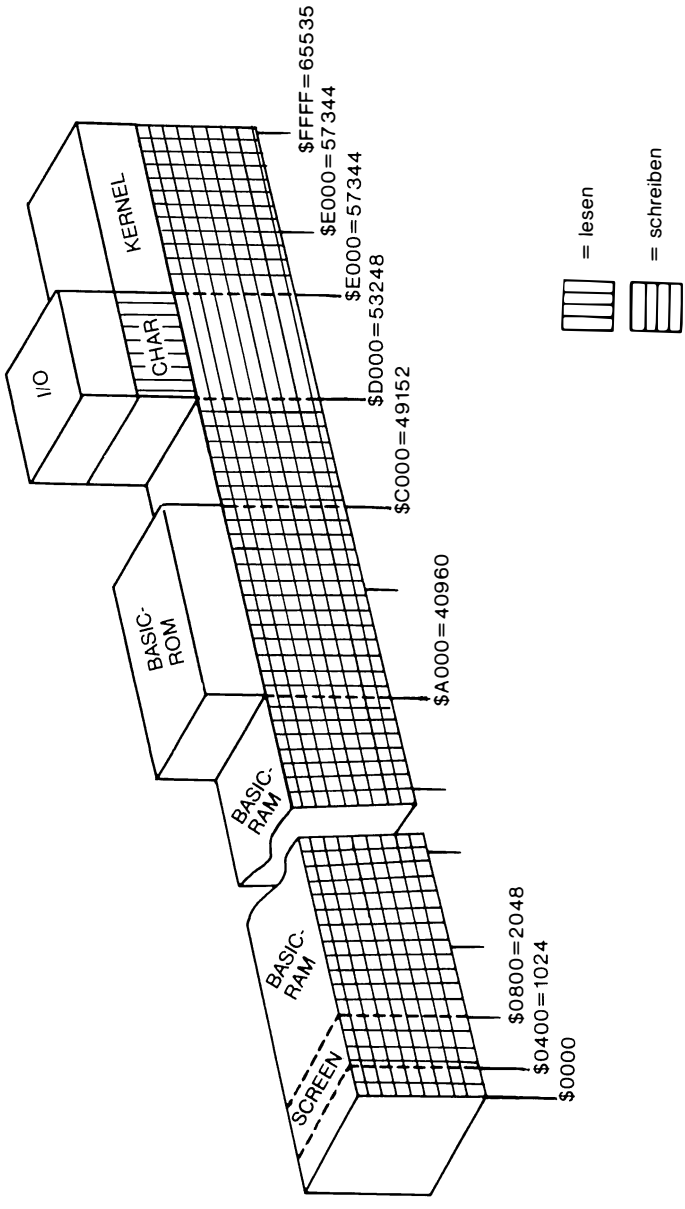


Bild 5.1.2-7 : nur RAM außer Zeichengenerator

$\overline{\text{CHAREN}}$=0; $\overline{\text{HIRAM}}$=0; $\overline{\text{LORAM}}$=1

5.1.3 VIC-Register, Sprites und Grafikspeicher

In der folgenden Übersicht sind alle Register des Video-Prozessors (die auch die Sprite-Steuerung beinhalten) mit einer Kurzbezeichnung dargestellt. Gegebenenfalls sind auch die Funktionen von einzelnen Bits erläutert. Den Abschluß bilden ein paar Darstellungen über die Speicherung von Grafiken.

Mit '*' bezeichnete Register sind für alle Sprites zuständig: je Sprite ein Bit, korrespondierend in den Nummern (Sprite 0 - Bit 0,...,Sprite 7 - Bit 7)

```
53248           Basisadresse  V


53248           Sprite 0: X-Position
53249, V+1      Sprite 0: Y-Position
53250, V+2      Sprite 1: X-Position
53251, V+3      Sprite 1: Y-Position
53252, V+4      Sprite 2: X-Position
53253, V+5      Sprite 2: Y-Position
53254, V+6      Sprite 3: X-Position
53255, V+7      Sprite 3: Y-Position
53256, V+8      Sprite 4: X-Position
53257, V+9      Sprite 4: Y-Position
53258, V+10     Sprite 5: X-Position
53259, V+11     Sprite 5: Y-Position
53260, V+12     Sprite 6: X-Position
53261, V+13     Sprite 6: Y-Position
53262, V+14     Sprite 7: X-Position
53263, V+15     Sprite 7: Y-Position
53264, V+16  *  Sprite 0-7 MSB der X-Position

53265, V+17     Steuerregister 1
                Bits 0-2: Weiches Abrollen (Y-Richtung)
                Bit    3: 24/25 Zeilenwahl (24 Zeilen: 0)
                Bit    4: Bildschirm ausschalten(=0)
                Bit    5: Bit-Map-Modus einschalten (=1)
                Bit    6: Erweiterter Hintergrundmodus(=1)
                Bit    7: Rasterwertregister (MSB)
```

```
53266, V+18      Rasterwertregister (Bits 0-7)
53267, V+19      Lichtgriffel (X-Richtung)
53268, V+20      Lichtgriffel (Y-Richtung)
53269, V+21  *   Sprites ein/aus (1=ein)

53270, V+22      Steuerregister 2
                 Bits 0-2: Weiches Abrollen (X-Richtung)
                 Bit    3: 39/40 Spaltenwahl (40 Spalten:1)
                 Bit    4: Vielfarbenmodus (=1)
                 Bits 5-7: nicht genutzt

53271, V+23      Sprite-Vergrößerungsregister (X-Richtung)

53272, V+24      Adress-Kontrollregister
                 Bit    0: nicht benutzt
                 Bits 1-3: Adresse des Zeichengenerators
                           bzw. Bit-Map
                 Bits 4-7: Adresse des Bildschirmspeichers

53273, V+25      Unterbrechungs-Flaggenregister
                 Bit    0: Rastervergleich
                 Bit    1: Zusammenstoß Sprite - Hinter-
                           grund
                 Bit    2: Zusammenstoß Sprite - Sprite
                 Bit    3: Unterbrechung durch Lichtgriffel
                 Bits 4-6: nicht benutzt
                 Bit    7: eines der Bits 0-3 gesetzt

53274, V+26      Unterbrechungs-Maskenregister
                 Bit    0: Rastervergleich (1=ein)
                 Bit    1: Zusammenstoß Sprite-Hintergrund
                 Bit    2: Zusammenstoß Sprite-Sprite
                 Bit    3: Unterbrechung durch Lichtgriffel
                 Bits 4-7: nicht benutzt

53275, V+27  *   Prioritätsregister Sprite - Hintergrund
                           (1=Sprite)
53276, V+28  *   Sprite-Multi-Color-Steuerregister
                           (1=MCM)
53277, V+29  *   Sprite-Vergrößerungsregister (Y-Richtung)
53278, V+30  *   Kollisionsregister Sprite - Sprite
53279, V+31  *   Kollisionsregister Sprite - Hintergrund

53280, V+32      Rahmenfarbe
                      1-schwarz
                      2-weiss
                      3-rot
                      4-purpur
                      5-grün
                      6-blau
                      7-gelb
```

8-orange
9-braun
10-hellrot
11-dunkelgrau
12-mittelgrau
13-hellgrün
14-hellblau
15-hellgrau

53281, V+33	Hintergrundfarbe 1 (Farben siehe 53280)
53282, V+34	Hintergrundfarbe 2 (Farben siehe 53280)
53283, V+35	Hintergrundfarbe 2 (Farben siehe 53280)
53284, V+36	Hintergrundfarbe 3 (Farben siehe 53280)
53285, V+37	Sprite-Vielfarbenregister 0
53286, V+38	Sprite-Vielfarbenregister 1
53287, V+39	Sprite 0-Farbregister (Farben siehe 53280)
53288, V+40	Sprite 1-Farbregister (Farben siehe 53280)
53289, V+41	Sprite 2-Farbregister (Farben siehe 53280)
53290, V+42	Sprite 3-Farbregister (Farben siehe 53280)
53291, V+43	Sprite 4-Farbregister (Farben siehe 53280)
53292, V+44	Sprite 5-Farbregister (Farben siehe 53280)
53293, V+45	Sprite 6-Farbregister (Farben siehe 53280)
53294, V+46	Sprite 7-Farbregister (Farben siehe 53280)

Raum für Notizen:

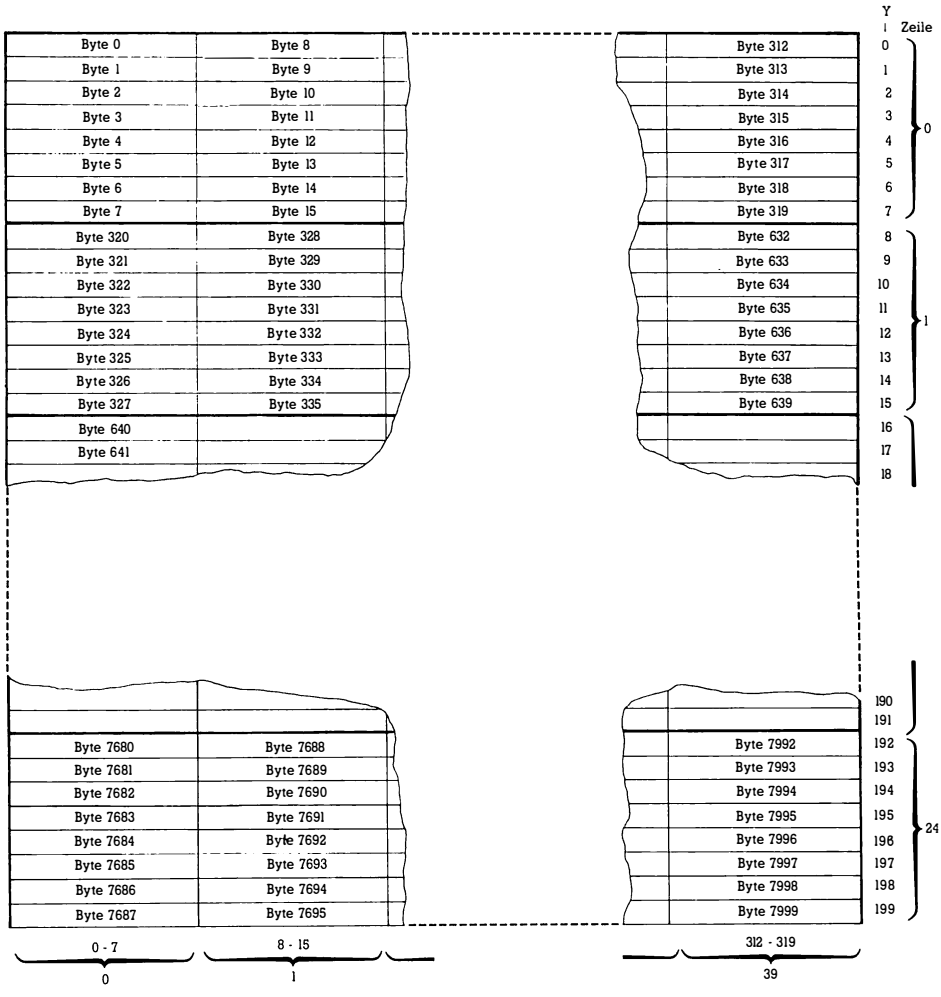


Bild 5.1.3-1 : Numerierung des Grafikspeichers. Zu den angegebenen Ziffern muß jeweils die Basisadresse (bei den Grafikbefehlen dieses Buches $E000) addiert werden.

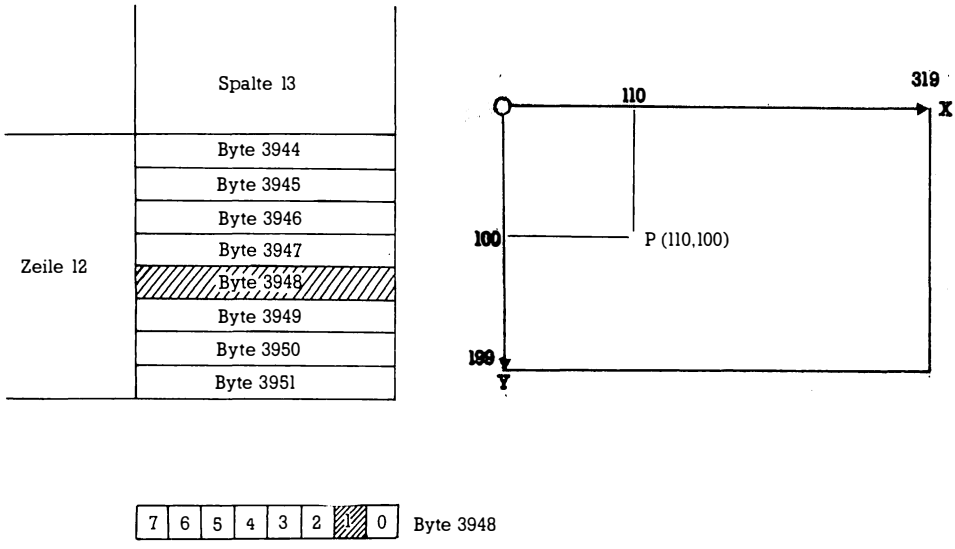


Bild 5.1.3-2 : Ansprechen eines Bildpunktes bei hochauflösender Grafik

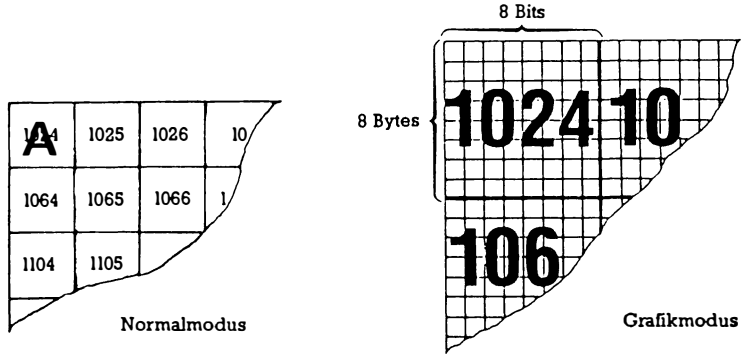


Bild 5.1.3-3 : Normalmodus und Grafikmodus. Im Normalmodus steuert das Bildschirm-RAM (1 Byte - 1 Zeichen) den Zeichengenerator an, der - über den Video-Prozessor - acht Byte (8x8 Punkte) an den Bildschirm weitergibt.

5.1.4 Sound-Register

Analog der Übersicht der VIC-Register stellen wir Ihnen hier die Register des Sound-Chips dar. Ergänzt wird diese Aufstellung von einer Tabelle der Frequenzen und der zugehörigen POKE-Werte.

5.1.4.1 Registerübersicht

54272		**Basisadresse SI**
54272-54296		SID-Chip Steuerregister (können nur beschrieben werden)
54272-54278		Stimme 1
54272	SI	Frequenz (Low Byte)
54273	SI+1	Frequenz (High Byte)
54274	SI+2	Impulsbreite (Low Byte)
54275	SI+3	Impulsbreite (High Byte)
54276	SI+4	Klangkontrollregister Bit 0: Key-Bit (Ton ein/aus) Bit 1: Synchronisation mit Stimme 3 Bit 2: Ringmodulation mit Stimme 3 Bit 3: Testbit (im Normalfall unbenutzt) Bit 4: Dreieckswelle Bit 5: Sägezahnwelle Bit 6: Rechteckwelle Bit 7: Rauschen
54277	SI+5	Steuerregister für Einsatz und Ausklingen Bits 0-3: Wert für Ausklingen (Decay) Bits 4-7: Wert für Toneinsatz (Attack)
54278	SI+6	Steuerregister für Halten und Nachklingen Bits 0-3: Wert für Nachklingen (Release) Bits 4-7: Wert für Halten (Sustain)

54279-54285 Stimme 2

54279	SI+7	Frequenz (Low Byte)
54280	SI+8	Frequenz (High Byte)
54281	SI+9	Impulsbreite (Low Byte)
54282	SI+10	Impulsbreite (High Byte)
54283	SI+11	Klangkontrollregister Bit 0: Key-Bit (Ton ein/aus) Bit 1: Synchronisation mit Stimme 1 Bit 2: Ringmodulation mit Stimme 1 Bit 3: Testbit (im Normalfall unbenutzt) Bit 4: Dreieckswelle Bit 5: Sägezahnwelle Bit 6: Rechteckwelle Bit 7: Rauschen
54284	SI+12	Steuerregister für Einsatz und Ausklingen Bits 0-3: Wert für Ausklingen (Decay) Bits 4-7: Wert für Toneinsatz (Attack)
54285	SI+13	Steuerregister für Halten und Nachklingen Bits 0-3: Wert für Nachklingen (Release) Bits 4-7: Wert für Halten (Sustain)

54286-54292 Stimme 3

54286	SI+14	Frequenz (Low Byte)
54287	SI+15	Frequenz (High Byte)
54288	SI+16	Impulsbreite (Low Byte)
54289	SI+17	Impulsbreite (High Byte)
54290	SI+18	Klangkontrollregister Bit 0: Key-Bit (Ton ein/aus) Bit 1: Synchronisation mit Stimme 2 Bit 2: Ringmodulation mit Stimme 2 Bit 3: Testbit (im Normalfall unbenutzt) Bit 4: Dreieckswelle Bit 5: Sägezahnwelle Bit 6: Rechteckwelle Bit 7: Rauschen
54291	SI+19	Steuerregister für Einsatz und Ausklingen Bits 0-3: Wert für Ausklingen (Decay) Bits 4-7: Wert für Toneinsatz (Attack)
54292	SI+20	Steuerregister für Halten und Nachklingen Bits 0-3: Wert für Nachklingen (Release) Bits 4-7: Wert für Halten (Sustain)

54293-54296 **Klangfilterfunktionen**

54293	SI+21	Cutoff-Frequenz des Filters (Low Byte)
54294	SI+22	Cutoff-Frequenz des Filters (High Byte)
54295	SI+23	Filter/Resonanz-Kontrollregister (1=ein) Bit 0: Filter für Stimme 1 Bit 1: Filter für Stimme 2 Bit 2: Filter für Stimme 3 Bit 3: Filter für externes Signal Bits 4-7: Resonanzwert
54296	SI+24	Modus- und Lautstärke Bits 0-3: Lautstärkenkontrollregister Bit 4: Tiefpaßfilter Bit 5: Bandpaßfilter Bit 6: Hochpaßfilter Bit 7: Stimme 3 ein/aus (1=aus)

Register, die nur gelesen werden können

54297	SI+25	Paddle X-Wert
54298	SI+26	Paddle Y-Wert
54299	SI+27	Momentaner Digitalwert von Stimme 3
54300	SI+28	Hüllkurvenregister (Stimme 3)

5.1.4.2 Frequenztabelle

Vielleicht werden Sie sich wundern, daß die hier abgebildeten Daten mit anderen Tabellen - z.B. der Tabelle im Handbuch - nicht übereinstimmen. Welche Tabelle ist nun die richtige? Auch andere Tabellen sind richtig, aber für Fernseher mit NTSC-Norm. Wieso?

Alle benötigten Frequenzen (auch die zur Steuerung des Fernsehbildes) werden von einem einzigen Quarz abgeleitet. Durch die verschiedenen Fersehnormen befinden sich auch unterschiedliche Oszillatoren in den 64ern, je nach Bestimmungsland.

Die hier vorgestellte Tabelle wurde aufgrund einer ausgemessenen Vergleichsschwingung erstellt. Die meisten Tabellen stellen Übersetzungen der amerikanischen Literatur und damit auch der dort gebräuchlichen Werte dar. Der Faktor zwischen POKE-Wert und Frequenz ist 0.0587.

Frequenz-parameter	POKE-Werte High	Low	Frequenz (Hz)	Ton
277	1	21	16	C2
294	1	38	17	CIS2
312	1	56	18	D2
331	1	75	19	DIS2
351	1	95	21	E2
372	1	116	22	F2
394	1	138	23	FIS2
417	1	161	24	G2
442	1	186	26	GIS2
468	1	212	27	A2
496	1	240	29	B2
526	2	14	31	H2
557	2	45	33	C1
590	2	78	35	CIS1
625	2	113	37	D1
662	2	150	39	DIS1
701	2	189	41	E1
743	2	231	44	F1
787	3	19	46	FIS1
834	3	66	49	G1
884	3	116	52	GIS1
937	3	169	55	A1
993	3	225	58	B1
1052	4	28	62	H1
1115	4	91	65	C
1181	4	157	69	CIS
1251	4	227	73	D
1325	5	45	78	DIS
1404	5	124	82	E
1488	5	208	87	F
1576	6	40	93	FIS
1670	6	134	98	G
1769	6	233	104	GIS
1874	7	82	110	A
1985	7	193	117	B
2103	8	55	123	H
2228	8	180	131	c
2361	9	57	139	cis
2501	9	197	147	d
2650	10	90	156	dis
2808	10	248	165	e
2975	11	159	175	f
3152	12	80	185	fis
3339	13	11	196	g
3538	13	210	208	gis
3748	14	164	220	a
3971	15	131	233	b
4207	16	111	247	h

Frequenz-parameter	POKE-Werte High	Low	Frequenz (Hz)	Ton
4457	17	105	262	c1
4722	18	114	277	cis1
5003	19	139	294	d1
5300	20	180	311	dis1
5615	21	239	330	e1
5949	23	61	349	f1
6303	24	159	370	fis1
6678	26	22	392	g1
7075	27	163	415	gis1
7496	29	72	440	a1
7942	31	6	466	b1
8414	32	222	494	h1
8914	34	210	523	c2
9444	36	228	554	cis2
10006	39	22	587	d2
10601	41	105	622	dis2
11231	43	223	659	e2
11899	46	123	698	f2
12607	49	63	740	fis2
13357	52	45	784	g2
14151	55	71	831	gis2
14992	58	144	880	a2
15884	62	12	932	b2
16829	65	189	988	h2
17830	69	166	1047	c3
18890	73	202	1109	cis3
20013	78	45	1175	d3
21203	82	211	1245	dis3
22464	87	192	1319	e3
23800	92	248	1397	f3
25215	98	127	1480	fis3
26714	104	90	1568	g3
28302	110	142	1661	gis3
29985	117	33	1760	a3
31768	124	24	1865	b3
33657	131	121	1976	h3
35658	139	74	2093	c4
37778	147	146	2218	cis4
40024	156	88	2349	d4
42404	165	164	2489	dis4
44925	175	125	2637	e4
47596	185	236	2794	f4
50426	196	250	2960	fis4
53425	208	177	3136	g4
56602	221	26	3323	gis4
59968	234	64	3520	a4
63534	248	46	3729	b4
67312	262	240	3951	h4

5.1.5 CIA-Register (Complex Interface Adapter)

Weniger bekannt als die Register von VIC und SID sind die Register der CIA's, obwohl sie den anderen an Leistungsfähigkeit nicht nachstehen. Ihre Wirkung läßt sich nicht in Farben und Lautstärke ausdrücken, für die interne Steuerung sind sie aber ein notwendiges Muß.

Zunächst stellen wir eine allgemeine Übersicht der Adressen vor, anschließend folgt eine genaue Beschreibung der Register eines Complex Interfaces Adapters.

5.1.5.1 Adressenübersicht

56320-56335 CIA 1

56320 Datenregister Port A

Tastatur:
Bits 0-7: Spaltenansteuerung der Matrix

Anschlüsse an CP2 (Control-Port 2):
Bits 0-3: Richtung (Joystick)
Bit 4: Feuerknopf (Joystick)
Bits 2/3: Paddle-Feuerknöpfe

Analogschalter:
Bits 6/7: Paddlesatzauswahl CP2/CP1

56321 Datenregister Port B

Tastatur:
Bits 0-7: Zeilenabfrage der Matrix

Anschlüsse an CP1:
Bits 0-3: Richtung (Joystick)
Bit 4: Feuerknopf (Joystick)
Bits 2/3: Paddle-Feuerknöpfe

```
56322           Datenrichtungsregister Port A
56323           Datenrichtungsregister Port B
56324-56327     Timer
56324           Timer A (Low Byte)  (für IRQ-Freq.)
56325           Timer A (High Byte) (für IRQ-Freq.)
56326           Timer A (Low Byte)
56327           Timer B (High Byte)
56238-56331     Realzeituhr
56328           Zehntelsekunden
56329           Sekunden (BCD)
56330           Minuten  (BCD)
56331           Bits 0-6: Stunden  (BCD)
                Bit    7: AM/PM (Anzeige vorm./nachm.)
56332           Serieller E/A-Puffer (synchron)
56333           Unterbrechungs-Kontrollregister
56334           Steuerregister A
56335           Steuerregister B


56576-56831     CIA 2

56576           Datenregister Port A
                Bits 0/1: Segmentwahl des VIC-II-Chip
                          00  Segment 3 (49152 - 65535)
                          01  Segment 2 (32768 - 49151)
                          10  Segment 1 (16384 - 32767)
                          11  Segment 0 (00000 - 16383)
                Bit 2: RS 232 Senderdaten-Ausgabe (TxD)
                Bit 3: ATN-Ausgangssignal
                Bit 4: Serieller Bus, Ausgabe Takt
                Bit 5: Serieller Bus, Ausgabe Daten
                Bit 6: Serieller Bus, Eingabe Takt
                Bit 7: Serieller Bus, Eingabe Daten

56577           Datenregister Port B
                Bit 0: RS 232 Daten empfangen (RxD)
                Bit 1: RS 232 Sendeaufforderung (RTS)
                Bit 2: RS 232 Datenterminal bereit (DTR)
                Bit 3: RS 232 Ringindikator (RI)
                Bit 4: RS 232 Träger entdeckt (DCD)
                Bit 6: RS 232 Bereit zum senden (CTS)
                Bit 7: RS 232 Datensatz bereit (DSR)

56378           Datenrichtungsregister Port A
56379           Datenrichtungsregister Port B
56380-56383     Timer
56380           Timer A (Low Byte)
56381           Timer A (High Byte)
56382           Timer A (Low Byte)
56383           Timer B (High Byte)
```

56384-56387	Realzeituhr
56384	Zehntelsekunden
56385	Sekunden
56386	Minuten
56387	Bits 0 - 6: Stunden Bit 7: AM/PM (Anzeige vorm./nachm.)
56388	Serieller E/A-Puffer (synchron)
56389	Unterbrechungs-Kontrollregister
56390	Steuerregister A
56391	Steuerregister B

5.1.5.2 Registerbeschreibung

Mit dem CIA können vielfältige Ein-/ Ausgabe-Operationen durchgeführt werden. Der Baustein enthält zwei parallele 8-Bit-Schnittstellen, eine serielle Schnittstelle, zwei hintereinander schaltbare 16-Bit-Timer und eine 24-Stunden-Uhr und diverse Interrupt-Möglichkeiten.

Jeder CIA besitzt 16 Register; die Basisadresse der Register des CIA 1 ist $DC00 (=56320), beim CIA 2 ist sie $DD00 (=56576).

Register 0 PRA Port-Register A

Durch dieses Register werden die Leitungen PA0 bis PA7 gesteuert. Beim Auslesen des Registers (z.B mit PEEK) erhalten Sie den Wert, der die Bitkombination der Zustände auf den Leitungen darstellt. Beim Beschreiben des Registers (mit POKE) können Sie die Leitungen, welche aus Ausgang programmiert sind, verändern.

Register 1 PRB Port-Register B

Siehe Port-Register A; statt PA0 bis PA7 werden die Leitungen PB0 bis PB7 angesprochen.

Register 2 DDRA Data-Direction-Register A

Je ein Bit dieses Registers programmiert eine der Leitungen PA0 bis PA7 auf Ein- oder Ausgang. Ist das entsprechende Bit = 1, arbeitet die Leitung als Ausgang, sonst als Eingang.

Register 3 DDRB Data-Direction-Register B

Vergleiche DDRA.

Register 4 TAL Timer A Low- Byte
Register 5 TAH Timer A High-Byte

Der Timer ist ein Register, das automatisch von einem vorgegebenen Startwert bis null herunterzählt. Je nach Betriebsart wird anschließend der Startwert neu geladen, oder/ und der Ablauf des Timers wird signalisiert.

Den momentanen Wert des Timers können Sie durch Lesen von TAL und TAH erfragen. Ein Schreibzugriff auf die beiden Register setzt den neuen Startwert, der erst geladen wird, wenn der momentane Zählvorgang abgeschlossen ist. Wollen Sie während es Zählvorgangs einen neuen Wert in den Timer hineinschreiben, so müssen Sie zusätzlich noch das Bit 4 von Register 14 setzen.

Register 6 TBL Timer B Low- Byte
Register 7 TBH Timer B High-Byte

Vergleiche Register 4 und Register 5

Register 8 TOD10S Time of Day (1/10 Sek.)

Zehntelsekunden der Echtzeituhr

Beim Schreiben oder Lesen des Stundenregisters wird ein Auffangregister (Latch) aktiviert, das durch Lesen oder Schreiben des Zehntelsekunden-Registers wieder freigegeben wird.

Beim Lesen der TOD-Register (8-11) wird

immer die Uhrzeit gelesen, beim Schreiben wird in Abängigkeit von Bit 7 im Register 15 die Uhrzeit oder die Alarmzeit gesetzt.

Register 9 TODSEC Time of Day (Sekunden)

Sekunden der Uhr im BCD-Format, d.h. Bit 0-3 enthalten die Sekunden, Bit 4-7 die Zehner-Sekundenstelle. Z.B. entspricht 45 Sekunden der Wert $45 = 69 (POKE-Wert).

Register 10 TODMIN Time of Day (Minuten)

Minuten der Uhr im BCD-Format (siehe Reg. 9)

Register 11 TODHR Time of Day (Stunden)

Stunden der Uhr. Bit 0-3 enthalten die Einerstunden, Bit 4 die Zehnerstunde und Bit 7 das Vormittags-/Nachmittags-Flag (1 = nachmittags). 12 Uhr mittags aber kann als $80 oder $12 dargestellt werden.

Register 12 SDR Serial Data Register

Dieses Register wirkt auf die Anschlüsse SP und CNT. Es ist ein Schieberegister zum Senden oder Empfangen von seriellen Daten. Die Datenrichtung wird durch Bit 6 von Register 14 bestimmt.

Eingabemodus: Ein Bit wird in das Schieberegister übernommen, wenn am Anschluß CNT eine steigende Flanke auftritt. Nachdem acht Bit übernommen wurden, wird das entsprechende Bit im Register 13 (ICR) gesetzt.

Ausgabemodus: Die Schieberate wird durch den Timer A bestimmt. Die Schiebeimpulse erscheinen am Anschluß CNT. Ist das Schieberegister leer, so wird ebenfalls das Bit im Interrupt-Kontrollregister gesetzt, es sei denn, daß vor Beendigung des Schiebevorgangs ein neuer Wert in das Schieberegister geschrieben wurde.

Register 13 ICR Interrupt-Kontrollregister

Unter dieser Registeradresse werden je nach Zugriffsart zwei verschiedene Register angesprochen. Ein Lesezugriff wirkt auf das Interrupt-Datenregister, ein Schreibvorgang ändert das Interrupt-Maskenregister. Beim Datenregister haben die einzelnen Bits folgende Bedeutungen:

Bit 0 - Timer A ist abgelaufen
Bit 1 - Timer B ist abgelaufen
Bit 2 - Uhrzeit ist gleich Alarmzeit
Bit 3 - Schieberegister voll bzw. leer
Bit 4 - am Anschluß FLAG trat eine fallende Flanke auf
Bit 5 - keine Bedeutung (immer 0)
Bit 6 - keine Bedeutung (immer 0)
Bit 7 - wenigstens eins der Bits 0 bis 4 sind im Datenregister **und** Maskenregister gesetzt.

Wird Bit 7 gesetzt, geht gleichzeitigder Anschluß IRQ des CIA auf low.

Mit dem Maskenregister, das fünf Bit (0-4) beinhaltet, kann man einzelne Interrupts verhindern, indem man das entsprechende Bit (s.o.) löscht. Dieses Register kann jedoch nicht direkt beschrieben werden, sondern man kann mit **einem** POKE-Befehl immer nur Bits setzen bzw. löschen. Für jedes zu verändernde Bit müssen Sie im POKE-Wert das entsprechende Bit setzen. Ist dann noch Bit 7 gesetzt, so werden die Bits gesetzt, ist Bit 7 gleich 0, so werden sie gelöscht. Durch POKEn von 127 werden alle Bits gelöscht.

Register 14 CRA Kontrollregister A

Die einzelnen Bits haben folgende Bedeutung:

Bit 0 - Timer A (0 = Stop, 1 = Start)

Bit 1 - Ein Unterlauf von Timer A wird an dem Anschluß PB6 signalisiert.

Bit 2 - Ist dieses Bit null, so wird an PB6 ein kurzer High-Impuls abgegeben, wenn Timer A abgelaufen ist und Bit 1 von Register 14 gesetzt ist. Ist dieses Bit eins, so wird unter den gleichen Bedingungen der Anschluß PB6 in die jeweils andere Lage gekippt.

Bit 3 - Ist dieses Bit null, so lädt Timer A nach einem Unterlauf den alten Startwert wieder und beginnt erneut zu zählen. Ist das Bit gesetzt, so hält er einfach an.

Bit 4 - Timer A wird sofort geladen

Bit 5 - Ist dieses Bit gleich null, so zählt Timer A Systemtaktpulse (Frequenz ca. 1 MHz), ist es eins, so zählt Timer A steigende Flanken am Anschluß CNT.

Bit 6 - Datenrichtung für Schieberegister (1 bedeutet: SP ist Ausgang, 0 bedeutet: SP ist Eingang)

Bit 7 - Dieses Bit muß gesetzt werden, wenn die Echtzeit-Uhr mit 50 Hertz gespeist wird.

Register 15 CRB Kontrollregister B

Die Bits 0-4 haben eine zu Register 14 völlig analoge Bedeutung, jedoch beziehen sich die Angaben jetzt auf Timer B und Anschluß PB7. Bit 5 und 6 bestimmen die Quelle der Zählimpulse für Timer B entsprechend folgender Tabelle:

Bit 6	Bit 5	Funktion
0	0	Timer B zählt Systemtakte
0	1	Timer B zählt steigende Flanken am Anschluß CNT
1	0	Timer B zählt Unterläufe von Timer A
1	1	Timer B zählt Unterläufe von Timer A, jedoch nur, wenn der Anschluß CNT high ist.

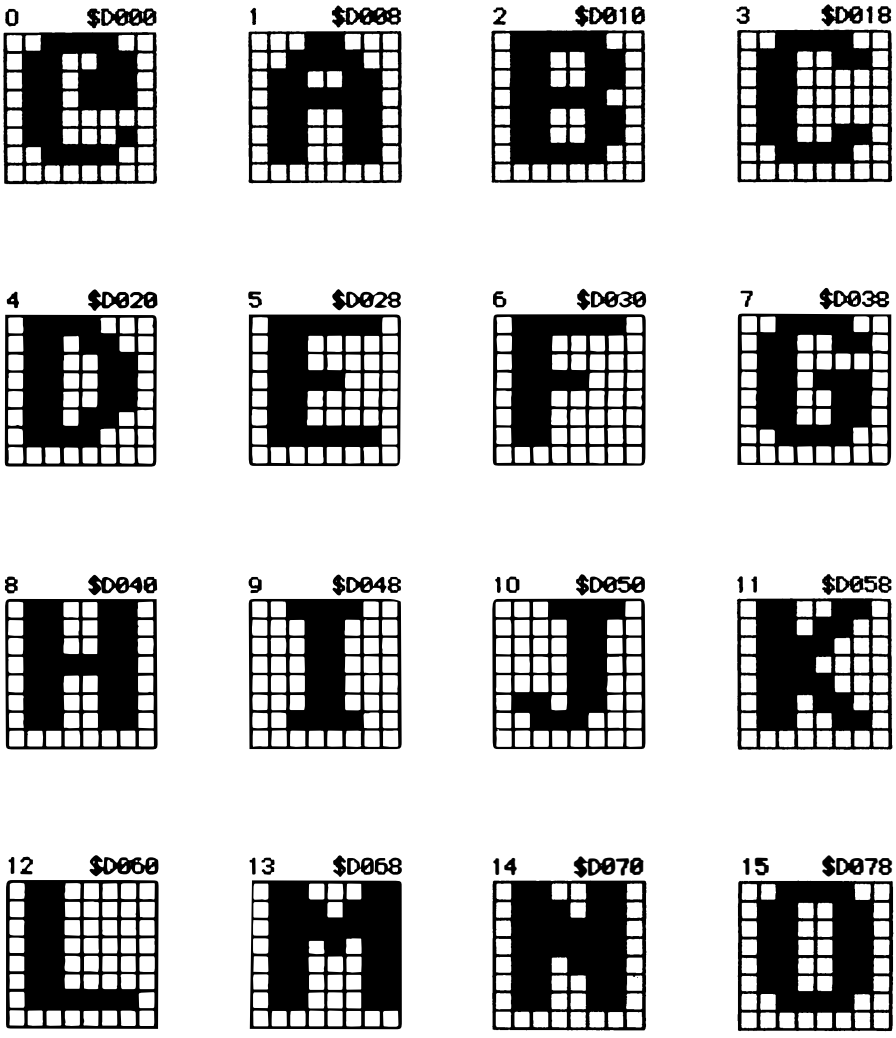
Bit 7 dieses Registers dient zur Wahl zwischen Alarm- und Uhrzeit beim Beschreiben der Register 8 bis 11. Ist dieses Bit gesetzt, so wird die Alarmzeit beschrieben, sonst die Uhrzeit.

Raum für Notizen:

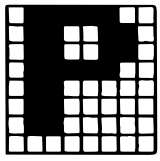
5.1.6 Zeichensatz

Um Ihnen das Erstellen eines eigenen Zeichensatzes zu ermöglichen, aber auch um das Auffinden diverser Grafikzeichen zu vereinfachen, ist im folgenden der gesamte Zeichensatz Ihres Commodore 64 abgebildet.

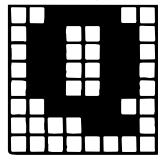
Die angegebenen Zahlen geben den Bildschirmcode (erste Zahl) und das erste Byte im Charakter-Generator an.



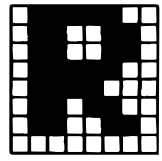
16 $D080



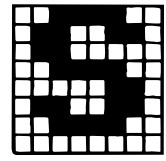
17 $D088



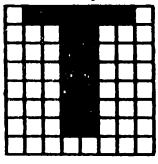
18 $D090



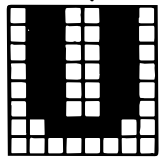
19 $D098



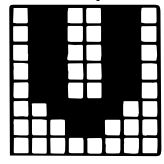
20 $D0A0



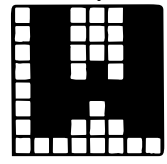
21 $D0A8



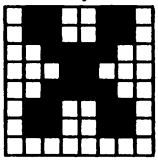
22 $D0B0



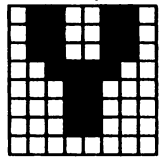
23 $D0B8



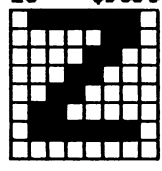
24 $D0C0



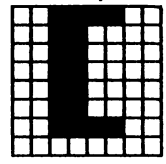
25 $D0C8



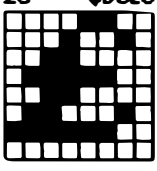
26 $D0D0



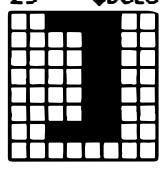
27 $D0D8



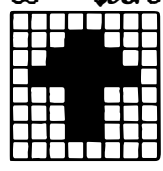
28 $D0E0



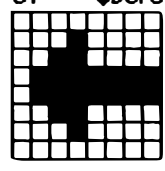
29 $D0E8



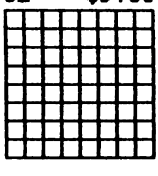
30 $D0F0



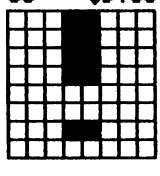
31 $D0F8



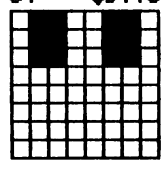
32 $D100



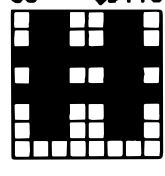
33 $D108



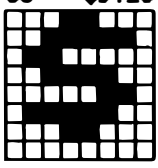
34 $D110



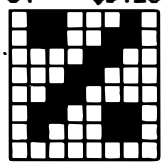
35 $D118



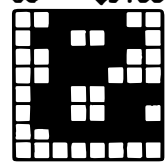
36 $D120



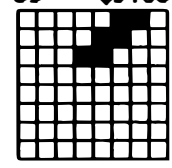
37 $D128



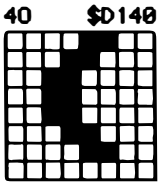
38 $D130



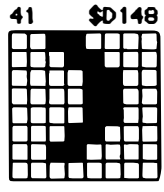
39 $D138



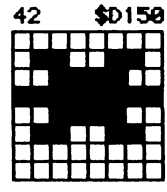
40 $D140



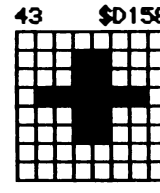
41 $D148



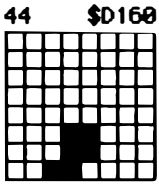
42 $D150



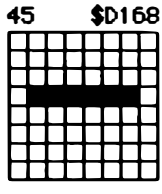
43 $D158



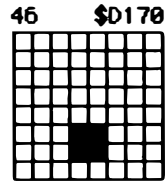
44 $D160



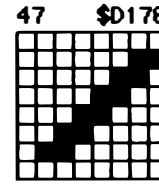
45 $D168



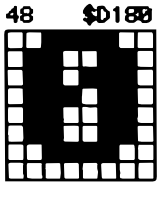
46 $D170



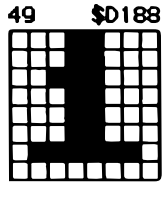
47 $D178



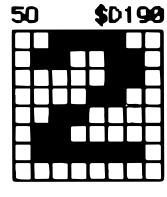
48 $D180



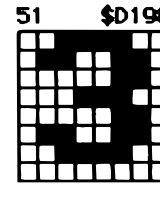
49 $D188



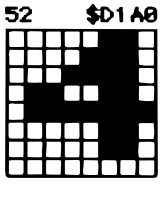
50 $D190



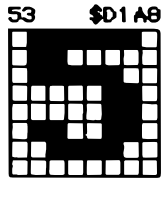
51 $D198



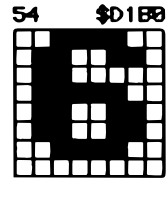
52 $D1A0



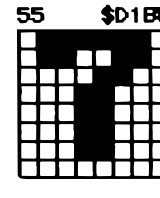
53 $D1A8

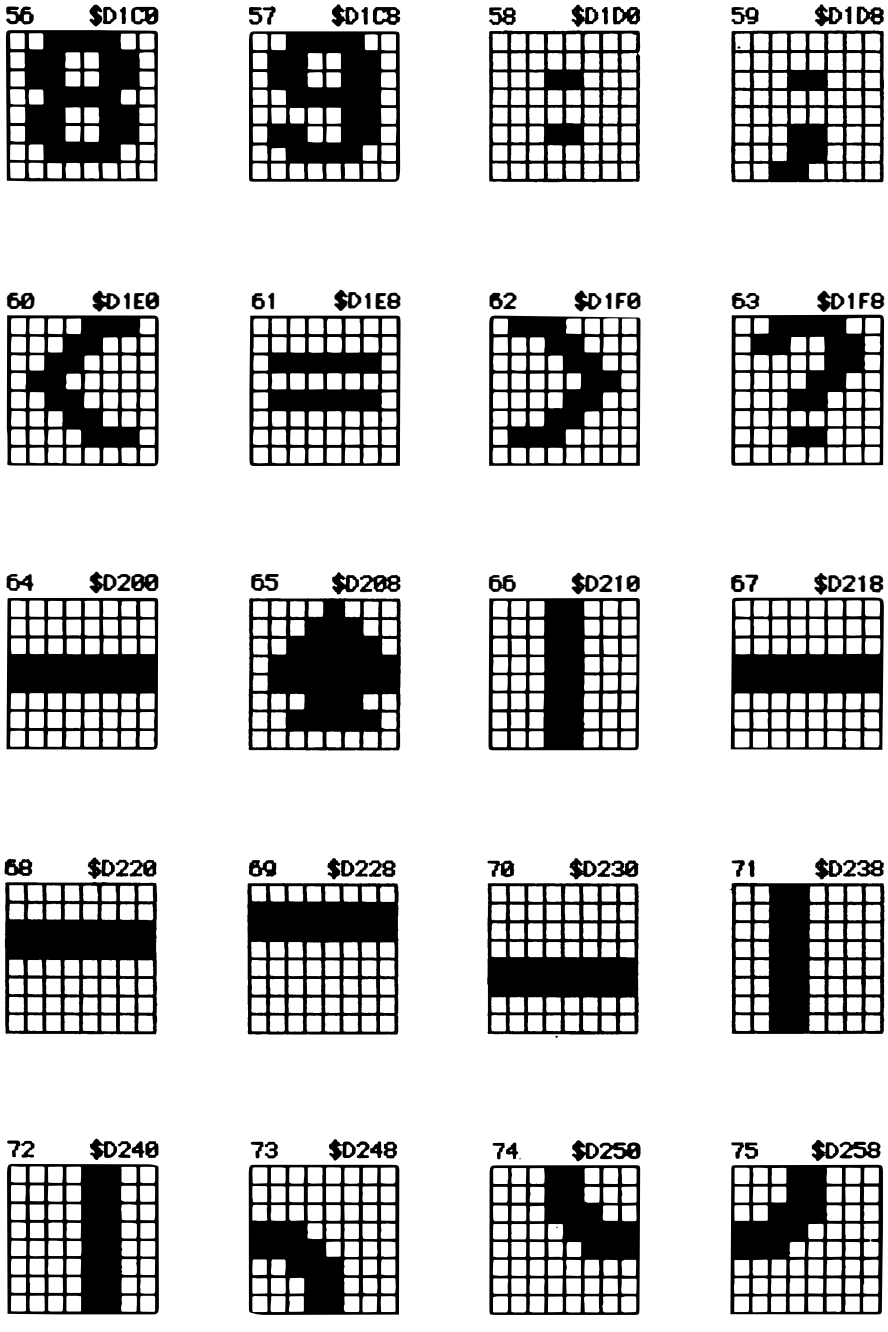


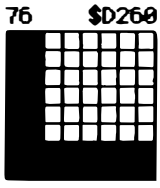
54 $D1B0

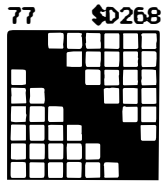


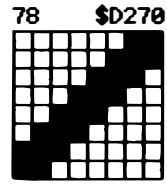
55 $D1B8

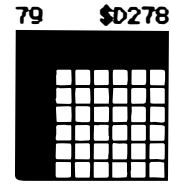


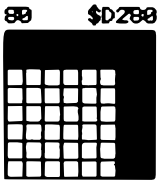

56 $D1C0
57 $D1C8
58 $D1D0
59 $D1D8
60 $D1E0
61 $D1E8
62 $D1F0
63 $D1F8
64 $D200
65 $D208
66 $D210
67 $D218
68 $D220
69 $D228
70 $D230
71 $D238
72 $D240
73 $D248
74 $D250
75 $D258

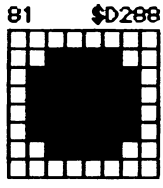

76 $D260

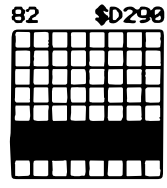

77 $D268

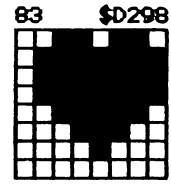

78 $D270

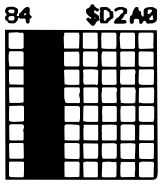

79 $D278

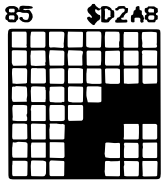

80 $D280

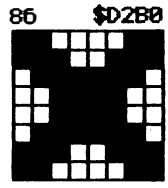

81 $D288

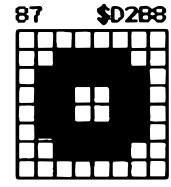

82 $D290

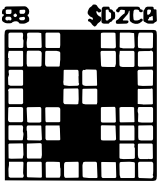

83 $D298

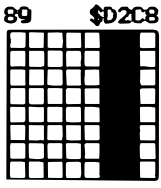

84 $D2A0

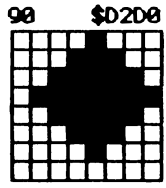

85 $D2A8

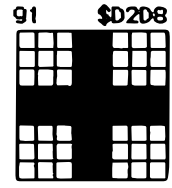

86 $D2B0

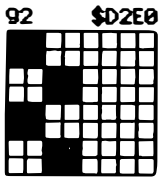

87 $D2B8

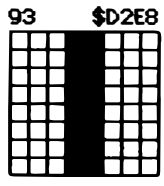

88 $D2C0

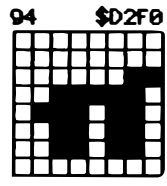

89 $D2C8

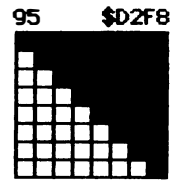

90 $D2D0

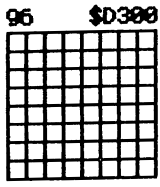

91 $D2D8

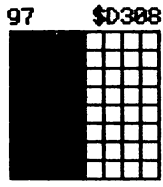

92 $D2E0

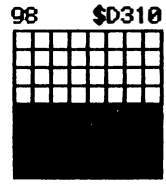

93 $D2E8

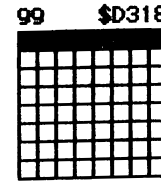

94 $D2F0

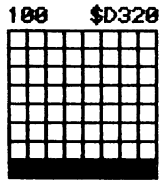

95 $D2F8

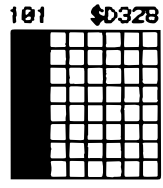

96 $D300

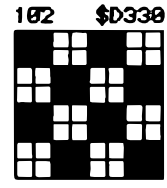

97 $D308

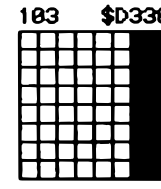

98 $D310

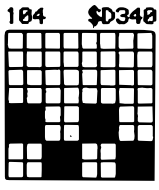

99 $D318

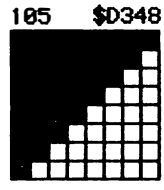

100 $D320

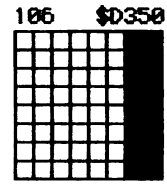

101 $D328

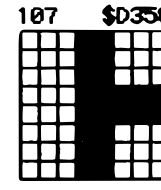

102 $D330

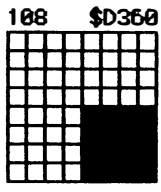

103 $D338

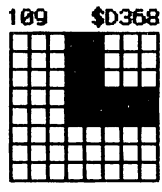

104 $D340

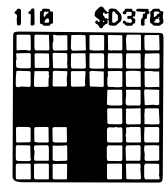

105 $D348

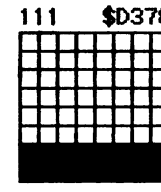

106 $D350

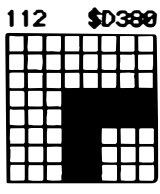

107 $D358

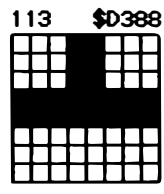

108 $D360

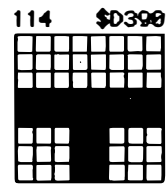

109 $D368

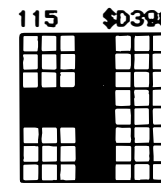

110 $D370

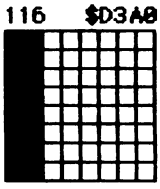

111 $D378

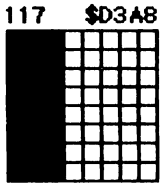

112 $D380

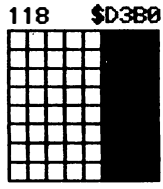

113 $D388

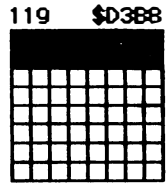

114 $D390

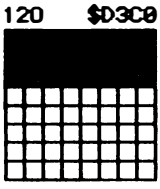

115 $D398

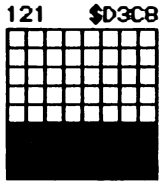

116 $D3A0

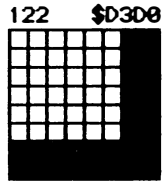

117 $D3A8

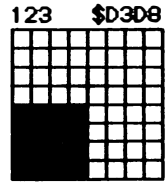

118 $D3B0

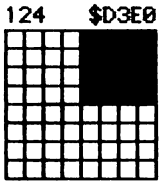

119 $D3B8

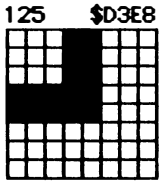

120 $D3C0

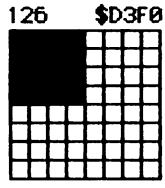

121 $D3C8

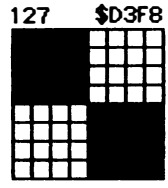

122 $D3D0

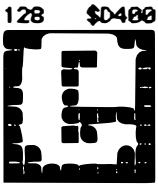

123 $D3D8

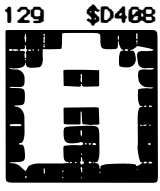

124 $D3E0

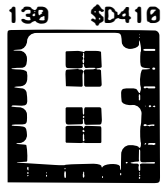

125 $D3E8

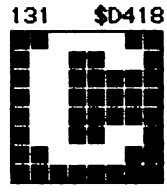

126 $D3F0


127 $D3F8

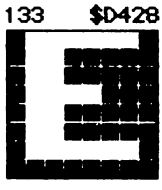

128 $D400

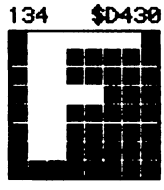

129 $D408


130 $D410


131 $D418


132 $D420

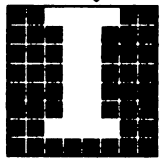

133 $D428


134 $D430

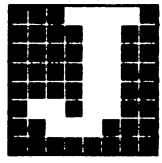

135 $D438

136 $D440

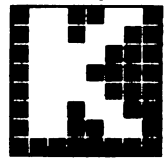
137 $D448



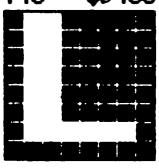
138 $D450



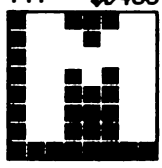
139 $D458



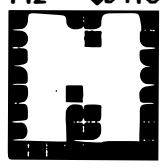
140 $D460



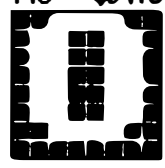
141 $D468



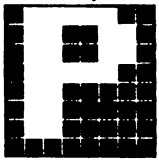
142 $D470



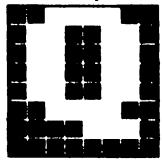
143 $D478



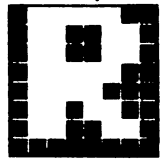
144 $D480



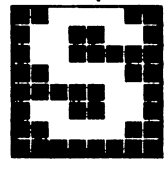
145 $D488



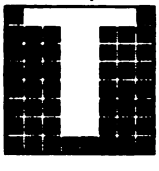
146 $D490



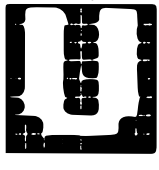
147 $D498



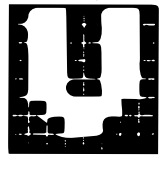
148 $D4A0



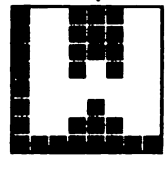
149 $D4A8



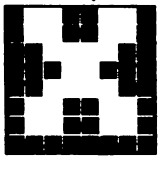
150 $D4B0



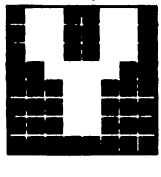
151 $D4B8



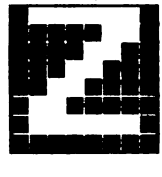
152 $D4C0



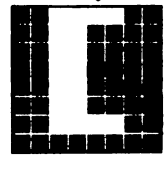
153 $D4C8



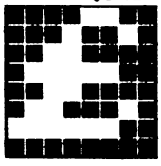
154 $D4D0



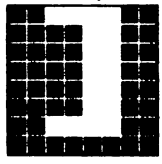
155 $D4D8



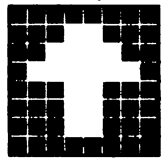
156 $D4E0



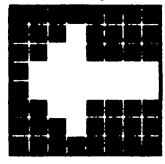
157 $D4E8



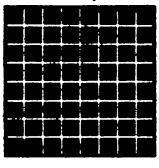
158 $D4F0



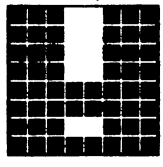
159 $D4F8



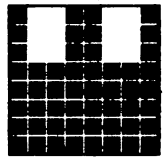
160 $D500



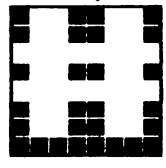
161 $D508



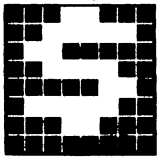
162 $D510



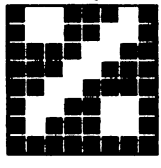
163 $D518



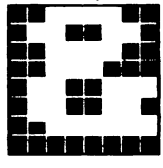
164 $D520



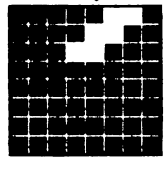
165 $D528



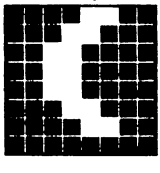
166 $D530



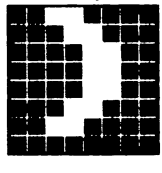
167 $D538



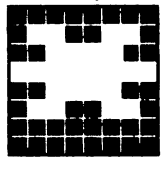
168 $D540



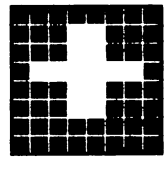
169 $D548



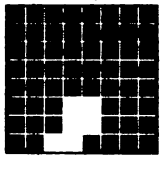
170 $D550



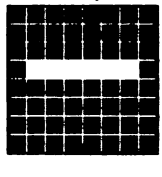
171 $D558



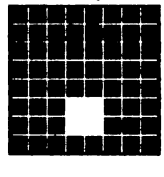
172 $D560



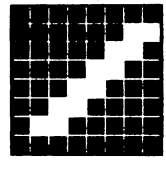
173 $D568



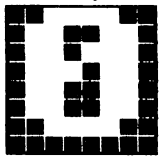
174 $D570



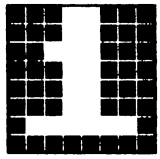
175 $D578



176 $D580



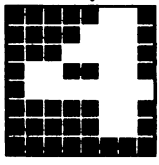
177 $D588



178 $D590

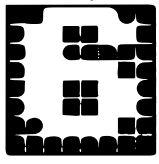
179 $D598

180 $D5A0

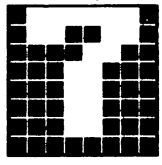


181 $D5A8

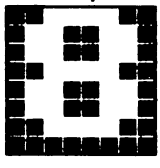
182 $D5B0



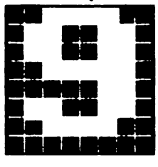
183 $D5B8



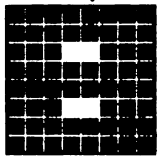
184 $D5C0



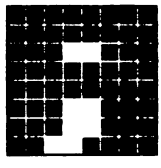
185 $D5C8



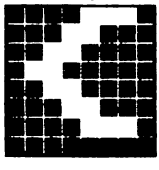
186 $D5D0



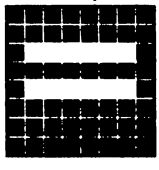
187 $D5D8



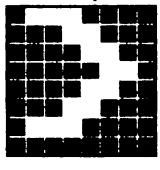
188 $D5E0



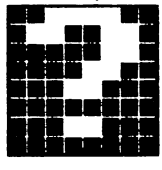
189 $D5E8



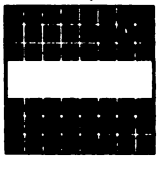
190 $D5F0



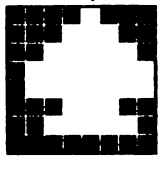
191 $D5F8



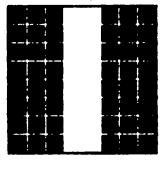
192 $D600



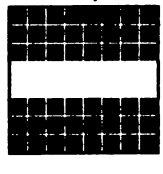
193 $D608

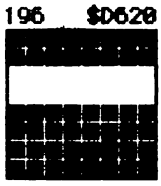


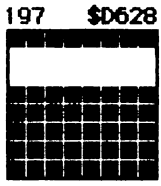
194 $D610

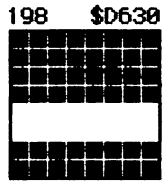


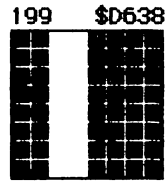
195 $D618

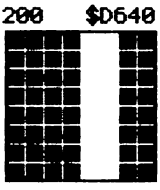


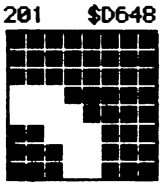

196 $D620

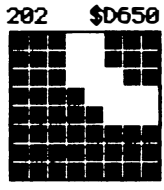

197 $D628

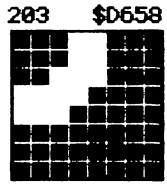

198 $D630

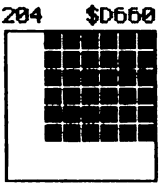

199 $D638

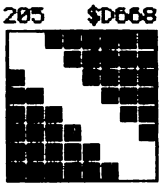

200 $D640

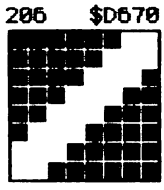

201 $D648

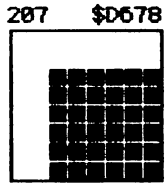

202 $D650

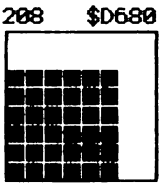

203 $D658

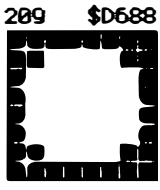

204 $D660

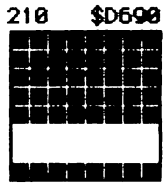

205 $D668

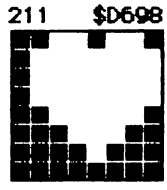

206 $D670

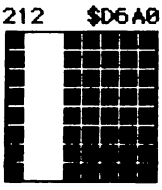

207 $D678

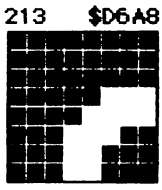

208 $D680

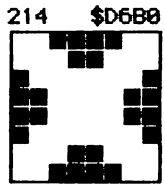

209 $D688

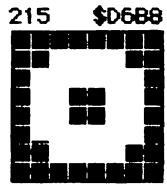

210 $D690

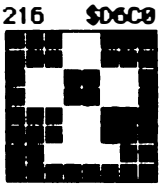

211 $D698

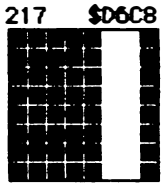

212 $D6A0

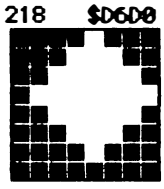

213 $D6A8

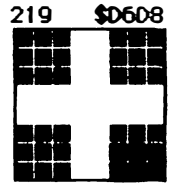

214 $D6B0

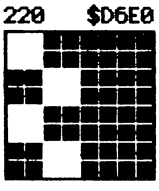

215 $D6B8

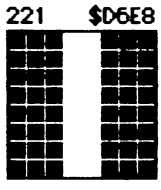

216 $D6C0

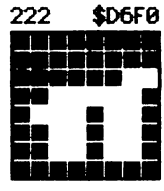

217 $D6C8

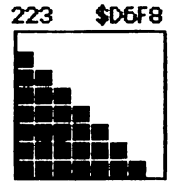

218 $D6D0

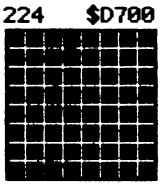

219 $D6D8

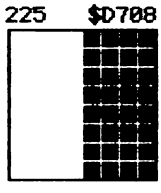

220 $D6E0

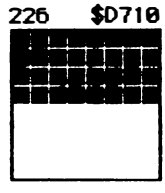

221 $D6E8

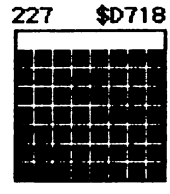

222 $D6F0

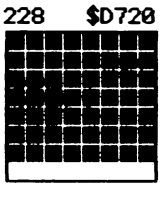

223 $D6F8

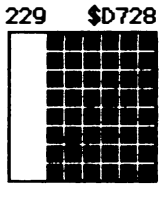

224 $D700

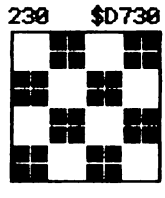

225 $D708

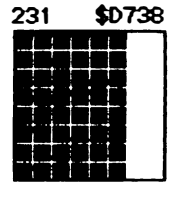

226 $D710

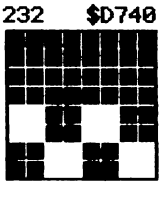

227 $D718

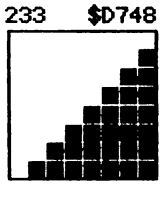

228 $D720

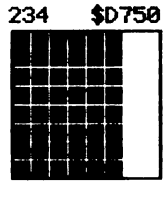

229 $D728

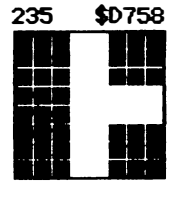

230 $D730

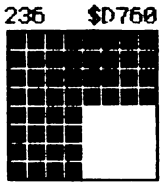

231 $D738

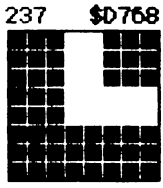

232 $D740

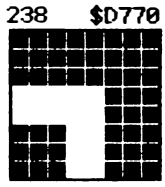

233 $D748

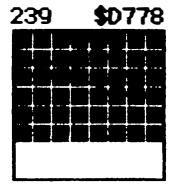

234 $D750

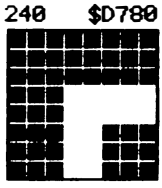

235 $D758

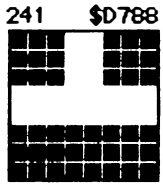

236 $D760

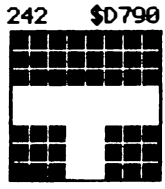

237 $D768

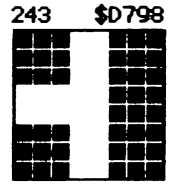

238 $D770

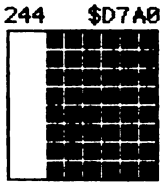

239 $D778

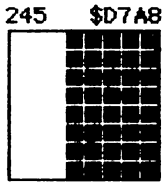

240 $D780

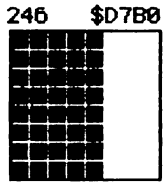

241 $D788

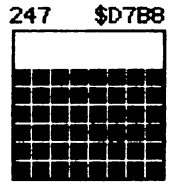

242 $D790

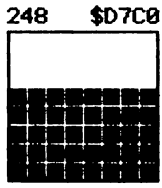

243 $D798

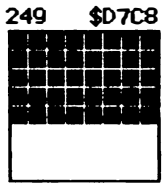

244 $D7A0

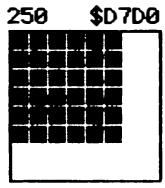

245 $D7A8

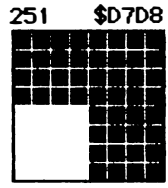

246 $D7B0

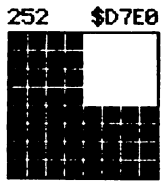

247 $D7B8

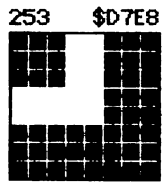

248 $D7C0

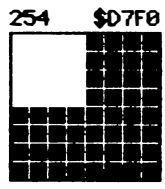

249 $D7C8

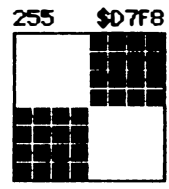

250 $D7D0

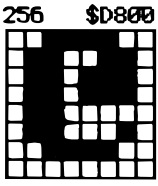

251 $D7D8

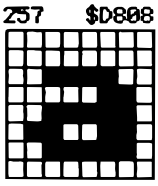

252 $D7E0

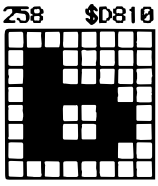

253 $D7E8

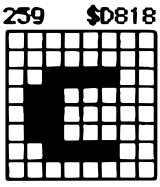

254 $D7F0

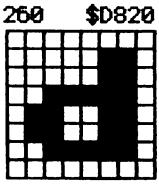

255 $D7F8

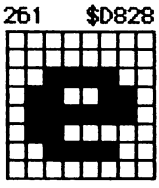

256 $D800

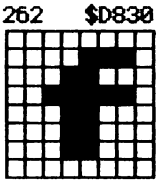

257 $D808

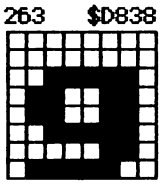

258 $D810

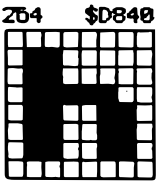

259 $D818

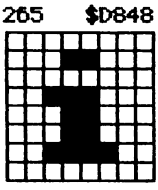

260 $D820

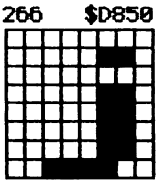

261 $D828

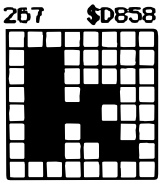

262 $D830

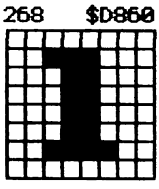

263 $D838

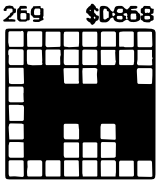

264 $D840

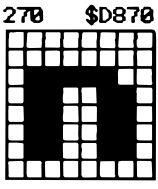

265 $D848

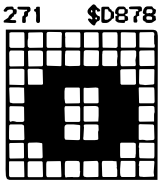

266 $D850

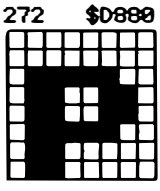

267 $D858

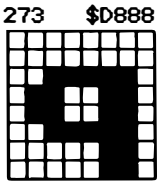

268 $D860

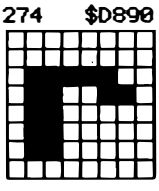

269 $D868

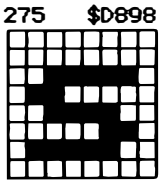

270 $D870

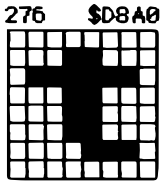

271 $D878

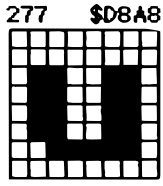

272 $D880

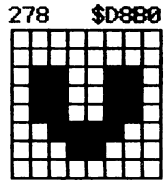

273 $D888

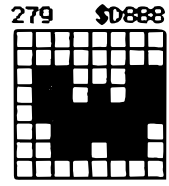

274 $D890

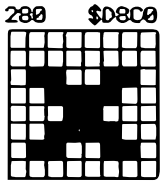

275 $D898

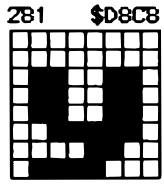

276 $D8A0

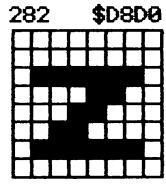

277 $D8A8

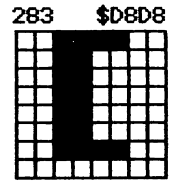

278 $D8B0

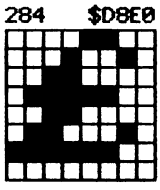

279 $D8B8

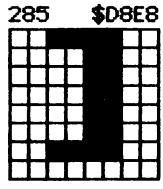

280 $D8C0

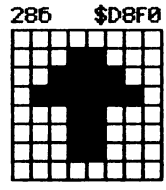

281 $D8C8

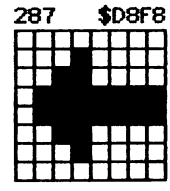

282 $D8D0

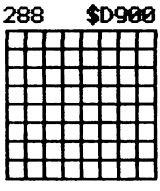

283 $D8D8

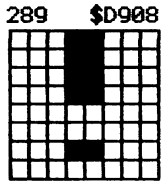

284 $D8E0

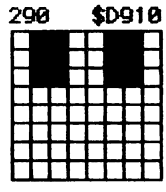

285 $D8E8

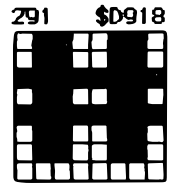

286 $D8F0

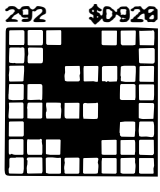

287 $D8F8

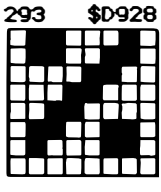

288 $D900

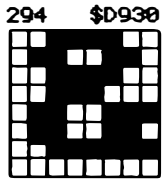

289 $D908

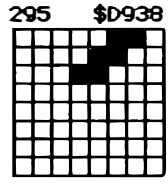

290 $D910

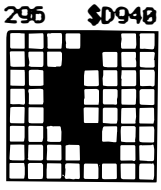

291 $D918

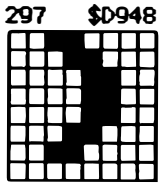

292 $D920

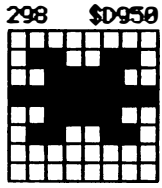

293 $D928

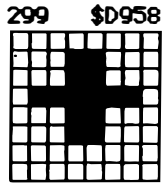

294 $D930

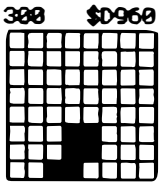

295 $D938

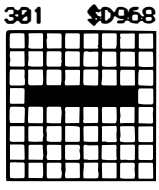

296 $D940

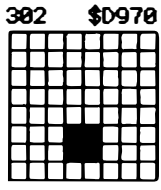

297 $D948

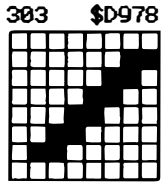

298 $D950

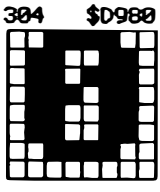

299 $D958

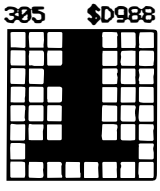

300 $D960

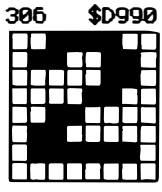

301 $D968

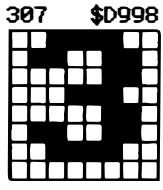

302 $D970

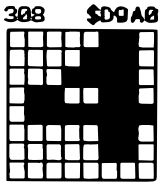

303 $D978

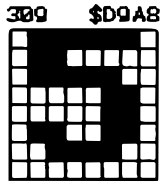

304 $D980

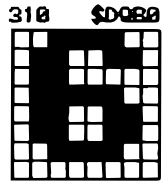

305 $D988

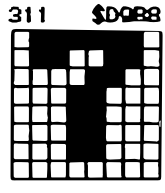

306 $D990

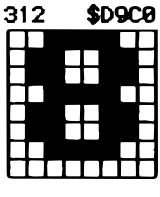

307 $D998

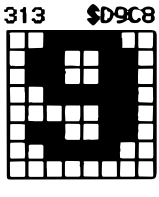

308 $D9A0

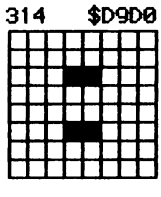

309 $D9A8

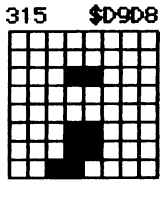

310 $D9B0


311 $D9B8

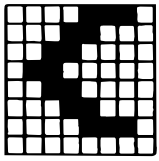

312 $D9C0


313 $D9C8

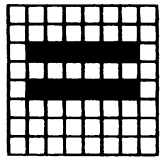

314 $D9D0


315 $D9D8

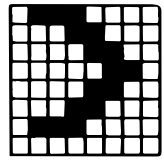
316 $D9E0



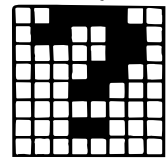
317 $D9E8



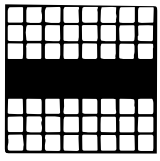
318 $D9F0



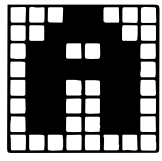
319 $D9F8



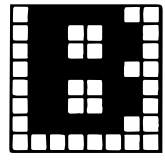
320 $DA00



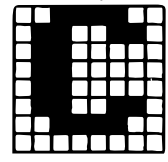
321 $DA08



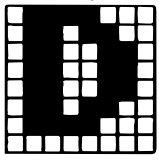
322 $DA10



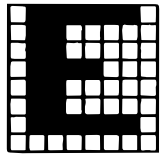
323 $DA18



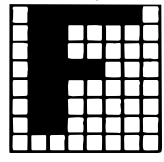
324 $DA20



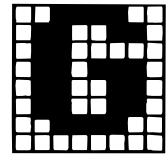
325 $DA28



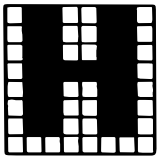
326 $DA30



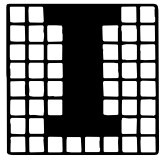
327 $DA38



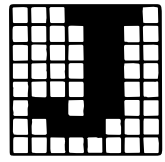
328 $DA40



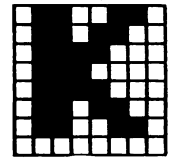
329 $DA48



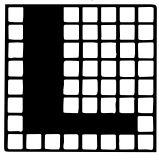
330 $DA50



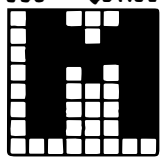
331 $DA58



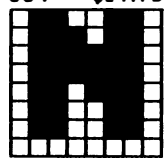
332 $DA60



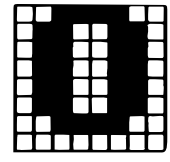
333 $DA68

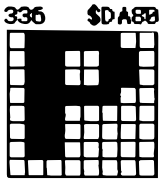


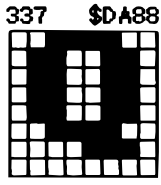
334 $DA70

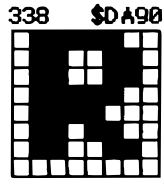


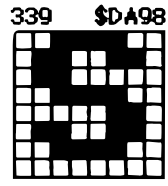
335 $DA78

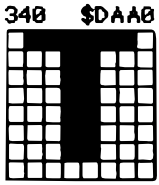


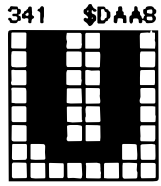

336 $DA80

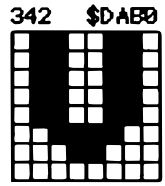

337 $DA88

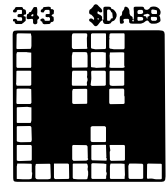

338 $DA90

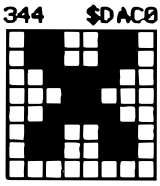

339 $DA98

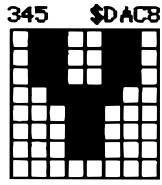

340 $DAA0

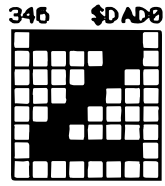

341 $DAA8

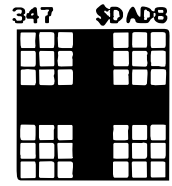

342 $DAB0

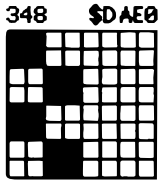

343 $DAB8

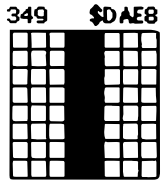

344 $DAC0

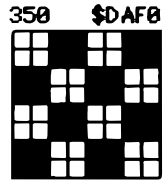

345 $DAC8

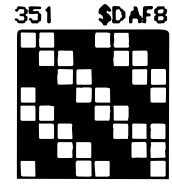

346 $DAD0

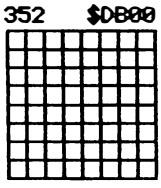

347 $DAD8

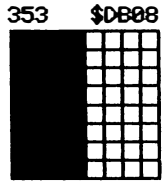

348 $DAE0

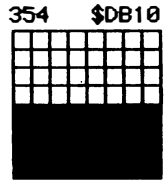

349 $DAE8

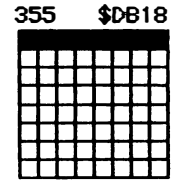

350 $DAF0

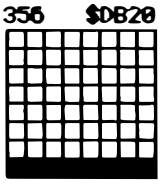

351 $DAF8

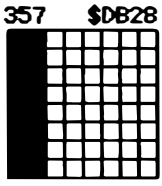

352 $DB00

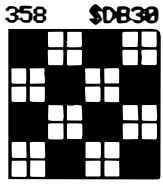

353 $DB08

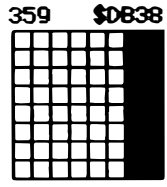

354 $DB10

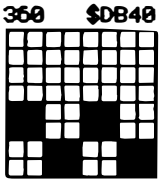

355 $DB18

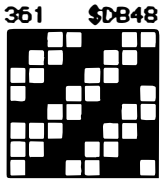

356 $DB20

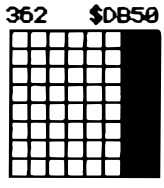

357 $DB28

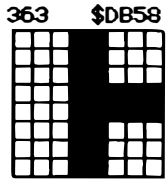

358 $DB30

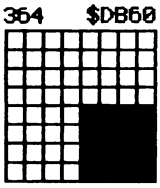

359 $DB38

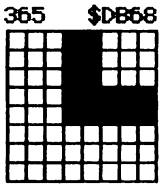

360 $DB40

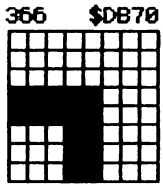

361 $DB48

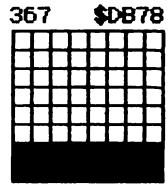

362 $DB50

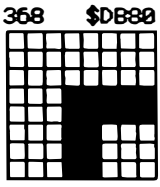

363 $DB58

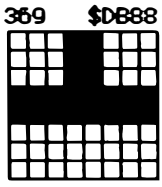

364 $DB60

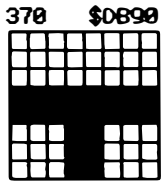

365 $DB68

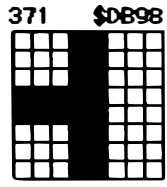

366 $DB70

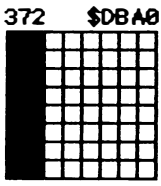

367 $DB78

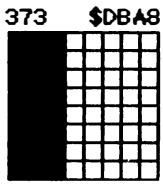

368 $DB80

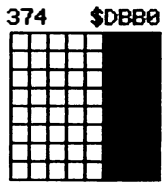

369 $DB88

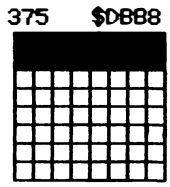

370 $DB90


371 $DB98

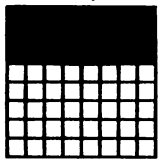

372 $DBA0


373 $DBA8

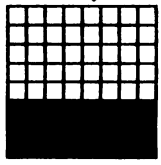

374 $DBB0


375 $DBB8

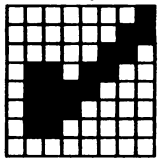
376 $DBC0



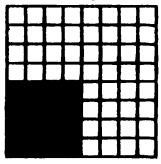
377 $DBC8



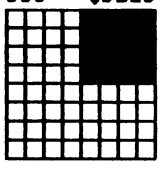
378 $DBD0



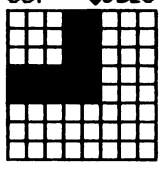
379 $DBD8



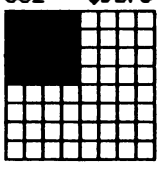
380 $DBE0



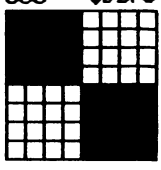
381 $DBE8



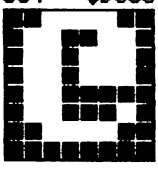
382 $DBF0



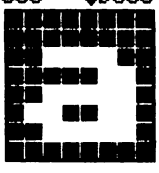
383 $DBF8



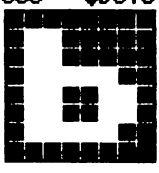
384 $DC00



385 $DC08



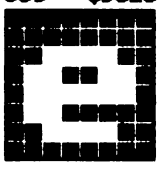
386 $DC10



387 $DC18

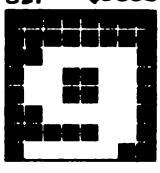
388 $DC40

389 $DC28

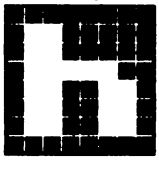


390 $DC30

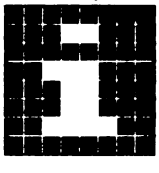
391 $DC38



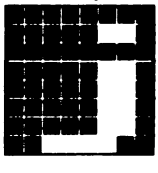
392 $DC40



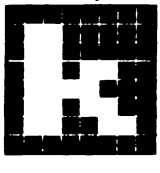
393 $DC48



394 $DC50

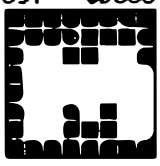


395 $DC58

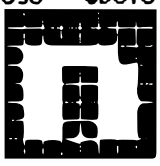


396 $DC60

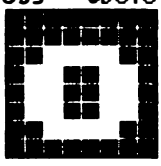
397 $DC68



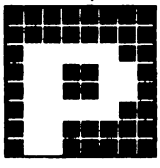
398 $DC70



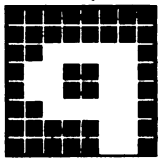
399 $DC78



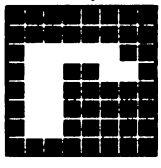
400 $DC80



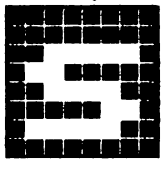
401 $DC88



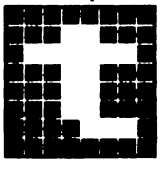
402 $DC90



403 $DC98

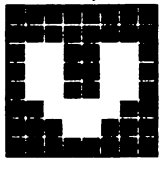


404 $DCA0

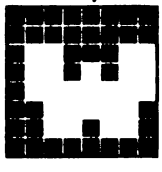


405 $DCA8

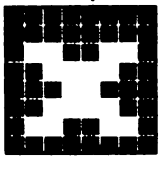
406 $DCB0



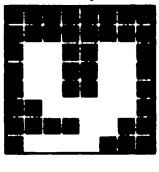
407 $DCB8



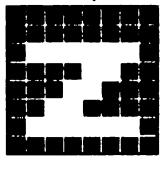
408 $DCC0



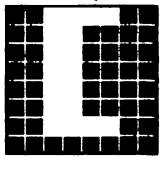
409 $DCC8



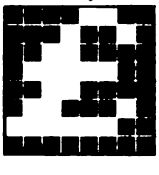
410 $DCD0



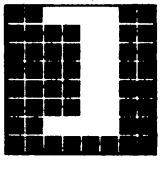
411 $DCD8



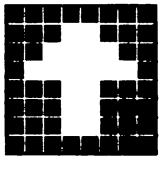
412 $DCE0



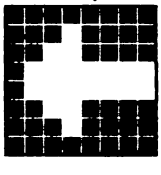
413 $DCE8

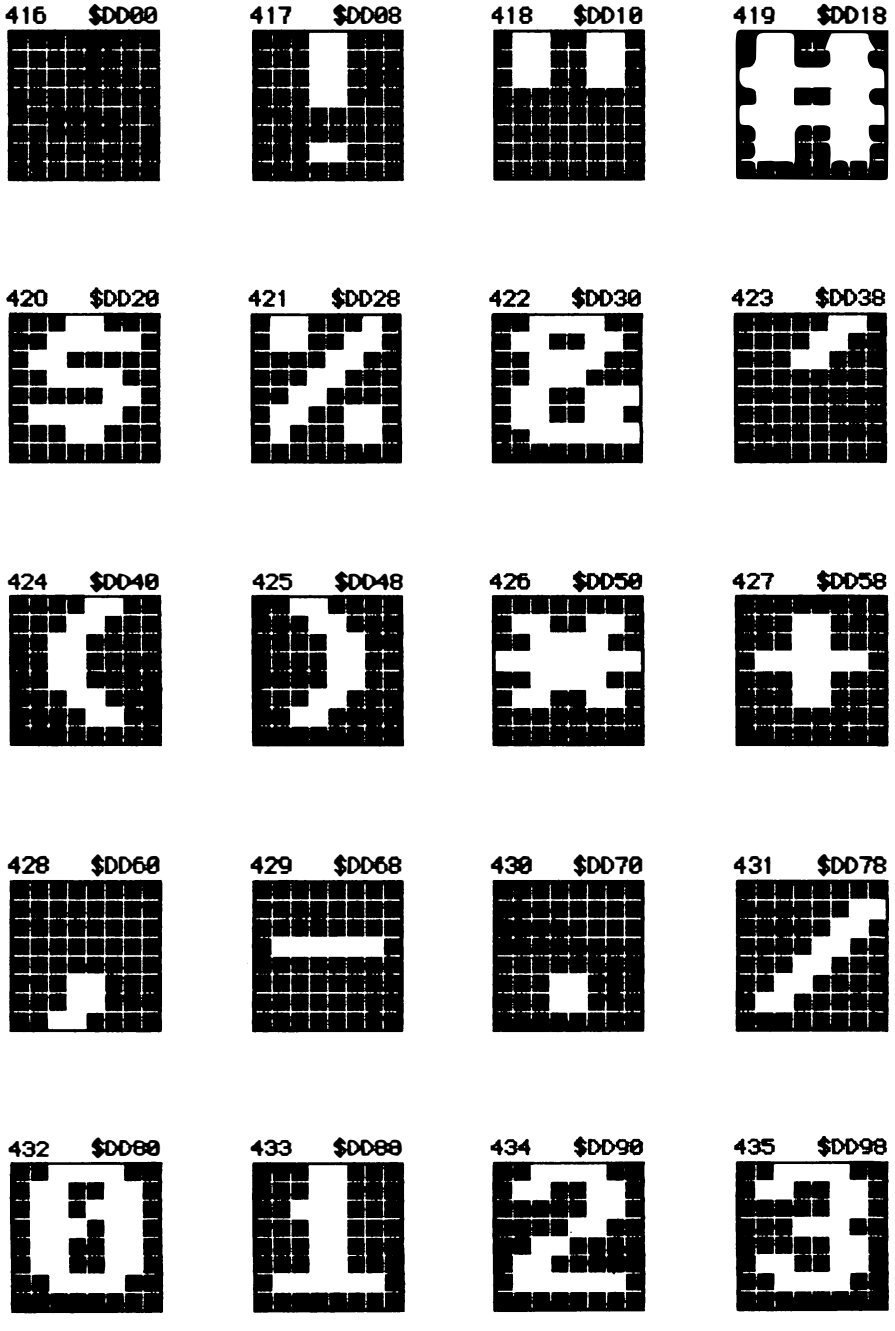


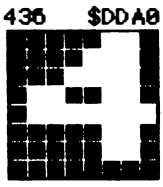
414 $DCF0

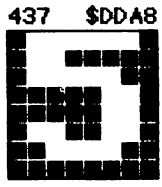


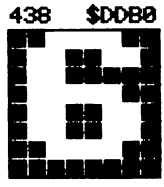
415 $DCF8

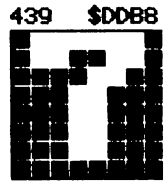


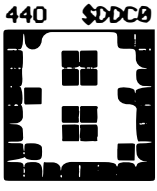

416 $DD00
417 $DD08
418 $DD10
419 $DD18
420 $DD20
421 $DD28
422 $DD30
423 $DD38
424 $DD40
425 $DD48
426 $DD50
427 $DD58
428 $DD60
429 $DD68
430 $DD70
431 $DD78
432 $DD80
433 $DD88
434 $DD90
435 $DD98

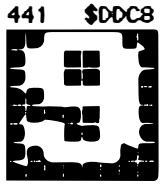

436 $DDA0

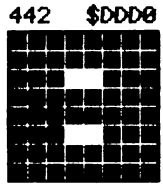

437 $DDA8

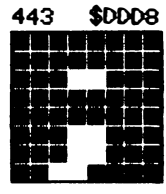

438 $DDB0

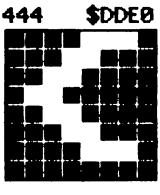

439 $DDB8

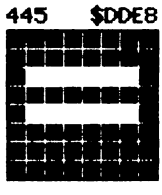

440 $DDC0

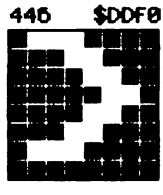

441 $DDC8

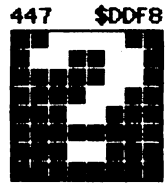

442 $DDD0

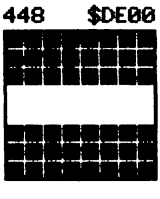

443 $DDD8

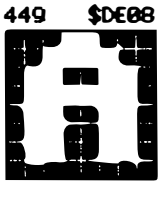

444 $DDE0

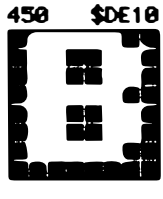

445 $DDE8

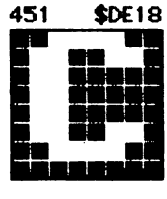

446 $DDF0

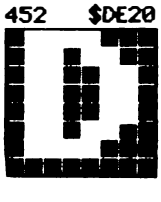

447 $DDF8

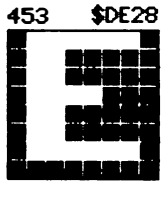

448 $DE00

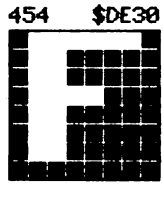

449 $DE08

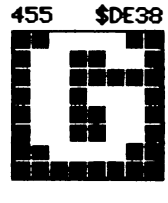

450 $DE10


451 $DE18

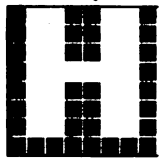

452 $DE20


453 $DE28

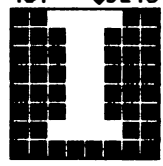

454 $DE30


455 $DE38

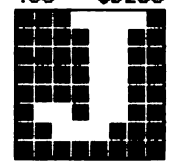
456 $DE40



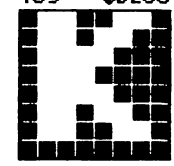
457 $DE48



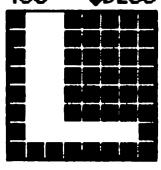
458 $DE50



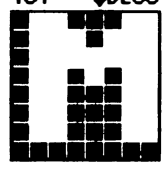
459 $DE58



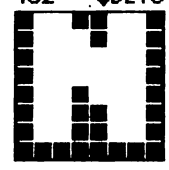
460 $DE60



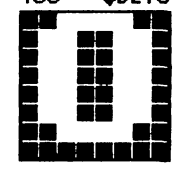
461 $DE68



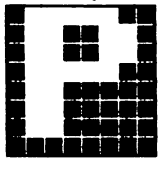
462 $DE70



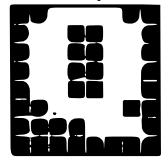
463 $DE78



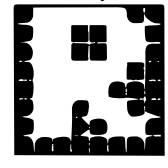
464 $DE80



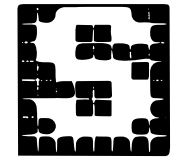
465 $DE88



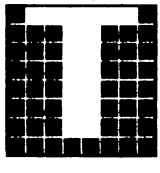
466 $DE90



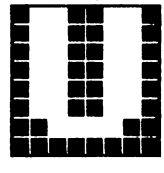
467 $DE98



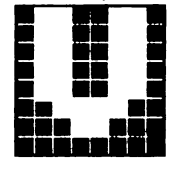
468 $DEA0



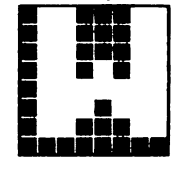
469 $DEA8



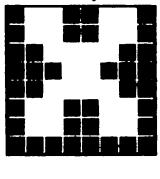
470 $DEB0



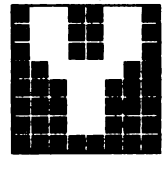
471 $DEB8



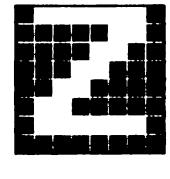
472 $DEC0



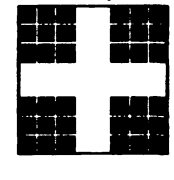
473 $DEC8



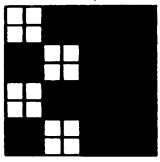
474 $DED0



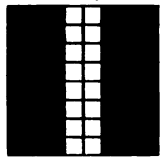
475 $DED8



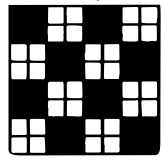
476 $DEE0



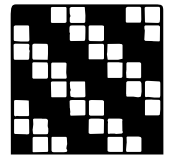
477 $DEE8



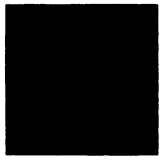
478 $DEF0



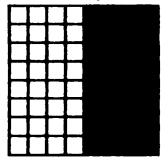
479 $DEF8



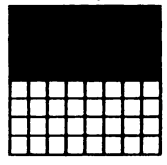
480 $DF00



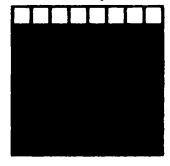
481 $DF08



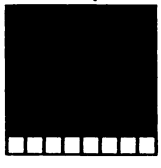
482 $DF10



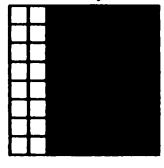
483 $DF18



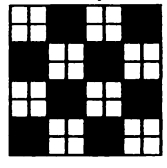
484 $DF20



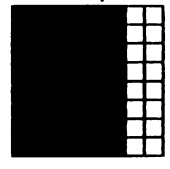
485 $DF28



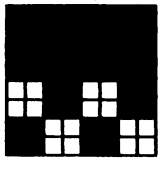
486 $DF30



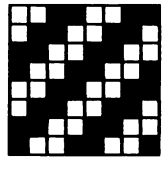
487 $DF38



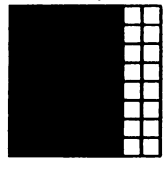
488 $DF40



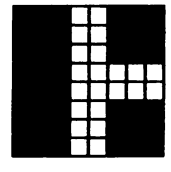
489 $DF48



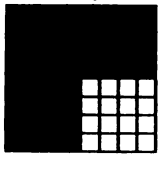
490 $DF50



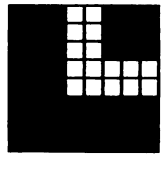
491 $DF58



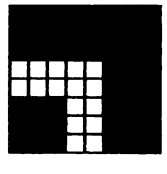
492 $DF60



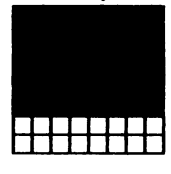
493 $DF68

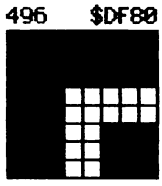


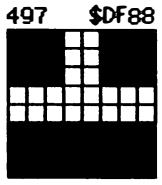
494 $DF70

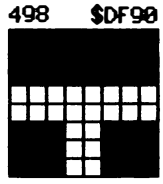


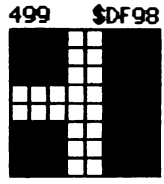
495 $DF78

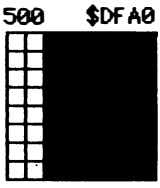


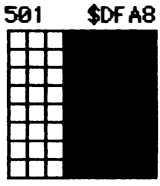

496 $DF80

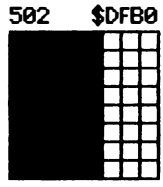

497 $DF88

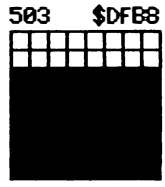

498 $DF90

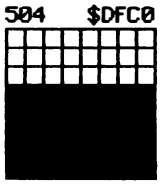

499 $DF98

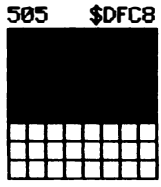

500 $DFA0

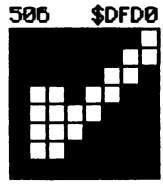

501 $DFA8

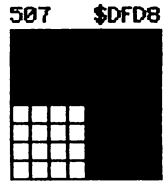

502 $DFB0

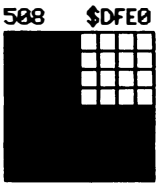

503 $DFB8

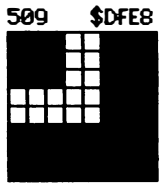

504 $DFC0

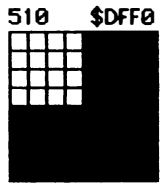

505 $DFC8

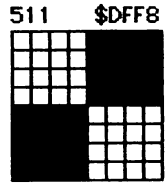

506 $DFD0


507 $DFD8


508 $DFE0


509 $DFE8


510 $DFF0


511 $DFF8

5.2 ROM-Routinen

Das im ROM abgelegte Betriebssystem des 64er enthält viele nützlicje Routinen, mit denen Sie sich das Programmieren von eigenen Maschinenroutinen sehr erleichtern können.

Zwei Typen von ROM-Routinen sind zu unterscheiden. Das sind zum einen die Basic-Routinen für Variablenverwaltung, Fließkomma-Arithmetik, zur Behandlung von Basic-Fehlermeldungen und ähnlichem. Diese Routinen werden Sie also vor allem dann verwenden, wenn Sie eng mit Basic zusammenarbeiten wollen und insbesondere auch Parameter von oder an Maschinenprogramme durch Basic-Variablen übergeben wollen.

Der andere Typ wird durch die sogenannten KERNAL-Routinen repräsentiert, welche sich mit der Ein/Ausgabe, mit der Bildschirmverwaltung, dem seriellen Bus und anderem beschäftigen. Die Parameterübergabe an diese Routinen ist wesentlich besser normiert, als bei den Basic-Routinen. Das erklärt sich dadurch, daß auf dieses Betriebssystem auch dann zugegriffen werden kann und muß, wenn der Basic-Interpreter ausgeschaltet ist und eine andere Programmiersprache geladen ist oder Sie selbst sich vollkommen von Basic lösen und nur die Ein- und Ausgabe mit den vom Betriebssystem vorgesehenen Routinen bewerkstelligen wollen.

5.2.1 KERNAL-Routinen

Für die Betriebssystem-Routinen des KERNAL existiert eine standardisierte Sprungtabelle, die am Ende des ROM-Bereiches abgelegt ist. Dort sind die Adressen von 39 Betriebssystem-Routinen festgelegt. Die Verwendung einer Sprungtabelle hat den Vorteil, daß Maschinenprogramme, die nur diese Sprungtabelle verwenden, nicht geändert werden brauchen, wenn Sie auf einen anderen Commodore-Rechner oder auf eine neue Version des Betriebssystems umgeschrieben werden sollen. Ein Betriebssystemaufruf gliedert sich prinzipiell in folgende Abschnitte:

Parameter bereitstellen

Die notwendigen Angaben werden einer KERNAL-Routine meist durch Prozessorregister übermittelt.

Routine aufrufen

Grundsätzlich ist dies ein Aufruf mit JSR Adresse.

Fehler behandeln

Manche Routinen melden einen Fehlerfall, indem sie das Carry-Flag setzen. Diese Fälle beeinflussen den logischen Ablauf des Hauptprogramms.

Ergebnis auswerten

Eventuell werden in den Prozessorregistern Ergebnis parameter übergeben.

Die oben erwähnten Fehler treten vor allem bei Ein-/Ausgabe-Operationen auf. Wurde der Fehler durch ein gesetztes Carry-Flag nach der Rückkehr vom KERNAL-Unterprogramm gemeldet, steht die Fehlernummer dann im Akkumulator. Folgende Fehlernummern sind möglich:

Fehlernummer	**Bedeutung**
0	Routine durch STOP-Taste unterbrochen
1	Zuviele Dateien sind offen
2	Die Datei ist bereits offen
3	Die Datei ist nicht offen
4	Datei nicht gefunden
5	Gerät nicht vorhanden
6	Die Datei ist keine Eingabe-Datei
7	Die Datei ist keine Ausgabe-Datei
8	Dateiname fehlt
9	Ungültige Geräteadresse
240	Die Speicherobergrenze wurde durch den RS-232 Puffer verändert

Es ist zu beachten, daß manche Ein- / Ausgabe-Routinen nicht diese Fehlermeldungen benützen, sondern man selbst durch den Aufruf der Routine READST den Eingabestatus feststellen muß.

Adresse Hex.	Adresse Dez.	Name	Zweck	Ver. Reg.	Parameter	Bemerkungen
FFA5	65445	ACPTR	Hole 1 Byte v. seriellen Bus	A,X	a:A	Vorher TALK und evtl. TKSA anwenden; Fehler über READST
FFC6	65478	CHKIN	Öffne Kanal für Eingabe	A,X	e:X	Eingabeparam. ist log. Filenummer; vorher OPEN anwenden; beinhaltet TALK und TKSA
FFC9	65481	CHKOUT	Öffne Kanal für Ausgabe	A,X	e:X	Eing.par. ist log. Filenr.; vorher OPEN anwenden; beinhaltet LISTEN und SECOND
FFCF	65487	CHRIN	Hole ein Zeich. vom aktuellen Eingabekanal	A,X	a:A	Für Eingabe nicht von Tastat. muß vorher CHKIN angewendet werden. Bei Tastatur kann ganze Zeile geholt werden, wenn Routine sooft aufgerufen wird, bis $OD gelesen wird. Beim 1. Mal blinkt Cursor.
FFD2	65490	CHROUT	Ein Zeichen auf Ausgabekanal	A	e:A	Für Ausgabe nicht auf BS muß vorher CHKOUT angew. werden
FFA8	65448	CIOUT	Ein Zeichen auf seriellen Bus		e:A	Vorher ist LISTEN und evtl. SECOND anzuwenden; Fehler über READST bestimmen
FF81	65409	CINT	Bildschirm und VIC initialis.	A,X,Y		Stellt Standardwerte im VIC her; löscht Bildschirm
FFE7	65511	CLALL	Schließt alle Dateien	A,X		Setzt Anz. off. Files auf 0; schließt Kanäle; Eingabeger.=Tastatur, Ausgabe=Bildschirm
FFC3	65475	CLOSE	Schließt eine Datei	A,X,Y	e:A	Eingabepar. ist log. Filenr. im Akku; entspr. Basbef.CLOSE

Adresse Hex.	Adresse Dez.	Name	Zweck	Ver. Reg.	Para meter	Bemerkungen
FFCC	65484	CLRCHN	Schließt Kanäle	A,X		Schließt E/A-Kanal; wenn nicht angewendet, können evtl mehrere Geräte aktiv bleiben.
FFE4	65508	GETIN	Hole ein Zeich. aus Tast.puffer oder von RS-232	A,X,Y	a:A	Tastatur:Puffer leer,dann A=0 RS-232: Status muß mit READST gelesen werden
FFF3	65523	IOBASE	Definiert Page f.I/O-Chip CIA1	X,Y	a:X,Y	Liefert immer Adresse (X=Low, Y=High) des ersten I/O Chips
FF84	65412	IOINIT	Initialisiert I/O-Chips	A,X,Y		Setzt alle E/A-Chips und Routinen in definierten Zustand
FFB1	65457	LISTEN	Kommando an Gerät am seriellen Bus	A	e:A	Eing.par. ist Geräteadr. 0-31 Gerät wartet dann auf Daten Fehler über READST bestimmen
FFD5	65493	LOAD	Lade Daten von Eingabegerät (nicht Tastatur oder RS-232) in den Speicher	A,X,Y	e:A e:X,Y a:X,Y	Akku=0 f. Load; A=1 f. Verify Wenn Sekundäradresse(SA)=0, dann werden Daten ab der Adr. in X,Y gespeichert; wenn SA=1 dann Startadresse aus Header; X,Y enth. letzte gesp. Adr.
						Vorher Fileparameter mit SETLFS und SETNAM setzen
FF9C	65436	MEMBOT	Hole/Setze Beginn von verfügbaren RAM	X,Y	a:X,Y e:X,Y	Wenn Carry=1, dann Adr. lesen Wenn Carry=0, dann Startadresse RAM setzen
FFC0	65472	OPEN	Öffne logische Datei	A,X,Y		Vorher SETLFS und SETNAM anw. Fehler auch über READST lesen
FFF0	65520	PLOT	Hole/Setze Cursor-Position	A,X,Y	X,Y	X=Zeile;Y=Spalte; Wenn Carry=1,dann holen; C=0,dann setzen

Adresse Hex.	Adresse Dez.	Name	Zweck	Ver. Reg.	Para meter	Bemerkungen
FF87	65415	RAMTAS	RAM testen und MEMBOT/MEMTOP setzen	A,X,Y	e:A	Löscht Speicherbereich $0000 bis $0101 und $0200 bis $03FF; setzt Video-Ram ab $0400
FFDE	65502	RDTIM	Hole Zeit (in 1/60 sec.)	A,X,Y	a: A,X,Y	Holt Zeit; MSB im Akku; nächstes Byte im X-Reg.; LSB in Y
FFB7	65463	READST	Hole Statusbyte	A	a:A	Übergibt I/O-Status im Akku
FF8A	65418	RESTOR	Setze Vektoren	A,X,Y		Standardwerte für KERNAL-Vektoren ($0314 - $0333) setzen
FFD8	65496	SAVE	Speichern	A,X,Y	e:A e:X,Y	Akku zeigt auf Adr. in Zero-page, wo Startadr.(L/H) steht X,Y = Low/High-Byte Endadr.
FF9F	65439	SCNKEY	Tastaturabfrage	A,X,Y		Legt Code von evtl. gedrückter Taste in Tastaturpuffer
FFED	65517	SCREEN	Bildsch.Format	X,Y	a:X,Y	setzt X auf 40, Y auf 25
FF93	65427	SECOND	Sekundäradr. nach LISTEN	A	e:A	Schickt Sekundäradresse auf seriellen Bus.
FFBA	65466	SETLFS	Setzt Filepara-meter		e:A e:X e:Y	Akku = log. Dateinummer X = Gerätenummer Y = Sek.adr (keine SA = 255)
FF90	65424	SETMSG	Systemmeldungen ein-/auschalten	A	e:A	Bit 7 aktiviert Fehlermeld. Bit 6 akt. Systemanweisungen
FFBD	65469	SETNAM	Filenamen setzen		e:A e:X,Y	Akku enth. Länge des Namens X,Y=Low/High Startadr. Name
FFDB	65499	SETTIM	Uhrzeit setzen (in 1/60 sec.)		e: A,X,Y	3 Byte (MSB im Akku, mittl. in X, LSB in Y) enth. Zeit
FFA2	65442	SETTMO	Timeoutflag f. IEEE-Karte (nur bei IEEE!)		e:A	Bit 7 enth. Timeoutflag; 0=ermögliche Timeoutabbruch 1=kein Timeoutabbruch

Adresse Hex.	Adresse Dez.	Name	Zweck	Ver. Reg.	Para meter	Bemerkungen
FFE1	65505	STOP	prüft auf STOP-Taste	A,X	a:A,Z	Zeroflag=1, wenn STOP gedr. Akku enth. letzte Zeile der Tastaturabfrage-Matrix
FFB4	65460	TALK	Talk-Kommando auf ser. Bus	A	e:A	Akku enthält Gerätenummer (0-31); Fehler über READST
FF96	65430	TKSA	Sekundäradresse nach TALK send.	A	e:A	Akku enthält Sekundäradresse Diese muß vorher mit $60 OR-verknüpft werden
FFEA	65514	UDTIM	Zeit erhöhen; Stoptaste prüf.	A,X		Erhöht Zeit um 1/60 Sec.; fragt STOP-Taste ab
FFAE	65454	UNLSN	Unlisten-Komm. auf ser. Bus	A		Geräte, die Daten empfangen werden abgemeldet
FFAB	65451	UNTALK	Untalk-Kommando auf ser. Bus	A		Geräte, die Daten senden werden abgemeldet
FF8D	65421	VECTOR	System-Vektoren lesen/setzen	A,X,Y	e:X,Y	X,Y enthalten Startadresse der Vektortabelle (32 Byte) Wenn Carry=1, dann Tab. lesen

5.2.2 KERNAL-Vektoren

Im folgenden finden Sie eine Liste der KERNAL-Routinen, die Sie durch eigene Routinen ersetzen oder ergänzen können, wenn Sie die entsprechenden Vektoren verstellen.

Zur Erleichterung des Änderns von bestimmten Vektoren existiert die KERNAL-Routine VECTOR. Wenn Sie diese Routine verwenden wollen, müssen Sie wie folgt vorgehen: Sie reservieren irgendwo im RAM 32 Byte. Die Startadresse dieses Bereiches übergeben Sie im X- und Y-Register (Low- und High-Byte), dann setzen Sie das Carry-Flag und rufen die Routine VECTOR auf. Dann verändern Sie die Vektoren in der Kopie, wobei die Reihenfolge der Vektoren die gleiche ist. Schließlich rufen Sie die Routine nochmals mit dem gleichen X- und Y-Register auf, aber mit gelöschtem Carry-Flag. Dann sind die neuen Vektoren gesetzt.

Als Adresse der Vektoren ist in der folgenden Tabelle nur der Offset (Dezimal) zu 788 ($0314) angegeben. Jeder Vektor ist in der Form Low-Byte, High-Byte gespeichert. Die Standardbelegung des Vektors ist in der dritten Spalte der unten abgebildeten Tabelle angegeben.

Offset	Funktion	Standard	Bemerkung
0	IRQ	$EA31	Vor Veränderung dieses Vektors müssen die Interrupts verhindert werden. Vor dem Sprung über diesen Vektor wurden vom Betriebssystem bereits Akku, X-Register und Y-Register auf den Stapel gelegt, so daß eine selbstgeschriebene IRQ-Routine nicht mit RTI enden darf, sondern mit JMP $AE7E, wobei gleich das IRQ-Bit im CIA1 gelöscht wird.
2	BRK	$FE66	Für Prozessor-Befehle BRK ($00); Akku, X, Y wie bei IRQ

4	NMI	$FE47	NMI's können vom CIA2 oder von der RESTORE-Taste ausgelöst werden. Akku, X, Y wurden nicht gesichert, es wurde nur das Interrupt-Disaple-Flag gesetzt.
6	OPEN	$F34A	Open-Routine
8	CLOSE	$F291	Close-Routine
10	CHKIN	$F20E	Eingabe-Gerät setzen
12	CKOUT	$F250	Ausgabe-Gerät setzen
14	CLRCH	$F333	KERNAL-Routine CLRCH
16	CHRIN	$F157	Eingabe eines Zeichens
18	CHROUT	$F1CA	Ausgabe eines Zeichens
20	STOP	$F6ED	Abfrage der STOP-Taste
22	GETIN	$F13E	Holen eines einzelnen Zeichens
24	CLALL	$F32F	Schließen aller Dateien
26	WARM	$FE66	Warmstart
28	LOAD	$F4A5	Load; Startadresse in $C3,$C4
30	SAVE	$F5ED	Save; Startadresse in $C1,$C2, Endadresse in $AE,$AF

Die Routine zur Tastaturdekodierung ist zwar auch eine Betriebssystemroutine, wir wollen den entsprechenden Vektor aber bei den Basic-Vektoren beschreiben, da dieser nicht mit der VEKTOR-Routine angesprochen wird.

5.2.3 Basic-Routinen

Außer den KERNAL-Routinen spielen die Routinen des Basic-Interpreters eine wichtige Rolle, vor allem dann, wenn die Maschinenprogramme mit Basic zusammanarbeiten sollen.

Wir haben im folgenden die wichtigsten Einsprungstellen in den Bereich des Basic:-Interpreters aufgelistet. In dieser Tabelle finden Sie außer der dezimalen und hexadezimalen Adresse noch einen geläufigen Namen und eine Spalte mit Bemerkungen, wo vor allem die notwendigen Parameter aufgeführt sind. Die genaue Funktion der Basic-Routinen entnehmen Sie bitte einem ROM-Listing. Es ist empfehlenswert, sich z.B. mit dem in #6# beschriebenen Disassembler einige Routinen oder Tabellen des Basic-ROM's ausdrucken zu lassen. Die dort verwendeten Adressen können Sie leicht entschlüsseln, wenn Sie die folgenden Tabellen und die in Kapitel 5.1.1 abgebildeten Zero-Page-Adressen mitverwenden.

Häufige Abkürzungen in den folgenden Tabellen sind:

FAC	Fließkommaakkumulator; 6-Byte-Register ab $61
ARG	Argument-Register für Arithmetik; 6 Byte ab $69
STRTOP	Derjenige String, dessen Deskriptor als letzter auf den String-Stack ($19-$21) gelegt wurde

Die Marken für Basic-Befehle beginnen immer mit 'BB'. Die Marken für Basic-Funktionen tragen den Namen der Funktion selbst.

Die Verwendung der Basic-Befehle als Unterprogramme für selbstgeschriebene Maschinenprogramme ist manchmal nützlich. Zum Beispiel kann der OPEN-Befehl geschickt verwendet werden in einer Routine, die anschließend noch Daten auf die neue Datei schreibt. Der Aufruf sieht dann etwa so aus:

```
SYS Adr,LfNr,Ga,Sa,"Dateiname"
```

Das dazugehörige Assemblerprogramm sähe dann so aus:

```
Adr: JSR  CHKCOM      ; prüfe, ob Komma folgt
     JSR  BBOPEN      ; Basic-Befehl OPEN aufrufen
     NOP              ; Hier  steht Programm  zur Bearbei-
     NOP              ; tung der Datei
     LDA  $49         ; log. Dateinummer
     JSR  CLOSE       ; KERNAL-Routine CLOSE schließt
                      ; Datei wieder
     RTS              ; Unterprogramm fertig
```

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
0073	115	CHRGET	Hole Zeichen aus Basic-Text; A:Akku=Zeichen bzw. Code
			Carry-Flag ist gesetzt, wenn Zeichen keine Ziffer ist
0079	121	CHRGOT	Hole aktuelles Zeichen des Basic-Textes; A: siehe CHRGET
A38A	41866	STAPSUCH	Stapelsuchroutine für FOR...NEXT und GOSUB
A3B8	41912	BLOCKMOV	Blockverschieberoutine E: $5F/$60 Alter Blockanfang
			E: $5A/$5B Altes Blockende+1; $5F/$60 Neues Blockende+1
A3FB	41979	STAPVOLL	prüft auf Platz im Stapel; wenn voll, dann 'out of mem.'
			E: Halbe Zahl der erforderlichen Bytes im Akku
A408	41992	MAKESPC	schafft Platz im Speicher für neue Prg.zeilen u. Variable
			E: A/Y Adresse, bis zu der Platz benötigt wird
A435	42037	ERROUTM	Ausgabe von 'out of memory'
A437	42039	ERROR	Fehlermeldung ausgeben E: Fehlernummer im X-Register
A474	42823	READY	'READY.' ausgeben und Eingabe erwarten
A480	42112	INLOOP	Eingabe-Warteschleife
A49C	42140	PRGZLE	Löschen und Einfügen von Programmzeilen
A4A9	42153	PRGZLOE	Löschen einer Programmzeile
A533	42291	PRGZBIND	Basic-Programmzeilen neu binden (Vorwärtszeiger neu ber.)
A560	42336	GETZEIL	holt eine Zeile vom Bilds. in den Eingabepuffer ab $0200
A571	42353	ERRSTRTL	Ausgabe von 'string too long'
A579	42361	PRGZUM	Umwandlung einer Programm-Zeile im Puffer ab $0200
			in den Interpreterkode
A613	42515	PZSUCHB	Startadresse einer Basic-Zeile suchen (ab Prg.beginn)
			E:$14,$15 Zeilennr.;A:$5F,$60 Adresse;Carry=1,wenn gef.
A617	42519	PZSUCHAX	wie PZSUCHB, jedoch ab Adresse in Akku,x (L,H)
A642	42562	BBNEW	Basic-Befehl NEW
A65E	42590	BBCLR	Basic-Befehl CLR
A68E	42638	PRGBEGIN	Programmzeiger auf Basic-Start setzen
A69C	42652	BBLIST	Basic-Befehl LIST
A717	42775	KLARTEXT	Interpreterkode in Befehlswort umwandeln und ausgeben

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
A742	42818	BBFOR	Basic-Befehl FOR
A7AE	42926	INTLOOP	Interpreterschleife führt Basic-Befehle aus; prüft auf STOP-Taste; setzt u.a. Zeiger für CONT
A7ED	42989	BASBEF	führt einen Basic-Befehl aus; E:Interpreterkode im Akku
A81D	43037	BBRESTOR	Basic-Befehl RESTORE
A82C	43052	CHSTOP	prüft auf STOP-Taste und bricht evtl. Programm ab
A82F	43055	BBSTOP	Basic-Befehl STOP
A831	43057	BBEND	Basic-Befehl END
A857	43095	BBCONT	Basic-Befehl CONT
A871	43121	BBRUN	Basic-Befehl RUN
A883	43139	BBGOSUB	Basic-Befehl GOSUB
A8A0	43168	BBGOTO	Basic-Befehl GOTO
A8D2	43218	BBRETURN	Basic-Befehl RETURN
A8F8	43256	BBDATA	Basic-Befehl DATA
A906	43270	NEXTST	sucht nächste Anweisung im Basic-Programm; A: Y= Distanz
A909	43273	NEXTLIN	sucht nächste Zeile des Basic-Programms ; A: Y= Distanz
A928	43304	BBIF	Basic-Befehl IF
A93B	43323	BBREM	Basic-Befehl REM
A94B	43339	BBON	Basic-Befehl ON
A96B	43371	GETADRZ	liest Ganzzahl als Ziffern aus Basic-Text; A: $14,$15
A9A5	43429	BBLET	Basic-Befehl LET
A9BB	43450	WERTZUW	Wertzuweisung an beliebige Variable; E: Var.adr $49/$4A; Typ-Flag (String/num.) im Akku; Integerflag im Stack
A9C4	43460	FACVARI	FAC nach Integer-Variable übertragen E: Var.adr $49/$4A
AA2C	43564	STRVAR	Wertzuweisung an String-Variable E: Var.adr $49/$4A ; Adresse des Stringdeskriptors (Lge,AdrL,AdrH) in $64/$65
AA80	43648	BBPRINT1	Basic-Befehl PRINT#
AA86	43654	BBCMD	Basic-Befehl CMD
AAA0	43680	BBPRINT	Basic-Befehl PRINT

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
AB1E	43806	STROUT	String ausgeben E: Akku/Y-Reg = Startadr. String (L/H)
			Endekennzeichen ist CHR$(0)
AB21	43809	STROUT1	String ausgeben, wenn String durch FRMEVL ausgewertet
AB3B	43835	SPCOUT	Ein Leerzeichen bzw. Cursor right (wenn ($13)=0) ausgeben
AB3F	43839	SPOUT	Ein Leerzeichen ausgeben
AB42	43842	OUTCURI	Ein Zeichen 'Cursor rechts' ausgeben
AB4F	43855	OUTFRAG	Ein Fragezeichen ausgeben
AB4D	43853	FEHLIGR	Fehlerbehandlung bei INPUT/GET/READ
AB7B	43899	BBGET	Basic-Befehl GET
ABA5	43941	BBINPUT1	Basic-Befehl INPUT#
ABA0	43936	BCLRCH	beendet Eingabe durch Aufruf von CLRCH und setzt $13 zur.
ABBF	43967	BBINPUT	Basic-Befehl INPUT
AC06	44038	BBREAD	Basic-Befehl READ
AD1D	44317	BBNEXT	Basic-Befehl NEXT
AD8A	44426	FRMNUM	holt Ausdruck aus Basic-Text und prüft auf numer. Wert
			Wert steht anschließend im Fließkommaakkumulator (FAC)
AD8D	44429	CHNUM	prüft auf numerisch
AD8F	44431	CHSTR	prüft auf String
AD99	44441	ERRTYPM	Ausgabe von 'type mismatch'
AD9E	44446	FRMEVL	holt und wertet beliebigen Ausdruck aus; Wert steht im
			FAC (num.) oder kann mit FRESTR (str.) geholt werden
AE83	44675	GETARI	arithmetischen Term holen
AEA8	44712	FPI	Fliesskommakonstante Pi = 3.14159265
AED4	44756	NOT	FAC = NOT FAC ; Basic-Funktion NOT
AEF1	44785	FRMKLA	holt Wert in Klammern (= CHKKLAUF & FRMEVL & CHKKLAZU)
AEF7	44791	CHKKLAZU	prüft, ob 'Klammer zu' im Basic-Text folgt
AEFA	44794	CHKKLAUF	prüft, ob 'Klammer auf' folgt
AEFD	44797	CHKCOM	prüft, ob 'Komma' folgt
AEFF	44799	CHKZEI	prüft, ob Zeichen im Akku im Basic-Text folgt

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
AF08	44808	ERRSYNT	Ausgabe von 'syntax error'
AF28	44840	GETVAR	holt Wert von Variable im Basic-Text nach FAC
AFE6	45030	BBOR	Basic-Befehl OR
AFE9	45033	BBAND	Basic-Befehl AND
B016	45078	VERGL	Vergleichsoperation; Ergebnis (0 bzw. -1) steht im FAC
B081	45185	BBDIM	Basic-Befehl DIM
B08B	45195	VARSUC	Variable aus Basic-Text suchen; A: Variablenadr. $47,$48
B0E7	45287	NAMSUC	Sucht Variable; E:Name $45,$46; A: Variablenadr. $47,$48
B113	45331	CHLETT	prüft auf Buchstabe; A:Carry=1,dann Buchstabe
B194	45460	ARRBEG	berechnet Zeiger auf erstes Arrayelement
B1A5	45477	FM32768	Fliesskommakonstante -32768
B1AA	45482	FLPAY	FAC nach Integer wandlen; A: Akku/Y-Reg.= High/Low
B1B2	45490	GETINTN	hole Integer-Wert; A:$65,$64 (L/H); beinhaltet CHRGET
B1BF	45503	FLPINT	wandelt FAC in vorz.beh. 16-Bit-Zahl; A:$65,$64 (L/H)
B245	45637	ERRBADS	Ausgabe von 'bad subscipt'
B248	45640	ILLQERR	Ausgabe von 'illegal quantity'
B34C	45900	ARRSIZ	berechnet Arraygröße
B37D	45949	FRE	FAC = FRE(0) ; beinh. GARCOL ; Basic-Funktion FRE
B39E	45982	POS	FAC = POS(0) ; Basic-Funktion POS
B3A6	45990	CHDIREKT	Test auf Direkt-Modus, wenn ja, dann 'illegel direct err'
B3AB	45995	ERRILLD	Ausgabe von 'illegal direct'
B3AE	45998	ERRUNFN	Ausgabe von 'undef'd function'
B3B3	46003	BBDEF	Basic-Befehl DEF
B3E1	46049	CHFNSYN	FN-Syntax prüfen
B3F4	46068	STR	STRTOP = STR$(FAC) ; Basic-Funktion STR$
B475	46197	STRZEIG	Stringverwaltung, Zeiger auf String berechnen
B487	46215	STRDEF	Neuen String definieren; Stringdescriptor kommt auf Stringstapel; E:Akku/Y = Stringadr.;Stringende b. CHR$(0)

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
B4CA	46282	PUSHSTR	Stringdeskriptor auf Stapel legen; E: Descriptor ab $61: ($61)=Stringlänge; ($62),($63)=Stringadresse
B4F4	46215	NEUSTR	Neuen String einrichten; wenn kein Platz, dann Fehlerm.; E:Akku=Länge des neuen Strings; A: X/Y = Neue Adr.(L/H)
B526	46374	GARCOL	Garbage Collection, nichtgebrauchte Strings entfernen
B63D	46653	STRPLUS	Stringverknüpfung +
B6A3	46755	FRESTR	Holt Stringdeskriptor vom Stapel; Aufruf sinnvoll nach FRMEVL; A:Stringadr. in $22/$23 , Länge im Akku
B6EC	46828	CHR	STRTOP = CHR$(FAC) ; Basic-Funktion CHR$
B700	46848	LEFT	Basic-Funktion LEFT$
B72C	46892	RIGHT	Basic-Funktion RIGHT$
B737	46903	MID	Basic-Funktion MID$
B77C	46973	LEN	FAC = LEN (STRTOP) ; Basic-Funktion LEN
B78B	46987	ASC	FAC = ASC (STRTOP) ; Basic-Funktion ASC
B79B	47003	GETBYTN	Liest Byte-Wert aus Basic-Text; A: X = Byte-Wert
			*: Diese Adresse beinhaltet vorher CHRGET-Aufruf
B79E	47006	GETBYT	wie GETBYTN, aber ohne CHRGET ; A: X = Byte-Wert
B7EB	47083	GETAB	Kombi-Routine: FRMNUM & FACADR & CHKCOM & GETBYT
B7F1	47089	GETBYTC	wie GETBYT, aber mit CHKCOM ; A: X = Byte-Wert
B7F7	47045	FACADR	Wandelt FAC in Adressformat; A: $14,$15 = 16-Bit-Zahl
B7AD	47021	VAL	FAC = VAL (STRTOP) ; Basic-Funktion VAL
B80D	46861	PEEK	FAC = PEEK (FAC) ; Basic-Funktion PEEK
B824	47140	BBPOKE	Basic-Befehl POKE
B82D	47149	BBWAIT	Basic-Befehl WAIT
			*** Fließkomma-Routinen ***
*:Konstante (A/Y) ist Zahl, deren Adr. durch Akku (Low) und Y (High) gegeben ist			
B849	47177	FACPLUO5	FAC = FAC + 0.5
B850	47184	FKMINUS	FAC = Konstante (A/Y) - FAC ; Fließk.-Subtraktion
B853	47187	FAMINUS	FAC = ARG - FAC ; Fließk.-Subtraktion

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
B867	47207	FPLUSK	FAC = Konstante (A/Y) + FAC ; Fließk.-Addition
B86A	47210	FPLUSA	FAC = ARG + FAC ; Fließk.-Addition
B97E	47486	ERROVFL	Ausgabe von 'overflow error'
B9BC	47548	FKLOG	Fliesskommakonstanten für LOG
B9EA	47594	LOG	FAC = LOG (FAC) ; Basic-Funktion LOG
BA28	47656	FMALK	FAC = Konstante (A/Y) * FAC ; Fließk.-Multiplik.
BA2B	47659	FMALA	FAC = ARG * FAC ; Fließk.-Multiplik.
BA8C	47756	FLDARGK	ARG = Konstante (A/Y)
BAE2	47842	FMAL10	FAC = FAC * 10
BAF9	47865	FK10	Fliesskommakonstante 10
BAFE	47870	FDURCH10	FAC = FAC / 10
BB0F	47887	FKDIV	FAC = Konstante (A/Y) / FAC ; Fließk.-Division
BB12	47810	FADIV	FAC = ARG / FAC ; Fließk.-Division
BB8A	48010	DIVZERR	Ausgabe von 'division by zero'
BBA2	48034	FLDFACK	FAC = Konstante (A/Y)
BBC4	48068	FACA4	Fließkomma-Akkumulator #4 mit FAC laden
BBCA	48074	FACA3	Fließkomma-Akkumulator #3 mit FAC laden
BBD0	48080	FACVAR	FAC in Variable (Adr. in $49,$4A) übertragen / Var = FAC
BBD4	48084	FACXY	FAC in Variable (Adr. in X,Y) übertragen
BBFC	48119	ARGFAC	FAC = ARG
BC0C	48140	FACARG	ARG = FAC
BC1B	48155	FACRUND	FAC runden
BC2B	48171	FSGN	Vorzeichen von FAC holen; A: Akku = $FF, wenn FAC negativ; $00, wenn FAC = 0; $01,wenn FAC pos. ; Flags entspr. Akku
BC39	48185	SGN	FAC = SGN (FAC) ; Basic-Funktion SGN
BC3C	48188	AFLP	Akku (vorz.beh. 8-bit) in Fließk. wandeln nach FAC
BC58	48216	ABS	FAC = ABS (FAC) ; Basic-Funktion ABS

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
BC5B	48219	FKVERGL	Konstante (A/Y) mit FAC vergleichen ; A: Akku = $FF, wenn FAC kleiner als Konst.; Akku = 0, wenn FAC = Konst.; Akku = $01, wenn FAC größer als Konst.; Flags entspr. Akku
BC9B	48283	FLPINT	Umwandlung FAC nach Integer (vorzeichenbehaftet) A: Low-Byte in $65; High-Byte in $64
BCCC	48332	INT	FAC = INT (FAC) ; Basic-Funktion INT
BCF3	48371	ASCFLP	Umwandlung ASCII nach Fließkomma (aus Basic-Text)
BDB3	48563	F99P9	Fliesskommakonstante 99999999.9
BDB8	48824	F9999	Fliesskommakonstante 999999999
BDBD	48573	F1E9	Fliesskommakonstante 1000000000
BDC2	48578	ERRZNR	Ausgabe der Zeilennummer bei Fehlermeldung
BDCD	48589	ADROUT	Zahl im Adreßbereich ausgeben; E: A/X enth. Hi/Low-Byte
BDDD	48605	FACASC	FAC nach ASCII-Format wandeln; String steht ab $0100
BF71	49009	SQR	FAC = SQR (FAC) ; Basic-Funktion SQR
BF78	49016	FKHOCH	FAC = Konstante (A/Y) hoch FAC ; Fließk.-Potenzierung
BF7B	49019	FAHOCH	FAC = ARG hoch FAC ; Fließk.-Potenzierung
BFE8	49128	F1	Fliesskommakonstante 1
BFED	49133	EXP	FAC = EXP (FAC) ; Basic-Funktion EXP
E043	57411	POLY1	FAC = P(FAC) ;P(x)=a*x + b*(x hoch 3) + c*(x hoch 5) ... Polynomberechnung; E: Akku/Y = Adresse der Koeff.tabelle
E059	57433	POLY	FAC = P(FAC);P(x)=a + b*x + c*(x hoch 2) + d*(x hoch 3).. Polynomberechnung; E: Akku/Y = Adresse der Koeff.tabelle
E097	57495	RND	FAC = RND (FAC) ; Basic-Funktion RND
E107	57607	OUTBREAK	Ausgabe von 'break'
			*** KERNAL-Aufrufe von Basic aus beinhalten Fehlerbehandlung ***
E10C	57612	BSOUT	Aufruf von CHROUT (ein Zeichen ausgeben)
E112	57618	BASIN	Aufruf von CHRIN (Zeichen einlesen)
E118	57624	BCHKOUT	Aufruf von CHKOUT (Ausgabegerät festlegen)
E11E	57630	BCHKIN	Aufruf von CHKIN (Eingabegerät festsetzen)

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
E124	57636	BGETIN	Aufruf von GETIN (ein Zeichen holen)
E12A	57642	BBSYS	Basic-Befehl SYS
E156	57686	BBSAVE	Basic-Befehl SAVE
E15F	57636	BSAVE	Aufruf von SAVE (Bereich speichern)
E165	57701	BBVERIFY	Basic-Befehl VERIFY
E168	57704	BBLOAD	Basic-Befehl LOAD
E1BE	57790	BBOPEN	Basic-Befehl OPEN
E1C1	57639	BOPEN	Aufruf von OPEN (Datei öffnen)
E1C7	57799	BBCLOSE	Basic-Befehl CLOSE
E1CC	57804	BCLOSE	Aufruf von CLOSE (Datei schließen)
E1D4	57812	PARLOSA	Parameter für LOAD und SAVE aus Basic-Text lesen
E206	57793	CHKWZ	prüft, ob weiter Zeichen im Basic-text folgen, wenn nicht wird zur übergeordneten Routine verzweigt
E219	57881	PAROPEN	Parameter für OPEN aus Basic-text lesen
E264	57956	COS	FAC = COS (FAC) ; Basic-Funktion COS
E26B	57963	SIN	FAC = SIN (FAC) ; Basic-Funktion SIN
E2B4	58036	TAN	FAC = TAN (FAC) ; Basic-Funktion TAN
E2E0	58080	FPI2	Fliesskommakonstante PI/2 = 1.57079633
E2E5	58085	F2PI	Fliesskommakonstante 2*PI = 6.28318531
E2EA	58090	FO25	Fliesskommakonstante 0.25
E30E	58126	ATN	FAC = ATN (FAC) ; Basic-Funktion ATN
E37B	58235	BASNMI	Basic-NMI-Einsprung (= erw. Warmstart); wie BASWARM, aber aber incl. CLRCH, BASINI1 (Stack init.,CONT sperren), CLI
E386	58246	BASWARM	Basic-Warmstart (ready.)
E394	58260	BASKALT	Basic-Kaltstart (BVECLAD, div. Zellen in Zero-Page mit Std.-Werten f. Basic besetzen, MELDNEW, dann BASWARM
E3A2	58279		Kopie der CHRGET-Routine; wird durch BASICINI in Zero-Page kopiert
E3BA	58298		Anfangswert fuer RND-Funktion 0.811635157

Adresse Hex.	Adresse Dez.	üblicher Name	Bemerkungen (A = Ausgabeparameter) (E = Eingabeparameter)
E3BF	58303	BASICINI	Basic-Initialisierungsroutine (setzt USR-Vektor/RAMTOP..)
E422	58402	MELDNEW	Einschaltmeldung ausgeben und NEW-Befehl
E447	58439		Tabelle der Standard Basic-Vektoren
E453	58451	BVECLAD	Standard Basic-Vektoren ($0300 bis $030B) laden
E4E0	58592	WART	Wartet auf Commodore-Taste bzw. 8.5 Sekunden
E544	58692	BSCLR	Bildschirm löschen
E566	58726	BSHOME	'Home' durchführen
E5A0	58784	VICINIT	Videocontroller initialisieren / E/A=Tast./Bilds.
E5B4	58804	GETTP	Ein Zeichen aus Tastaturpuffer holen; A:Akku = Zeichen
E684	59012	CHHOCHKM	prüft auf Hochkomma-Kode im Akku; wenn ja, Flag $D4 inv.
E8EA	59626	SCROLL	Bildschirm eine Zeile nach oben rollen
E9E0	59872	FARADR2	wie FARADR; aber auch noch $AC,$AD nach $AE,$AF umw.
E9F0	59888	ZEIADR	lädt $D1,$D2 mit Adresse von BS-Zeile (Video); E:X=Zeile
E9FF	59903	ZEILOE	Bildschirmzeile löschen; E:X = Zeilennr.
EA1C	59932	ZEICH	eine BS-Position bedrucken; E: Akku=BS-Code, X=Farbcode, Y=Spalte ;vorher ist ZEIADR und FARADR aufzurufen
EA24	59940	FARADR	wandelt Video-Adresse in Farbadresse; E:$D1,$D2;A:$F3,$F4
EEB3	61107	WAIT1MS	wartet ca. 1 Millisekunde
F0BD	61629		Tabelle der Betriebssystem-Meldungen
F12B	61739	ERRKOUT	Betriebssystem-Meldung ausgeben; E: Y = Offset zu $F0BD
F30F	62223	SUCHFNR	sucht log. File-Nr.; E:X=Filenr.; A:X=Nummer in File-Tab.
F31F	62239	SETFPARX	setzt Fileparameter auf Werte aus Tabelle; E:X=Nr. in Tab
F5AF	62895	SEARMELD	'searching for name'; Filename: Adr in $BB,$BC; Lge. $B7
F68F	63119	SAVEMELD	'saving name' ausgeben; Filename: Adr in $BB,$BC; Lge $B7
F817	63511	WARTPLAY	wartet auf Play-Taste
F838	63544	WARTRECP	wartet auf Record & Play
FCCA	64714	RECMOTAU	Recorder-Motor aus
FD02	64770	ROMMODUL	prüft auf ROM-Modul in $8000; Zero-Flag gesetzt, wenn ja Kriterium: Text "CBM80" erscheint ab $8003

5.2.4 Basic-Vektoren

Der Basic-Interpreter besitzt einige Schlüssel-Routinen, die immer wieder angesprungen werden. Wenn man diese Routinen durch eigene ersetzt oder ergänzt, kann man die Eigenschaften des Interpreters weitgehend verändern, ohne den ganzen Interpreter neu schreiben zu müssen.

Insgesamt gibt es 8 derartige Routinen, wenn man von der CHRGET/CHRGOT-Routine absieht, die in der Zero-Page steht. Die Routinen sind nach Adressen aufsteigend geordnet.

Tastaturdekodierung

Vektor: $028F,$0290 = 655,656
Standardwert : $EB48

Die Routine wird vom Betriebssystem aufgerufen, wenn die Nummer der gedrückten Taste feststeht. Die Tasten Shift/C=/CTRL wurden bereits in Zelle $028D signalisiert.

Sie können diese Routine verstellen, um z.B. die Funktionstasten zu belegen.

Basic-Warmstart / Fehler

Vektor: $0300,$0301 = 768,769
Standardwert : $EB48

Die Routine wird aufgerufen, wenn irgendein Basic-Fehler (z.B. SYNTAX ERROR) auftritt, wenn STOP/RESTORE gedrückt wurde oder der Prozessorbefehl BRK mit dem Standard-BRK-Vektor ausgeführt wird.

Im X-Register ist Bit 7 gesetzt, wenn kein Fehler auftrat. Sonst steht dort die Nummer des Fehlers (siehe Kapitel 5.11.1).

Wenn Sie diesen Vektor verstellen, muß nicht immer ein Programmabbruch die Folge eines Fehlers im Basic-Programm sein. Sinnvoll ist dies z.B. bei den Fehlern 'DIVISION BY ZERO' oder 'FILE OPEN'.

Ihre Routine verlassen Sie entweder mit einem Sprung auf $EB4B (Fehler normal behandeln bzw. 'READY.', je nach X-Register) oder Sie springen in die Interpreterschleife ($A7AE), wenn kein Programmabbruch erfolgen soll.

Eingabe-Warteschleife

Vektor: $0302,$0303 = 770,771
Standardwert : $A483

Wenn am Bildschirm 'READY.' erscheint, wird anschließend zur Eingabe-Warteschleife verzweigt. Der Rechner wartet dann auf die Eingabe einer Programmzeile oder eines Kommandos.

Sie können an dieser Stelle z.B. die Grafik ausschalten, damit das oben erwähnte 'READY.' auch immer gleich in Klarschrift erscheint, oder Sie programmieren eine Routine zur automatischen Vorgabe der nächsten Zeilennummer.

Kodieren einer Zeile

Vektor: $0304,$0305 = 772,773
Standardwert : $A57C

Hier wird die Befehlszeile, die sich im Puffer ab $0200 befindet, in Interpreterkode umgewandelt. Endekriterium ist das Auftreten des Codes '0'.

Ersetzen Sie die Routine durch Ihre eigene, so muß lediglich das Y-Register mit der Länge der Zeile besetzt werden. Die Routine können Sie mit RTS oder mit JMP $A57C abschließen.

Dekodieren einer Zeile

Vektor: $0306,$0307 = 774,775
Standardwert : $A71A

Diese Routine ist so etwas ähnliches wie die Umkehrung der Kodier-Routine, jedoch wird hier das Ergebnis (der Klartext) sofort am Bildschirm ausgegeben. Außerdem wird nicht die ganze Zeile, sondern nur ein Codezeichen dekodiert, weil die Schleife, die jeweils das nächste Zeichen bereitstellt, außerhalb des Bereichs steht.

Für Ihre Zwecke können Sie den Vektor verstellen, um eigene, kodierte Befehle zu listen. Die Routine sollte mit JMP $A6EF, also mit einem Sprung auf die LIST-Schleife beendet werden.

Befehl ausführen

Vektor: $0308,$0309 = 776,777
Standardwert : $A7E4

Im Moment des Aufrufs dieser Routine steht der Befehlszeiger ($7A,$7B) vor dem nächsten Basic-Code. Diesen können Sie untersuchen und zu eigenen Befehlen verzweigen, wie es auch in unseren Basic-Erweiterungen (Kapitel 2.2.1) geschehen ist.

Arithmetisches Element auswerten

Vektor: $030A,$030B = 778,779
Standardwert : $AE86

Hier wird der Wert eines arithmetischen Elementes, also der Wert einer Variablen, einer Konstanten oder einer Funktion bestimmt. Die Vorgehensweise beim Verstellen des Vektors ist ähnlich wie bei der zuletzt genannten Routine und ist ebenfalls in Kapitel 2.2.1 erläutert.

USR-Funktion auswerten

Vektor: $0311,$0312 = 785,786
Standardwert : $B248

Dieser Vektor steht normalerweise auf 'ILLEGAL QUANTITY'. Mit Hilfe dieses Vektors können Sie aber leicht eigene Funktionen definieren, die Sie von Basic aus mit USR(X) aufrufen können. Wenn Sie keine weiteren Maßnahmen treffen, ist der Typ der USR-Funktion numerisch, und ein numerischer Parameter wird automatisch im Fließkomma-Akkumulator übergeben. Dort muß auch in diesem einfachen Fall der Funktionswert abgelegt werden. Wie Sie andere Parametertypen bearbeiten, entnehmen Sie bitte Kapitel 2.2.1.8.

Beispiel Autostart

Als Beispiel zur Anwendung von Basic-Vektoren wollen wir Ihnen hier ein Hilfsmittel vorstellen, das einerseits dem Programmschutz dient und andererseits die Benutzerfreundlichkeit erhöht.

Wenn Sie das unten abgebildete Programm eingeben und mit GOTO 900 starten, erhalten Sie ein Programm mit dem Namen 'AUTOSTART', das sofort seine Arbeit aufnimmt, wenn Sie es mit

```
LOAD"AUTOSTART",8,1
```

laden. In den Zeilen ab 100 können Sie Ihr Programm eintragen, was Sie mit END abschließen sollten, weil ja sonst ein zweitesmal das Programm gespeichert wird.

Die Zeilen 80 und 90 setzen den beim Autostart verbogenen Vektor wieder zurück; lassen Sie diese Zeilen weg, stürzt der Rechner ab, wenn das Programm abgebrochen oder beendet wird.

Soviel zur Anwendung des Programms, nun zur Theorie. In den Adressen 770, 771 befindet sich der Vektor für die Eingabeschleife, die angesprungen wird, nachdem 'READY.' ausgegeben wurde, also auch nach einem LOAD-Befehl. Den Vektor richten wir auf eine kleine Hilfsroutine ab Adresse 828, die nichts anderes tut, als den RUN-Befehl aufzurufen und anschließend zur Interpreterschleife zu verzweigen.

Dieses Programmstück und den Vektor müssen wir mit dem Basic-Programm zusammen speichern, was ab Zeile 1110 durch Aufruf der KERNAL-Routine SAVE geschieht.

Mit dem Befehl LOAD"Autostart",8,1 laden wir den verbogenen Vektor, die Hilfsroutine und das eigentliche Programm. Am Bildschirm erscheint zwar noch kurz 'READY.', aber die Hilfsroutine startet sofort das Programm.

```
80 POKE 770,131                                          <070>
90 POKE 771,164                                          <087>
100 PRINT"HIER IST DAS TESTPROGRAMM"                     <214>
110 PRINT"2.ZEILE"                                       <036>
120 END                                                  <248>
900 OPEN 1,8,15,"S:AUTOSTART"                            <017>
910 CLOSE 1                                              <098>
1000 PRINT"{CLR}"    :REM BILDSCHIRM LOESCHEN            <086>
1010 POKE 828,169 :REM LDA #                             <183>
1020 POKE 829,0   :REM      0                            <142>
1030 POKE 830,32  :REM JSR                               <133>
1040 POKE 831,113 :REM     $  71                         <093>
1050 POKE 832,168 :REM      A8                           <095>
1060 POKE 833,76  :REM JMP                               <166>
1070 POKE 834,174 :REM     $  AE                         <163>
1080 POKE 835,167 :REM      A7                           <126>
1090 POKE 770,60  :REM EINGABE-VEKTOR                    <201>
1100 POKE 771,3   :REM AUF 828 STELLEN                   <067>
1110 SYS 57812 "AUTOSTART",8                             <110>
1120 POKE 780,193     :REM VEKTOR AUF 193                <128>
1130 POKE 781,PEEK(45):REM SPEICHERN BIS                 <149>
1140 POKE 782,PEEK(46):REM (45),(46)                     <195>
1150 POKE 193,2       :REM SPEICHERN AB                  <001>
1160 POKE 194,3       :REM 3*256 + 2                     <064>
1170 SYS 65496        :REM SAVE-ROUTINE                  <141>
1180 POKE 770,131     :REM STANDARD-WERTE                <100>
1190 POKE 771,164     :REM FUER EINGABE                  <141>
1200 END                                                 <052>
```

5.2.5 Variablenzeiger

Bei Verwendung von reinem Basic sind auch die Variablenzeiger von entscheidender Bedeutung. Sie erlauben z.B. das Retten eines Programmes nach einem NEW-Befehl oder die direkte Manipulation von Variablen, aber auch das Kürzen

des Basic-Bereiches zur Unterbringung von Maschinenprogrammen.

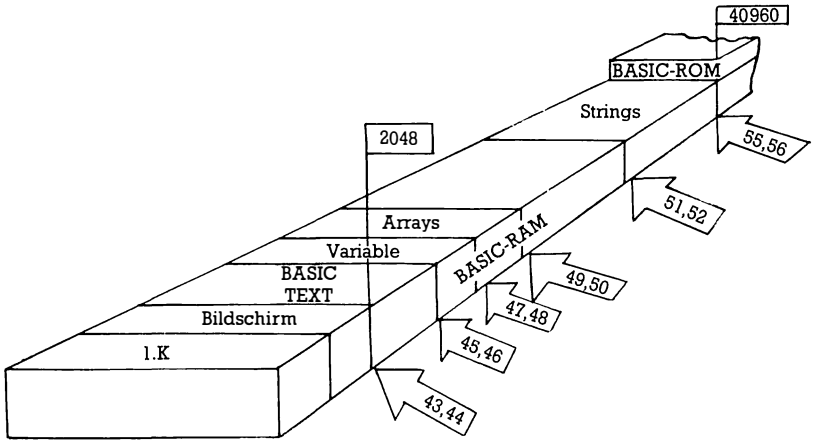


Bild 5.2.5-1 : Schematische Darstellung der Basic-Zeiger

Hexadez. Dezimal Bedeutung

$2B,$2C 43,44 Zeiger auf Basic-Programmstart

Ab der angegebenen Adresse steht das Basic-Programm. Die erste Adresse des Basic-Programmbereiches muß unbedingt mit 0 belegt werden, weil der Interpreter sonst seine Arbeit nicht richtig aufnimmt.

$2D,$2E 45,46 Zeiger auf Beginn der Variablentabelle

Hier steht die Adresse des ersten Eintrages in der Variablentabelle, wobei pro Variable sieben Byte reserviert sind. Die

ersten beiden enthalten den Variablennamen, wobei die beiden MSB's im Variablennamen über den Typ Auskunft geben, da Bit 7 in beiden Zeichen nicht gebraucht wird.

1.Zeichen	2.Zeichen	Typ
Bit 7=0	Bit 7=0	Fließkomma
Bit 7=0	Bit 7=1	String
Bit 7=1	Bit 7=0	-
Bit 7=1	Bit 7=1	Integer

$2F,$30 47,48 Zeiger auf Beginn der Arrays

Der Beginn der Arrays ist in 47 und 48 abgelegt, beginnend mit einem Feldkopf, der wenigstens 7 Byte lang ist. Für jede weitere Dimension wird der Feldkopf um 2 Byte länger. Im Gegensatz zu den einfachen Variablen sind die Werte je nach Typ mit unterschiedlicher Wortlänge gespeichert: Integer-Variablen belegen je zwei Byte, Fließkomma-Variablen je fünf Byte und String-Variablen je drei Byte (ohne den eigentlichen String).

Nach NEW oder CLR wird dieser Zeiger dem Zeiger in $2D,$2E gleichgesetzt.

$31,$32 49,50 Zeiger auf Ende der Arrays

Ab hier beginnt der freie Speicher, dessen Länge mit FRE(0) abgefragt werden kann.

Nach NEW oder CLR wird dieser Zeiger ebenfalls dem Zeiger in $2D,$2E gleichgesetzt.

$33,$34 51,52 Zeiger auf Beginn der Strings

Ab der hier angegebenen Adresse werden schließlich die eigentlichen Zeichenreihen abgelegt.

Dieser Zeiger muß auf den Wert des Zeigers in $37,$38 gesetzt werden, wenn letzterer verändert wird!

$37,$38 55,56 Ende des Basic-RAM's

Dieser Zeiger zeigt auf das Byte nach der letzten von Basic benutzten RAM-Zelle. Wenn Sie diesen Zeiger vermindern, schaffen Sie Platz für Maschinenprogramme oder sonstige Daten, die nicht von Basic aus verändert werden sollen. Vergessen Sie aber nicht, auch den Zeiger auf den Beginn der Strings mitzuverändern oder CLR oder NEW zu schreiben, wodurch auch der Zeiger umbesetzt wird.

5.3 Prozessor-Befehle

Wer als Assembler-Programmierer schnell etwas über einen Befehl wissen will, ist in diesem Kapitel richtig. Auf ausreichende Informationen wird auf entsprechende Fachliteratur verwiesen (z.B. auch #6#). Außerdem sollen Ihnen die Tabellen das Nachvollziehen der in diesem Buch abgebildeten Assembler-Programme erleichtern.

5.3.1 Alphabetische Übersicht der Prozessor-Befehle

ADC - **AD**diere zum Akkumulator mit **C**arry-Flag
AND - UND-Verknüpfung einer Speicherzelle mit dem Akkumulator
ASL - Shift Left; alle Bits um eins nach links schieben von rechts wird eine Null nachgezogen, und das äußerst linke Bit befindet sich anschließend im Carry-Flag

BCC - **B**ranch if **C**arry **C**lear
BCS - **B**ranch if **C**arry **S**et; verzweige, wenn das Carry- Flag gesetzt ist
BEQ - **B**ranch if **EQ**ual; verzweige, wenn Zero-Flag = 1
BIT - Teste Speicherzelle mit Akkumulator
BMI - **B**ranch if **MI**nus; verzweige, wenn Negativ-Flag=1
BNE - **B**ranch if **N**ot **E**qual; verzweige, wenn Zero-Flag=0
BPL - **B**ranch if **PL**us; verzweige, wenn Negativ-Flag =0
BRK - **BR**ea**K**; Abbruch (Software-Interrupt)
BVC - **B**ranch if O**V**erflow **C**lear; verzweige, wenn kein

Überlauf vorhanden ist (Overflow-Flag = 0)
BVS - **B**ranch O**V**erflow Set; verzweige, wenn ein Überlauf vorhanden ist (Overflow-Flag = 1)

CLC - CLear **C**arry; Carryflag löschen
CLD - CLear **D**ecimal Flag; Umschalten auf Binärarithmetik
CLI - CLear **I**nterupt Flag; Interrupt-Flag löschen (Interrupt ermöglichen)
CLV - CLaer O**V**erflowflag; Überlauf-Flag löschen
CMP - Co**MP**are Accu ; vergleiche Akku mit Speicher
CPX - Com**P**are **X** ; vergleiche X-Register mit Speicher
CPY - Com**P**are **Y** ; vergleiche Y-Register mit Speicher

DEC - DECrement Memory ; Speicherinhalt um eins erniedrigen
DEX - **DE**crement **X** ; X-Register um eins erniedrigen
DEY - D**E**crement **Y** ; Y-Register um eins erniedrigen

EOR - **E**xklusiv-**O**de**r**-Verknüpfung Speicherzelle mit Akkumulator

INC - **IN**Crement Memory ; Speicherinhalt um eins erhöhen
INX - **IN**crement **X** ; X-Register um eins erhöhen
INY - **IN**crement **Y** ; Y-Register um eins erhöhen

JMP - **J**u**MP** ; Unbedingter Sprung
JSR - **J**ump **S**ub**r**outine ; Unterprogramm-Aufruf

LDA - Loa**D** **A**ccu ; Lade Akkumulator mit Speicher
LDX - Loa**D** **X** ; Lade X-Register mit Speicher
LDY - Loa**D** **Y** ; Lade Y-Register mit Speicher
LSR - Shift Right;alle Bits um '1' nach rechts schieben, von links wird eine Null nachgezogen, und das äußerst rechte Bit befindet sich anschließend im Carry-Flag

NOP - No **O**peration ; Leerbefehl

ORA - Oder-Verknüpfung Speicherzelle mit Akkumulator

PHA - Pus**h** Accu ; Akku in Kellerspeicher (Stack)
PHP - Push **P**rocessorstatus ; Prozessor-Status-Register in Kellerspeicher
PLA - **P**u**l**l **A**ccu ; Akku aus Kellerspeicher holen
PLP - **P**u**l**l **P**rocessorstatus ; Prozessor-Status-Register aus Kellerspeicher holen

ROL - **R**Otate **L**eft - Rotiere ein Bit linksherum (Akku oder Speicher); Bit 7 geht in das Carry-Flag, das alte Carry-Flag geht nach Bit 0
ROR - **R**Otate **R**ight - Rotiere ein Bit rechtsherum (Akku

oder Speicher); Bit 0 geht in das Carry- Flag, das alte Carry-Flag geht nach Bit 7
RTI - **R**e**T**urn from **I**nterrupt - Rückkehr vom Interrupt; restauriert Programmzähler und Prozessorstatus
RTS - **R**e**T**urn from **S**ubroutine - Rückkehr vom Unterprogramm; restauriert nur Programmzähler

SBC - **Sub**trahiere vom Akkumulator mit **C**arry-Flag
SEC - **SE**t **C**arry; Carryflag setzen
SED - **SE**t **D**ecimal Flag; Umschalten auf Dezimalarithmetik
SEI - **SE**t **I**nterupt Flag; Interrupt-Flag setzen (Interrupt verhindern)
STA - **ST**ore **A**ccu; speichere Akkumulator
STX - **ST**ore **X**; speichere X-Register
STY - **ST**ore **Y**; speichere Y-Register

TAX - **T**ransfer **A**ccu into **X**; Akkumulator in X-Register übertragen
TAY - **T**ransfer **A**ccu into **Y**; Akkumulator in Y-Register übertragen
TSX - **T**ransfer **S**tack-Pointer into **X**; Stackpointer in X-Register übertragen
TXA - **T**ransfer **X** into **A**ccu; Akkumulator in X-Register übertragen
TXS - **T**ransfer **X** into **S**tackpointer; X-Register in Stackpointer übertragen
TYA - **T**ransfer **Y** into **A**ccu; Akkumulator in Y-Register übertragen

5.3.2 Prozessor-Befehle in numerischer Reihenfolge

Wir wollen in diesem Kapitel eine Tabelle angeben, die sämtliche Befehle des 6510-Prozessors enthält. Diese Tabelle ist nützlich, wenn Sie ein kleines Programm per Hand disassemblieren wollen. Aus dieser Tabelle sind auch die nicht verwendeten Codes ersichtlich. Zu jedem Befehl wird neben dem Code die mnemotechnische Bezeichnung mit der entsprechenden Adressierungsart angegeben.

Ab - Absolute, Absolut-X-indizierte, Absolut-Y-indizierte oder indirekte Adressierung (2 Byte folgen)
Zp - Zero-Page-, Zero-Page-X-indizierte, Zero-Page-Y-indizierte, oder vor- bzw. nachindizierte Adressierung (1 Byte folgt)
By - unmittelbare Adressierung (1 Byte folgt)
A - Akkumulator (es folgt kein Byte)
Rel - relative Adressierung (1 Byte folgt)

```
00 BRK
01 ORA (Ab,X)
02 illegal
03 illegal
04 illegal
05 ORA Zp
06 ASL Zp
07 illegal
08 PHP
09 ORA #By
0A ASL A
0B illegal
0C illegal
0D ORA Ab
0E ASL Ab
0F illegal
10 BPL Rel
11 ORA (Zp),Y
12 illegal
13 illegal
14 illegal
15 ORA Zp,X
16 ASL Zp,X
17 illegal
18 CLC
19 ORA Ab,Y
1A illegal
1B illegal
1C illegal
1D ORA Ab,X
1E ASL Ab,X
1F illegal
20 JSR Ab
21 AND (Zp,X)
22 illegal
23 illegal
24 BIT Zp
25 AND Zp
26 ROL Zp
27 illegal
28 PLP
29 AND #By
2A ROL A
2B illegal
2C BIT Ab
2D AND Ab
2E ROL Ab
2F illegal
30 BMI Rel
31 AND (Zp),Y
32 illegal
33 illegal
34 illegal
35 AND Zp,X
36 ROL Zp,X
37 illegal
38 SEC
39 AND Ab,Y
3A illegal
3B illegal
3C illegal
3D AND Ab,X
3E ROL Ab,X
3F illegal
40 RTI
41 EOR (Zp,X)
42 illegal
43 illegal
44 illegal
45 EOR Zp
46 LSR Zp
47 illegal
48 PHA
49 EOR #By
4A LSR A
4B illegal
4C JMP Ab
4D EOR Ab
4E LSR Ab
4F illegal
50 BVC Rel
51 EOR (Zp),Y
52 illegal
53 illegal
54 illegal
55 EOR Zp,X
56 LSR Zp,X
57 illegal
58 CLI
59 EOR Ab,Y
5A illegal
5B illegal
5C illegal
5D EOR Ab,X
5E LSR Ab,X
5F illegal
60 RTS
61 ADC (Zp,X)
62 illegal
63 illegal
64 illegal
65 ADC Zp
```

```
66 ROR Zp
67 illegal
68 PLA
69 ADC #By
6A ROR A
6B illegal
6C JMP (Ab)
6D ADC Ab
6E ROR Ab
6F illegal
7O BVS Rel
71 ADC (Zp),Y
72 illegal
73 illegal
74 illegal
75 ADC Zp,X
76 ROR Zp,X
77 illegal
78 SEI
79 ADC Ab,Y
7A illegal
7B illegal
7C illegal
7D ADC Ab,X
7E ROR Ab,X
7F illegal
8O illegal
81 STA (Zp,X)
82 illegal
83 illegal
84 STY Zp
85 STA Zp
86 STX Zp
87 illegal
88 DEY
89 illegal
8A TXA
8B illegal
8C STY Ab
8D STA Ab
8E STX Ab
8F illegal
9O BCC Rel
91 STA (Zp),Y
92 illegal
93 illegal
94 STY Zp,X
95 STA Zp,X
96 STX Zp,Y
97 illegal
98 TYA
99 STA Ab,Y
9A TXS
9B illegal
9C illegal
9D STA Ab,X
9E illegal
9F illegal
AO LDY #By
A1 LDA (Zp,X)
A2 LDX #By
A3 illegal
A4 LDY Zp
A5 LDA Zp
A6 LDX Zp
A7 illegal
A8 TAY
A9 LDA #By
AA TAX
AB illegal
AC LDY Ab
AD LDA Ab
AE LDX Ab
AF illegal
BO BCS Rel
B1 LDA (Zp),Y
B2 illegal
B3 illegal
B4 LDY Zp,X
B5 LDA Zp,X
B6 LDX Zp,Y
B7 illegal
B8 CLV
B9 LDA Ab,Y
BA TSX
BB illegal
BC LDY Ab,X
BD LDA Ab,X
BE LDX Ab,Y
BF illegal
CO CPY #By
C1 CMP (Zp,X)
C2 illegal
C3 illegal
C4 CPY Zp
C5 CMP Zp
C6 DEC Zp
C7 illegal
C8 INY
C9 CMP #By
CA DEX
CB illegal
```

CC CPY Ab
CD CMP Ab
CE DEC Ab
CF illegal
D0 BNE Rel
D1 CMP (Zp),Y
D2 illegal
D3 illegal
D4 illegal
D5 CMP Zp,X
D6 DEC Zp,X
D7 illegal
D8 CLD
D9 CMP Ab,Y
DA illegal
DB illegal
DC illegal
DD CMP Ab,X
DE DEC Ab,X
DF illegal
E0 CPX #By
E1 SBC (Zp,X)
E2 illegal
E3 illegal
E4 CPX Zp
E5 SBC Zp
E6 INC Zp
E7 illegal
E8 INX
E9 SBC #By
EA NOP
EB illegal
EC CPX Ab
ED SBC Ab
EE INC Ab
EF illegal
F0 BEQ Rel
F1 SBC (Zp),Y
F2 illegal
F3 illegal
F4 illegal
F5 SBC Zp,X
F6 INC Zp,X
F7 illegal
F8 SED
F9 SBC Ab,Y
FA illegal
FB illegal
FC illegal
FD SBC Ab,X
FE INC Ab,X
FF illegal

Raum für Notizen:

Hexadezimal - Dezimal - Umwandlungstabelle

(In der linken Spalte befindet sich das erste Zeichen, in der Kopfzeile das zweite Zeichen. Die beiden rechten Spalten bilden die Grundlage für größere Werte, z.B. $2000 = 8192)

!	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

Beispiele: $7F =127 ; $F7 = 247 ; $B00 = 2816 ; $B000 = 45056
$A77 = 2560+119 = 2679 ; $ABCD = 40960+2816+205 = 43981

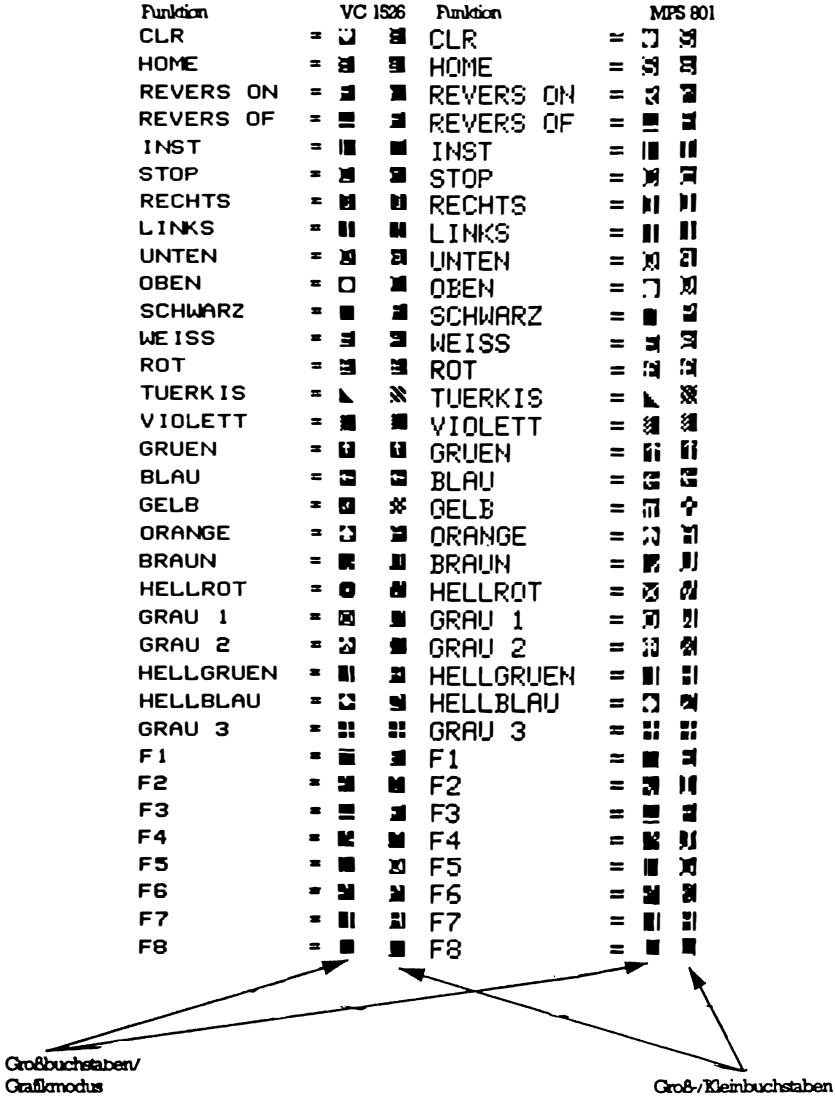
5.5 ASCII- und CHR$-Codes

Nr.	Code	Nr.	Code
0	-----	39	'
1	-----	40	(
2	-----	41	)
3	-----	42	*
4	-----	43	+
5	WEISS	44	,
6	-----	45	-
7	-----	46	.
8	SPERRT SHIFT/C=	47	/
9	ENTSPERRT SHIFT/C=	48	0
10	-----	49	1
11	-----	50	2
12	-----	51	3
13	RETURN-TASTE	52	4
14	UMSCHALTUNG KLEIN-SCHRIFT	53	5
15	-----	54	6
16	-----	55	7
17	CURSOR NACH UNTEN	56	8
18	REVERSE-SCHRIFT EIN	57	9
19	HOME	58	:
20	DEL	59	;
21	-----	60	<
22	-----	61	=
23	-----	62	>
24	-----	63	?
25	-----	64	§
26	-----	65	A
27	-----	66	B
28	ROT	67	C
29	CURSOR NACH RECHTS	68	D
30	GRUEN	69	E
31	BLAU	70	F
32	LEERZEICHEN	71	G
33	!	72	H
34	"	73	I
35	#	74	J
36	$	75	K
37	%	76	L
38	&		

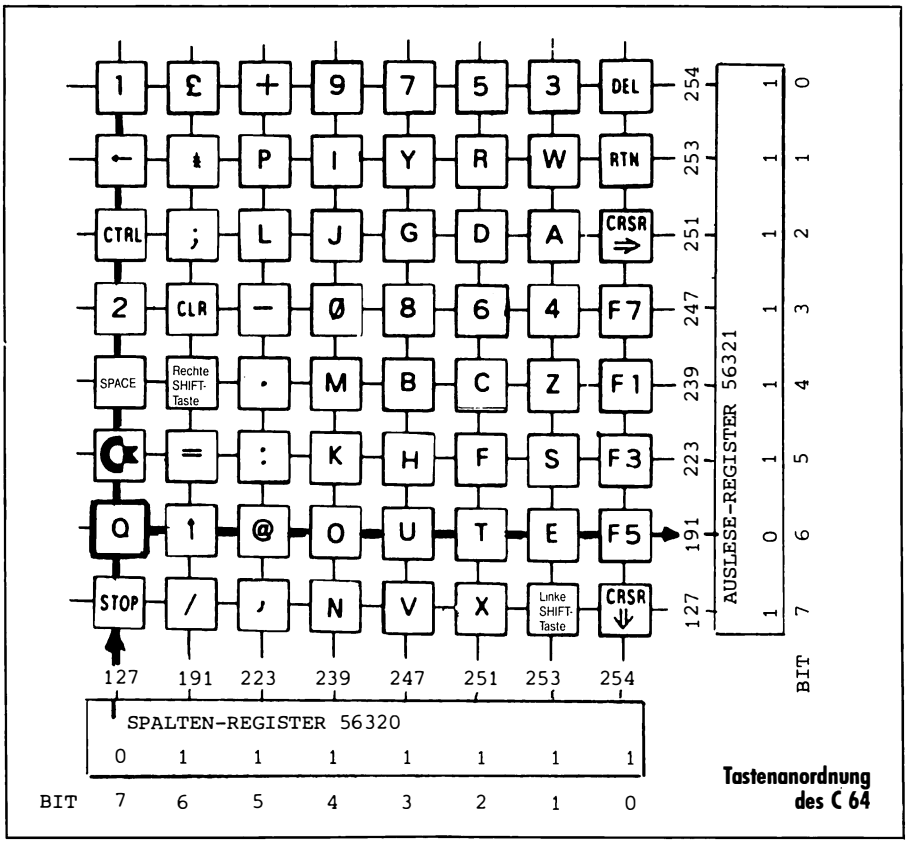
Nr.	Code	Nr.	Code
77	M	122	♠
78	N	123	┼
79	O	124	▒
80	P	125	▏
81	Q	126	▕
82	R	127	▔
83	S	128	
84	T	129	ORANGE (SHIFT/SCHWARZ)
85	U	130	
86	V	131	
87	W	132	
88	X	133	FUNKTIONSTASTE F1
89	Y	134	FUNKTIONSTASTE F3
90	Z	135	FUNKTIONSTASTE F5
91	[	136	FUNKTIONSTASTE F7
92	£	137	FUNKTIONSTASTE F2 (SHIFT F1)
93	]	138	FUNKTIONSTASTE F4 (SHIFT F3)
94	↑	139	FUNKTIONSTASTE F6 (SHIFT F5)
95	←	140	FUNKTIONSTASTE F8 (SHIFT F7)
96	─	141	RETURN-TASTE MIT SHIFT
97	♠	142	UMSCHALTUNG GROSS-SCHRIFT
98	┃	143	
99	▁	144	SCHWARZ
100	▔	145	CURSOR NACH OBEN
101	▔	146	REVERSE-SCHRIFT AUS
102	▁	147	CLR / BILDSCHIRM LOESCHEN
103	▏	148	INST / EINFUEGEN
104	▕	149	BRAUN (SHIFT/WEISS)
105	▗	150	HELLROT (SHIFT/ROT)
106	▝	151	GRAU 1 (SHIFT/CYAN)
107	▂	152	GRAU 2 (SHIFT/PURPUR)
108	▙	153	HELLGRUEN (SHIFT/GRUEN)
109	▝	154	HELLBLAU (SHIFT/BLAU)
110	▗	155	GRAU 3 (SHIFT/GELB)
111	▛	156	PURPUR
112	▜	157	CURSOR NACH LINKS
113	♠	158	GELB
114	▁	159	CYAN
115	♠	160	SHIFT/LEERZEICHEN
116	▏	161	▌
117	▗	162	▀
118	▚	163	▔
119	▖	164	▁
120	♠	165	▏
121	▕	166	▒
		167	▕

Nr.	Code	Nr.	Code	Nr.	Code
168		198	F	228	d
169	▀	199	G	229	e
170	▁	200	H	230	f
171	▂	201	I	231	g
172	▃	202	J	232	h
173	▄	203	K	233	i
174	▅	204	L	234	j
175	▆	205	M	235	k
176	▇	206	N	236	l
177	▉	207	O	237	m
178	▊	208	P	238	n
179	▋	209	Q	239	o
180	▌	210	R	240	p
181	▍	211	S	241	q
182	▎	212	T	242	r
183	▏	213	U	243	s
184	▐	214	V	244	t
185	▒	215	W	245	u
186	▔	216	X	246	v
187	▕	217	Y	247	w
188	▖	218	Z	248	x
189	▗	219	Ä	249	y
190	▘	220	Ö	250	z
191	?	221	Ü	251	ä
192	ß	222	^	252	ö
193	A	223	_	253	ü
194	B	224	`	254	ß
195	C	225	a	255	
196	D	226	b		
197	E	227	c		

5.6 Steuerzeichen auf Drucker



5.7 Tastaturmatrix



5.8 Prioritäten der Operatoren

Die möglichen Operatoren sind nachfolgend nach ihrer Priorität (1-hoch; 9-niedrig) geordnet.

1	Klammern	zur Kennzeichnung des Ablaufes der Abarbeitung
	Arithmetische Operatoren	
2	(Pfeil nach oben)	in diesem Buch:** Potenzierung
3	-	(unäres) Minus
4	*	Multiplikation
4	/	Division
5	+	Addition
5	-	Subtraktion
	Vergleichende Operatoren	
6	=	Gleichheit
6	<>; ><	in diesem Buch: NE Ungleichheit
6	<	in diesem Buch: LT kleiner als
6	>	in diesem Buch: GT größer als
6	>= ;>=	in diesem Buch: GE größer oder gleich
6	<= ;<=	in diesem Buch: LE kleiner oder gleich
	Boolsche Operatoren	
7	NOT	Nicht/Umkehrung
8	AND	logische UND-Verknüpfung
9	OR	logische ODER-Verknüpfung

5.9 Statusübersicht

Bit Dezimal	Lesen von Kassette	Verifizieren auf Diskette	Gerät am seriellen Bus
Bit 0 dez.:1	-	-	Zeitüberschreitung beim Schreiben
Bit 1 dez.:2	-	-	Zeitüberschreitung beim Lesen
Bit 2 dez.:4	Kurzer Block (weniger Bytes als erwartet)	-	-
Bit 3 dez.:8	Langer Block (mehr Bytes als erwartet)	-	-
Bit 4 dez.:16	irreparabler Lesefehler	Abweichung beim Verifizieren	-
Bit 5 dez.:32	Prüfsummenfehler	Prüfsummenfehler	-
Bit 6 dez.:64	Dateiende	Dateiende	Dateiende
Bit 7 dez.:-128	Bandende	Floppy nicht betriebsbereit	Gerät nicht vorhanden

5.10 Gerätenummern und Sekundäradressen

Gerät	Gerätenr.	Sekundäradresse	Tätigkeit
Tastatur	0	-	
Kassettenrekorder 1	1	0	Datei zum Lesen öffnen
		1	Datei zum Schreiben öffnen
		2	Datei zum Schreiben mit Bandendemarkierung öffnen
RS 232 Schnittstelle	2		
Bildschirm	3	-	
Drucker	4 (oder 5)	7	alternativer Zeichensatz
Geräte des Anwenders	(5) 6 7		
Floppy	8 (normal)	0	Programmdatei laden
		1	Programmdatei speichern
		2-14	für Anwender
		15	Kommandokanal
Zweite Floppy	9	siehe Floppy	

Geräte des Anwenders	(9)			
	10			
	11			
nicht verfügbar	12 - 255			

5.11 Fehlermeldungen

Da das Handbuch nur eine Übersicht der Fehlermeldungen bringt, wollen wir hier etwas ausführlicher darauf eingehen. Der Vollständigkeit halber werden im zweiten Unterkapitel die Fehlermeldungen, die durch die Floppy bedingt sind, auch angeführt.

In einem Assembler-Programm können Sie die Fehler aufgrund der Fehlernummer selbst bearbeiten, wenn Sie den Basic-Warmstart-Vektor ($0300/$0301-768/769) verändern.

5.11.1 Basic-Fehlermeldungen (Rechner)

Folgende Punkte sind bei allen Fehlern zu beachten:

- Tritt der Fehler im Direktmodus auf, so wird die Fehlerangabe gefolgt von dem Wort 'ERROR'

- Wird der Fehler im Programm gefunden, so wird die Meldung im Direktmodus gefolgt von 'IN (Zeilennummer)', was eine schnelle Lokalisierung des Fehlers ermöglicht. Ausnahmen sind hier die Hinweise, die ohne Programmabbruch ausgeführt werden:

  REDO FROM START und EXTRA IGNORED

- Nach jeder Fehlermeldung befindet sich der Computer im READY-Modus und erwartet neue Anweisungen

- Die Werte der Variablen bleiben erhalten, was die Fehlerfindung sehr vereinfacht

- Der Stack (Kellerspeicher) des Betriebssystems wird nach Fehlermeldungen zurückgesetzt, so daß Rücksprungadressen für GOSUB-Befehle und FOR...NEXT-Schleifen nicht mehr zur Verfügung stehen.

- CONT ist nach einem Programmabbruch durch einen Fehler deshalb nicht möglich

Alphabetische Übersicht

Nr.	Fehlermeldung	Bedeutung
18	BAD SUBSCRIPT	Angegebener Index nicht DIMensioniert
26	CAN'T CONTINUE	Fortsetzung des Programms nicht möglich
5	DEVICE NOT PRESENT	Gerät nicht am seriellen Bus
20	DIVISION BY ZERO	Division durch Null
24	FILE DATA	Falsche Eingabe-Variable
3	FILE OPEN	Datei bereits offen
4	FILE NOT FOUND	Datei nicht gefunden
2	FILE NOT OPEN	Datei nicht offen
25	FORMULA TOO COMPLEX	Zu komplizierter String-Ausdruck
21	ILLEGAL DIRECT	Unerlaubtes Direktkommando
9	ILLEGAL DIVICE	Falsche Gerätenummer
14	ILLRGAL QUANTITY	Unerlaubter Wert
29	LOAD	Eingelesenes Programm nicht OK (nur bei Kassettenrekorder)
8	MISSING FILE NAME	Dateiname fehlt (leer)
10	NEXT WITHOUT FOR	NEXT ohne FOR
6	NOT INPUT FILE	Keine Eingabedatei (nur bei Kassettenrekorder)
7	NOT OUTPUT FILE	Keine Ausgabedatei (nur bei Kassettenrekorder)
13	OUT OF DATA	Zuwenig Daten in DATA-Zeilen
16	OUT OF MEMORY	Zuwenig Speicher oder Stacküberlauf
15	OVERFLOW	Überlauf des Zahlenbereiches
12	RETURN WITHOUT GOSUB	RETURN ohne GOSUB
19	REDIM'D ARRAY	Feld wurde bereits dimensioniert
23	STRING TOO LONG	Zeichenreihe zu lang
11	SYNTAX	Fehlerhafter Befehl
1	TOO MANY FILES	Zu viele Dateien
22	TYPE MISMATCH	Falsche Variablentypen (String, Zahlen)
17	UNDEF'D STATEMENT	Zeilennummer nicht gefunden
27	UNDEF'D FUNCTION	Funktion nicht definiert
28	VERIFY	Fehler bei Programmüberprüfung

Hinweise (ohne Programmabbruch):

REDO FROM START	Zeichenreihe sollte in Zahlvariable eingelesen werden
EXTRA IGNORED	zuviele ',',';' und ':' in eingegebener Zeichenreihe

Die Fehlermeldungen im einzelnen

18 BAD SUBSCRIPT — **Angegebener Index nicht dimensioniert**

Bei einer indizierten Variablen (Feld, Array) wurde ein Index im Programm verwendet, der nicht dimensioniert wurde. Felder bis zu 10 Elementen brauchen nicht dimensioniert zu werden, jedoch tritt der oben genannte Fehler auf, sobald Sie den Index '11' ansprechen. In der Regel tritt dies in Schleifen auf und bedeutet in nicht seltenen Fällen auch einen logischen Fehler.

26 CAN'T CONTINUE — **Fortsetzung des Programms nicht möglich**

Dieser Fehler kann nur auftreten, nachdem Sie CONT im Direktmodus zur Programmfortsetzung eingegeben haben. Dies ist der Fall, wenn Sie das Programm nach einer Fehlermeldung (Stack leer) oder Programmänderung (Variablen gelöscht) fortsetzen wollen, oder das Programm überhaupt noch nicht gestartet haben.

5 DEVICE NOT PRESENT — **(Gerät nicht am seriellen Bus)**

Dieser Fehler tritt nur bei den Befehlen OPEN, CLOSE, INPUT#, GET#, PRINT#, CMD, LOAD, SAVE und VERIFY auf. Dies ist der Fall, wenn kein Gerät am IEC-Bus angeschlossen und eingeschaltet ist.

Ist ein Gerät am IEC-Bus angeschlossen (Steckerverbindung) und eingeschaltet, so tritt dieser Fehler aber auch dann nicht auf, wenn Sie ein anderes Gerät (andere Gerätenummer) ansprechen, was entweder nicht angeschlossen oder nicht eingeschaltet ist.

20 DIVISION BY ZERO Division durch Null

Für Mathematiker klar: Ist der Divisor bei einer Division '0', so ist dies eine mathematisch unerlaubte Operation und wird auch von dem Computer nicht ausgeführt. Prüfen Sie Ihr Programm auch auf logische Fehler.

24 FILE DATA Falsche Eingabe-Variable

Diese Fehlermeldung kann nur beim Einlesen von einer Datei auftreten (INPUT# oder GET#). In diesem Falle wurde versucht eine numerische Variable (z.B. A oder A%) mit Buchstaben oder Sonderzeichen zu versehen. Prüfen Sie die Zahl Ihrer Trennzeichen in der Datei oder auch den Dateinamen, vielleicht haben Sie nur die falsche Datei angesprochen.

3 FILE OPEN Datei bereits offen

Der 'FILE OPEN'-Fehler tritt nur beim OPEN-Befehl auf, wenn eine Datei geöffnet werden soll, die bereits angesprochen wurde. Haben Sie eine logische Dateinummer ein zweitesmal verwendet?

4 FILE NOT FOUND Datei nicht gefunden

Dieser Fehler bezieht sich nur auf Programmdateien und kann somit nur bei den Befehlen LOAD und VERIFY auftreten, allerdings sowohl bei der Floppy als auch bei dem Kassettenrekorder. Beim Kassettenrekorder kann dieser Befehl auch noch beim Öffnen einer Datei zum Lesen (OPEN) gegeben sein.

2 FILE NOT OPEN Datei nicht offen

Dieser Fehler tritt bei INPUT#-, GET#- oder PRINT#-Befehlen sowie CMD auf, wenn eine Datei angesprochen wird (logische Dateinummer) die vorher nicht geöffnet oder bereits

wieder geschlossen wurde. Vielleicht haben Sie sich aber auch nur bei der Eingabe der logischen Dateinummer vertippt.

25 FORMULA TOO COMPLEX Zu komplizierter Stringausdruck

Ein FOMULA TOO KOMPLEX ERROR tritt auf, wenn Sie mit zu vielen Stringbearbeitungsbefehlen (MID$, RIGHT$, LEFT$ und STR$) arbeiten. Hier hilft das Aufteilen in mehrere Anweisungen. Läßt sich Ihr Programm anschließend nicht mehr auslisten, so hilft manchmal die Eingabe von POKE 24,0.

21 ILLEGAL DIRECT Nicht erlaubtes Direktkommando

Dieser Fehler wird durch die Eingabe von GET, GET#, INPUT, INPUT# und DEF FN() ausgelöst, wenn Sie diese im Direktmodus eingeben. Da die Direktbefehle den Basic-Eingabepuffer benützen, würde er bei den obengenannten Befehlen gleichzeitig zwei Funktionen besitzen, was nicht erlaubt ist. Für INPUT bzw. INPUT# kann man dies durch LEST bzw. LEST# (Kapitel 2.2.9.4) umgehen.

9 ILLEGAL DEVICE Falsche Gerätnummer

Eine falsche Gerätenummer kann z.B. 3 bei LOAD/SAVE sein.

14 ILLEGAL QUANTITY Unerlaubter Wert

Der Fehler ILLEGAL QUANTITY tritt auf, wenn Sie die Definitionsbereiche der einzelnen Parameter oder Variablentypen überschreiten.

Im **Bytebereich** liegen die zugelassenen Werte zwichen 0 und 255 und so führen u.a. folgende Befehle mit größeren oder kleineren Daten auf ein ILLEGAL QUANTITY ERROR:

ON, WAIT, POKE, OPEN, CAP, SPC, ASC, CHR$, LEFT$, MID$, RIGHT$.

Der **Integerbereich** geht von -32767 bis 32768. Dieser Bereich kann durch Integer-Variablen nach mathematischen Operationen aber auch durch die logischen Operatoren

NOT, AND und OR

ausgelöst werden.

Im **Adreßbereich** (16-Bit Werte/0 bis 65535) können die Befehle

WAIT, POKE, PEEK und SYS

einen Fehler auslösen und bei den mathematischen Operationen

SQR, LOG und ATN sowie (Pfeil nach oben/Potenzieren)

sind die üblichen **mathematischen Einschränkungen** zu beachten.

29 LOAD **eingelesenes Programm nicht in Ordnung**

Bei Verwendung von Kassetten wird mit dem SAVE-Kommando das Programm zweimal hintereinander auf Kassette geschrieben und bei dem Ladevorgang das erste Programm in den Hauptspeicher geladen und anschließend mit dem zweiten Programm verifiziert. Treten hier Unstimmigkeiten auf (es kann durchaus die zweite Programmspeicherung defekt sein), so wird ein LOAD ERROR ausgegeben. Wie dieser umgangen werden kann, ist in Kapitel 4 beschrieben.

8 MISSING FILE NAME **Dateiname fehlt**

Dieser Fehler tritt auf, wenn Sie eine Datei ansprechen und als Dateiname eine leere Zeichenreihe angeben (z.B. bei LOAD/SAVE)

10 NEXT WITHOUT FOR NEXT ohne FOR

Der Fehler mit der Nummer 10 tritt auf, wenn das Programm auf ein NEXT stößt, aber keine FOR...NEXT-Schleife begonnen wurde. Prüfen Sie Ihr Programm auf logische Fehler.

6 NOT INPUT FILE Keine Eingabedatei

Dieser Fehler tritt wiederum nur bei Kassettenrekordern auf. Hier wurde zum Lesen die falsche Sekundäradresse eingegeben (muß zum Lesen '0' sein).

7 NOT OUTPUT FILE Keine Ausgabedatei

Analog zu dem eben genannten Fehler ist hier die Sekundäradresse wahrscheinlich 0, während Sie die Datei zum Schreiben öffnen wollten. Die Sekundäradresse muß zum Schreiben '1' oder '2' sein.

13 OUT OF DATA Zuwenig DATA-Zeilen

Dieser Fehler wird durch den READ-Befehl ausgelöst, wenn in den DATA-Zeilen zu wenig Daten zum Einlesen zur Verfügung stehen. Achtung: Stehen zu viele Daten in DATA-Zeilen, kann der Rechner natürlich nicht wissen, ob diese später noch verwendet werden. Ein Nichtauftreten dieses Fehlers ist also kein Kriterium für die korrekte Anzahl von Daten. Prüfen Sie auch sorgfältig die Reihenfolge der Daten, um logische Fehler zu vermeiden.

16 OUT OF MEMORY Kein freier Speicher mehr, Stacküberlauf

Dieser Fehler tritt dann auf, wenn Sie mit Ihrem Programm und den Daten den gültigen Arbeitsbereich für ein Basic-Programm ausgefüllt haben. Benutzen Sie die im Kapitel 1 angegebenen Hinweise zum Speicherplatz sparen.

Haben Sie noch genug freien Speicher, so liegt der Fehler in einem Stacküberlauf, d.h. der Stack für die Rückkehradressen von GOSUB...RETURN und FOR...NEXT kann keine weiteren Daten mehr aufnehmen.

Prüfen Sie in diesem Fall, ob eine andere Struktur der Unterprogramme möglich ist, wobei es nur auf die Tiefe von ineinandergeschachtelten Unterprogrammen ankommt. Unterprogramme, die nur einmal aufgerufen werden, können - wenn Sie die Zeilennummern nicht ändern wollen mit GOTO'Anfang' ... GOTO'weiter' konstruiert werden. Haben Sie mehrere Einsprungstellen, muß eine etwas aufwendigere Konstruktion herhalten. Versehen Sie das Unterprogramm mit einem zusätzlichen Eingabeparameter und verzweigen Sie aufgrund dieses Parameters mittels des Befehls ON A GOTO X,Y,Z.

Beispiel:

```
100 GOSUB 10000
110 ... andere Anweisungen

350 GOSUB 10000
360 ... andere Anweisungen

570 GOSUB 10000
580 ... andere Anweisungen

10000 Anweisungen ...
10999 RETURN
```

wird dann zu:

```
100 HI=1 : GOTO 10000
110 ... andere Anweisungen

350 HI=2 : GOTO 10000
360 ... andere Anweisungen

570 HI=3 : GOTO 10000
580 ... andere Anweisungen

10000 Anweisungen ...
10999 ON HI GOTO 110,360,580
```

FOR...NEXT-Schleifen können Sie durch einen Zähler mit folgender Programmkonstruktion übergehen:

```
I=I+1
IF I LT 'Endekriterium' GOTO '1.Anweisung der Schleife'
```

Aber es brauchen nicht alle Schleifen und Unterprogramme umgewandelt zu werden; meist reichen ein oder zwei Schleifen aus. Auch die Reihenfolge ist nicht von Bedeutung: Nehmen sie die Schleife / das Unterprogramm, was Sie am leichtesten ändern können.

15 OVERFLOW Überlauf des Zahlenbereiches

Diese Fehlermeldung erscheint ähnlich dem ILLEGAL QUANTITY ERROR, wenn Sie bei Fließkommazahlen den gültigen Zahlenbereich (-1E38 bis +1E38) über- oder unterschreiten.

12 RETURN WITHOUT GOSUB/RETURN ohne GOSUB

Fehlernummer 12 kann nur durch den Befehl 'RETURN' ausgelöst werden. Ähnlich dem NEXT WITHOUT FOR ERROR wurde hier das Ende eines Unterprogrammes erreicht, ohne daß ein Unterprogramm aufgerufen wurde. Neben dem Auftreten in der üblichen Testphase tritt dieser Fehler häufig auf, wenn Sie das Programm während des Laufes abgebrochen haben und nach Änderung einer Zeile mit CONT weitermachen wollen, während das Programm zum Zeitpunkt des Abbruches in einem Unterprogramm war.

19 REDIM'D ARRAY Feld bereits dimensioniert

Der REDIM'D ARRAY ERROR tritt auf, wenn Sie eine Variable ein zweitesmal dimensionieren wollen, da nur einmal dimensioniert werden darf. Der Fehler tritt häufig auf, wenn die Dimensionierung innerhalb einer Schleife steht, aber auch bei einer Dimensionierung, wenn vorher einem Element dieses Feldes im Bereich 0 bis 10 bereits ein Wert zugeordnet wurde und das Feld damit angesprochen wurde.

23 STRING TOO LONG Zeichenreihe zu lang

Ein STRING TOO LONG ERROR tritt auf, wenn Sie mittels einer Zeichenreihenoperation ('+') einer Zeichenreihe mehr als 255 Zeichen zuweisen wollen. Bei einem Input# von

einer Datei wird dieser Fehler gemeldet, wenn Sie mehr als 80 Zeichen ohne Carriage Return (CHR$(13)) einlesen wollen. Bei Dateinamen liegt die Grenze bei 16 Zeichen.

11 SYNTAX Fehlerhafter Befehl

Ein Syntaxfehler tritt meistens auf, wenn Sie sich verschrieben haben, d.h. wenn Sie einen Befehl verwendet haben, der vom Interpreter nicht verstanden wird. Achten Sie auch auf gleiche Zahl von öffnenden wie schließenden Klammern, was besonders bei umfangreichen Zeichenreihenoperationen oder Zahl/String-Konvertierungen vorkommen kann. Aber auch zuviele oder zuwenige Trennzeichen, wie Kommas oder Doppelpunkte, führen zu einem Syntax-Fehler. Auch Zeilennummern, die größer als 63999 sind, lehnt der Interpreter mit einem SYNTAX ERROR ab. In einigen Fällen tritt auch der SYNTAX ERROR an Stelle eines ILLEGAL QUANTITY ERRORS.

1 TOO MANY FILES Zuviele Dateien geöffnet

Dieser Fehler kann nur bei einem OPEN-Befehl auftreten, da nie mehr als 10 Dateien geöffnet sein können. Bei einem weiteren OPEN-Befehl tritt dieser Fehler auf, wobei auch wieder der Merker für die Anzahl der Dateien zurückgesetzt wird (siehe FILE OPEN ERROR).

22 TYPE MISMATCH Falsche Variablentypen (String, Zahlen)

Wenn Sie Zeichenreihenkonstanten und Zeichenreihenvariablen auf der einen Seite und Zahlvariablen und Zahlkonstanten auf der anderen Seite gegenseitig zuordnen, so tritt ein TYPE MISMATCH ERROR auf. Hier schaffen die Basic-Befehle VAL() und STR$() Abhilfe. Vielleicht haben Sie auch nur ein '$' zuviel oder zuwenig an einer Variablen.

17 UNDEF'D STATEMENT Zeilennummer nicht vorhanden

Bei RUN (Zeilennummer), ON (Variable), GOSUB (Zeilennummer), GOTO (Zeilennummer), GOSUB (Zeilennummer) oder THEN

(Zeilennummer) tritt dieser Fehler auf, wenn Sie eine Zeilennummer angeben, die bisher noch nicht spezifiziert wurde. Dies ist häufig beim Löschen von REM-Zeilen der Fall, wenn diese angesprochen werden. Abhilfe schafft eine leere Zeile mit REM, sofern keine logischen Fehler vorliegen.

27 UNDEF'D FUNCTION Funktion nicht definiert

Hier kann nur die Anweisung FN() den Fehler auslösen, wenn die angegebene Funktion vorher nicht definiert wurde. Achtung: Ein Definieren irgendwo innerhalb des Programmes ist nicht sinnvoll, da das Programm beim Auftreten von FN() bereits die Definition mit DEF FN() passiert haben muß.

28 VERIFY Fehler bei Programmüberprüfung

Sowohl beim Kassettenrekorder als auch bei der Floppy hat man die Möglichkeit, das soeben gespeicherte Programm mit dem VERIFY-Befehl zu verifizieren. Wenn es auch bei der Floppy seltener gebraucht wird, so ist es beim Kassettenrekorder ein sehr hilfreiches Mittel. Tritt nach dem Verifizieren ein VERIFY ERROR auf, so sollte man das Programm nochmals speichern.

Hinweise ohne Programmabbruch

REDO FROM START Nochmal eingeben

Wurde bei einem INPUT-Befehl eine Zahl angefordert und Sie geben ein nichtnumerisches Zeichen ein oder sind Anführungszeichen gesetzt, so wird der INPUT-Befehl vom Basic-Interpreter nochmals ausgeführt, D.h. Sie können Ihre Eingabe wiederholen.

EXTRA IGNORED **Ein Teil der Daten wird ignoriert**

Auch dieser Fehler tritt bei INPUT auf, geht aber nach der Eingabe nicht auf diesen Befehl zurück. Die Meldung entsteht, wenn Sie in einer angeforderten Zeichenreihe ein Komma oder einen Doppelpunkt eingeben, ohne daß weitere Variablen die hinteren Stringteile aufnehmen können.

Die Probleme mit INPUT können durch die Routine 'Textzeile lesen' aus Kapitel 2.2.9 oder dem Input-Ersatz aus Kapitel 4.1 umgangen werden.

Raum für Notizen:

5.11.2 Fehlermeldungen von der Floppy

Allgemeine Übersicht (Einlesen mit INPUT#15,A,B$,C,D)

A	B$	C	D	
	Keine Fehler			
00	OK	0	0	Alles in Ordnung
01	FILES SCRATCHED	Z	0	C enthält Anzahl gelöschter Dateien
73	CBM DOS V2.6 V 170	0	0	Meldung der DOS-Version
	Lesefehler			
20	READ ERROR	T	S	Kein Block 'header'
21	READ ERROR	T	S	Kein Synchronisationszeichen
22	READ ERROR	T	S	Datenblock nicht vorhanden
23	READ ERROR	T	S	Prüfsummenfehler im Datenblock
24	READ ERROR	T	S	Byte wurde falsch dekodiert
27	READ ERROR	T	S	Prüfsummenfehler im 'header'
71	DIR ERROR	0	0	Directory Fehler
	Schreibfehler			
25	WRITE ERROR	T	S	Schreib-/Prüffehler
28	WRITE ERROR	T	S	Datenblock zu lang
	Syntax Fehler			
30	SYNTAX ERROR	0	0	Allgemeine Syntax
31	SYNTAX ERROR	0	0	Ungültiges Kommando
32	SYNTAX ERROR	0	0	Zu langes Kommando
33	SYNTAX ERROR	0	0	Ungültiger Dateiname
34	SYNTAX ERROR	0	0	Dateiname fehlt
39	SYNTAX ERROR	0	0	Ungültiges Kommando

Bedienungsfehler

72	DISK FULL	T	S	Diskette voll
29	DISK ID MISMATCH	T	S	Disk ID stimmt nicht überein
74	DRIVE NOT READY	O	O	Laufwerk enthält keine Diskette (oder termische Überlastung)
63	FILE EXISTS	O	O	Datei existiert bereits
62	FILE NOT FOUND	O	O	Datei wurde nicht gefunden
61	FILE NOT OPEN	O	O	Datei ist nicht 'offen'
52	FILE TOO LARGE	T	S	Recordnummer ist zu hoch
64	FILE TYPE MISMATCH	O	O	Dateitypendurcheinander
67	ILLEGAL SYSTEM T OR S	T	S	unerlaubte Spur oder Sektor
66	ILLEGAL TRACK AND SEKTOR	T	S	unerlaubte Spur oder Sektor
65	NO BLOCK	T	S	Block nicht frei
70	NO CHANNEL	O	O	Kein Kanal frei
51	OVERFLOW IN RECORD	T	S	Record ist zu kurz für die Daten
50	RECORD NOT PRESENT	T	S	Record ist nicht vorhanden
60	WRITE FILE OPEN	O	O	Datei bereits zum Schreiben geöffnet
26	WRITE PROTECT ON	T	S	Schreibschutz 'geklebt'

Ein großer Teil der Fehler (Lesefehler, Schreibfehler und Syntaxfehler) dienen der korrekten Speicherung der Daten auf Diskette und somit dem Anwender. In Kapitel 2 haben wir ein kurzes Programm aufgezeigt, mit dem die Fehlermeldungen von der Floppy abgefragt werden können.

Achten Sie darauf, an den richtigen Stellen die Fehler von der Floppy abzufragen, da durch ein neues Floppy-Kommando in der Regel diese Fehlermeldung wieder überschrieben wird. Lesen Sie also die Fehlermeldung unmittelbar nach jedem Floppykommando (was sehr zeitaufwendig ist, weil der Rechner auf die Floppy warten muß) oder unmittelbar vor jedem Floppykommando (was einer genauen Programmkonzeption bedarf).

Auch das Blinken der roten Kontrolleuchte weist auf einen Floppyfehler hin.

Abhilfe bei Lese- und Schreibfehlern

Bei Lesefehlern sollte man nicht gleich die Diskette wegwerfen, sondern erneut - auch mehrmals - einen Lesevorgang versuchen. Manchmal hilft auch das neue Zentrieren der Diskette innerhalb der Schutzhülle (von Hand). Falls Sie trotz Lesefehler noch lesen können, sollten Sie jedoch diese Diskette mit äußerster Vorsicht handhaben und zunächst ein Backup machen.

Bei Schreibfehlern ist die Sache nicht ganz so gravierend. Auf jeden Fall sollten Sie eine neue Diskette heranziehen und die anderen Daten von der Diskette gegebenenfalls kopieren.

Syntaxfehler

30/39 allgemeiner Syntaxfehler / falscher Befehl

Diesen Fehler erhalten Sie, wenn Sie die Zeichenreihe, die ein Kommando an die Floppy übergibt, fehlerhaft aufbauen. Dies kann mehrere Ursachen haben:

- ':' ohne vorhergehende 'O', '1' oder '$'
- zwischen Ziel und Quelldatei ist kein '='
- zwischen mehreren Quellennamen kein ','
- führende Leerzeichen im Kommando

31 Ungültiges Kommando

Ein ungültiges Kommando hat meist einen Eingabefehler als Ursache.

32 Zu langes Kommando

Es dürfen nicht mehr als 58 Zeichen in den Kommandostring eingetragen werden.

33 Ungültiger Dateiname

Beachten Sie, daß der Dateiname nicht länger als 16 Zeichen lang sein darf und nicht alle Sonderzeichen zulässig sind. Auch kann das '*' im OPEN- oder SAVE-Befehl an falscher Stelle verwendet worden sein. Beim Lesen und Schreiben wird manchmal ab dem 17. Zeichen abgeschnitten.

Achten Sie auch auf die verschiedene Zahl von Namen bei den Kommandos N (mindestens 1), S (mindestens 1), R (genau 2), und C (mindestens 2).

73 CBM DOS V2.6 V 170 Meldung der DOS-Version

Diese Meldung finden Sie direkt nach dem Einschalten der Floppy im Fehlerkanal. Sie erscheint auch, wenn Sie auf eine Diskette schreiben wollen, die mit einem anderen Commodore Rechner formatiert wurde. Näheres dazu siehe Floppy-Handbuch auf Seite 39.

71 DIR ERROR Block availible Map defekt

Die BAM stellt das Inhaltsverzeichnis der Diskette für die einzelnen Blöcke dar. In ihr ist festgehalten, welche Blöcke bereits durch Dateien genutzt sind und welche noch für den Benutzer verwendbar sind. Durch ein Reinitialisieren kann manchmal der Fehler behoben werden, was allerdings zur Folge hat, daß die von offenen sequentiellen Dateien genutzten Blöcke der BAM wieder zugeordnet werden und somit die Daten verloren sind. Läßt sich die BAM nicht mehr einladen, so ist die Diskette wertlos, da sie nicht mehr gelesen werden kann.

72 DISK FULL Diskette voll

Diese Meldung erscheint nicht nur, wenn alle 664 Blöcke der Diskette belegt sind, sondern auch, wenn die maximale Anzahl von 144 Einträgen in der Datei erreicht ist. Löschen Sie über den Kommandokanal Dateien oder formatieren Sie mit Hilfe des Kommandokanals eine neue Diskette, bevor das Programm verloren geht.

29 DISK ID MISMATCH — **Identifikationen der Disketten stimmen nicht überein**

Beim Initialisieren erhält die Diskette eine zweistellige Identifikationsnummer (ID). Legen Sie eine neue Diskette ein, so muß diese zunächst initialisiert werden, was im Normalfall die Floppy für Sie erledigt. Treten Probleme auf, so können Sie die Initialisierung auch von Hand durchführen mit: OPEN 1,8,15,"I" : CLOSE1

74 DRIVE NOT READY — **keine Diskette im Laufwerk / thermische Überlastung**

Sinn dieses Befehls ist es, den Benutzer darauf hinzuweisen, daß sich keine Diskette im Laufwerk befindet. Noch wichtiger ist jedoch, die Meldung bei Auftreten einer thermischen Überlastung, wenn Sie die Floppy über längere Zeit in Gebrauch hatten, ohne diese abzuschalten. In den meisten Fällen ist dann zwar noch ein Lesen von Daten möglich, jedoch kein Speichern mehr, wodurch sich natürlich die Informationsdichte auf Ihrer Diskette langsam verringert, wenn Sie in der Testphase immer wieder neue Versionen 'speichern'. Bei längerem Gebrauch sollten Sie auf jeden Fall die Kontrollampe Ihrer Floppy im Auge behalten und entsprechend reagieren. Ein Ventilator schafft hier eine kleine Abhilfe.

63 FILE EXISTS — **Datei existiert bereits**

Wollen Sie eine Programm- oder Datendatei überschreiben, so erhalten Sie die Meldung FILE EXISTS. Wollen Sie eine andere Programmversion unter dem gleichen Namen speichern, so können Sie einerseits die Datei mit dem Kommando 'S:' über den Kommandokanal löschen und dann die Datei erneut abspeichern (sichere Methode) oder beim SAVE-Befehl ein 'Klammeraffe:' vor den Dateinamen setzen (schnelle Methode).

62 FILE NOT FOUND **Datei nicht gefunden**

Dies tritt bei lesendem Zugriff auf Dateien auf. Datei angelegt? Schreibfehler? Siehe auch unter Basic-Fehlermeldungen.

61 FILE NOT OPEN **Datei nicht offen**

Trotz gleicher Wirkung unterscheidet sich der Floppy-Fehler vom gleichnamigen Fehler im Rechner. Im vorliegenden Fall ist die logische Datei im Rechner zwar noch offen, aber auf der Floppy schon geschlossen. Dies passiert z.B. beim Schließen des Kommando-Kanales.

52 FILE TOO LARGE **Zu hohe Datensatznummer**

Nur bei Direktzugriffsdateien tritt dieser Fehler auf. Einerseits kann die Datensatznummer zu hoch sein, andererseits kann aber auch die Diskette voll sein, so daß nach Ausführen des Schreibens dieses Datensatzes ein 'DISK FULL' gemeldet würde.

64 FILE TYPE MISMATCH **Falscher Dateityp**

Hier wurde versucht, eine Direktzugriffsdatei als sequentielle Datei anzusprechen oder umgekehrt, oder eine von beiden Möglichkeiten in Verbindung mit einer Programmdatei.

67 ILLEGAL SYSTEM T OR S **Unerlaubte Spur oder Sektor**

Bei einem Direktzugriffsbefehl auf Blöcke wurde eine nicht zulässige Spur oder ein nicht zulässiger Sektor innerhalb der Spur angegeben.

66 ILLEGAL TRACK AND SECTOR Unerlaubte Spur oder Sektor

Siehe letzte Fehlermeldung

65 NO BLOCK Block besetzt

Dieser Fehler tritt nur in Verbindung mit dem Direktzugriffsbefehl 'B-A' auf, wenn der gewünschte Block nicht frei ist. Die beiden letzten Daten der Fehlermeldung geben den nächsten freien Block mit höherer Nummer an.

70 NO CHANNEL Kein Kanal frei

Wenn mehr Datenkanäle geöffnet werden sollen als zur Verfügung stehen, oder ein direkt angesprochener Kanal belegt ist, tritt dieser Fehler auf.

51 OVERFLOW IN RECORD Datensatz ist zu kurz

Die Daten, die in den Datensatz (bei Direktzugriffsdaten) geschrieben werden sollen, überschreiten die zu Beginn festgelegte Datensatzlänge. Trennzeichen (sogenannte Delimiter) sind dabei mitzuzählen.

50 RECORD NOT PRESENT Datensatz nicht vorhanden

Entweder haben Sie einen Datensatz, der nicht existiert, direkt angesprochen, oder bei sequentiellem Durchgang durch eine Direktzugriffsdatei den letzten Datensatz bereits gelesen und das Dateiende erreicht. Dieser Fehler taucht auch auf, wenn Sie eine relative Datei mit falscher Satzlänge öffnen wollen.

60 WRITE FILE OPEN **Datei bereits zum Schreiben geöffnet**

Wird dieser Fehler ausgegeben, so versuchen Sie eine Datei zum Schreiben zu öffnen, die nicht geschlossen wurde. Obwohl auf eine Datei durch mehrere logische Dateinummern zum Lesen zugegriffen werden kann, darf auf eine Datei, auf die schreibend zugegriffen werden soll, nur diese Datei mit der einen logischen Dateinummer zugreifen. Auch ein lesender Zugriff ist ausgeschlossen.

26 WRITE PROTECT ON **Schreibschutz angebracht**

Die Schreibschutzkerbe auf der linken Seite Ihrer Diskette (beim Einschieben gesehen) ist zugeklebt, was einen schreibenden Zugriff auf die Daten verhindert.

Anhang

Anhang 1 : Literaturverzeichnis

1 # Knuth, Art of Computer Programming, Part 3, Sorting & Searching

2 # Sedgewick, The Analysis of Quicksort Programms, Acta Informatika 1977, Heft 7, Seite 327-356

3 # IBM-series, Sorting & Sorting Systems, System Programming 1977

4 # Schneider, Eberl: Das Commodore 64-Buch, Band 1: Leitfaden für Einsteiger, 1983

5 # Schneider, Eberl: Das Commodore 64-Buch, Band 3: Leitfaden für Fortgeschrittene, 1984

6 # Schneider, Eberl: Das Commodore 64-Buch, Band 4: Leitfaden für Systemprogrammierer, 1984

7 # Schneider, Eberl: Das Commodore 64-Buch, Band 5: Simon's Basic, 1984

8 # Schneider, Eberl, Turba: Das Commodore 64-Buch, Band 7: Leitfaden für Profis, 1984

9 # Schneider, Basic ohne Probleme, Band 3: Programmentwicklung und Datenorganisation, 1983

10 # Vorlesungsunterlagen HSBW-M, Jahrgang 1976, Informatik

Anhang 2 : Vergleichsoperatoren

Aus drucktechnischen Gründen wurden in diesem Buch folgende Operatoren in Anlehnung an andere Programmiersprachen umbenannt:

LT	kleiner als	- **L**ess **T**hen
GT	größer als	- Greater Than
LE	kleiner gleich	- **L**ess or **E**qual
GE	größer gleich	- **G**reater or **E**qual
NE	ungleich	- Not **E**qual

**	Potenzierungsoperator 'hoch'

Anhang 3 : Basic-Loader der Erweiterungen

Damit Sie ohne weitere Hilfsmittel wie Assembler oder Monitor die Maschinenprogramme der Basic-Erweiterungen eingeben können, bilden wir hier ein Basic-Programm ab, das den Maschinencode in DATA-Zeilen gespeichert hat. Aus Speicherplatzgründen haben wir zwei Basic-Loader-Programme abgebildet, das erste für Maschinenprogramme im Bereich $C000 bis $CBFF, das zweite für den Bereich $8E00 bis $9FFF. Der Vorspann zu beiden Programmen ist fast gleich, so daß Sie die Zeilen bis 1000 zweimal verwenden können.

Wie in Kapitel 2.2 beschrieben, brauchen Sie nicht alle DATA-Zeilen einzugeben, wenn Sie nur bestimmte Erweiterungen benötigen. Aus diesem Grund ist es auch nicht sinnvoll, die Prüfsumme über alle Datas abzuprüfen. Statt dessen haben wir für jede Zeile eine Prüfsumme angegeben, so daß Sie auch sofort sehen, wo Sie sich evtl. vertippt haben.

Vom Programm aus ist die Zeilennummer der aktuellen DATA-Zeile kontrollierbar, so daß es auch möglich ist, zu prüfen, ob die Zeilennummer korrekt ist. In diesem Programm sind nur Zeilennummern zugelassen, die durch 16 teilbar sind. Außerdem kann die korrekte Anzahl von DATA's in der Zeile (16 Werte + 1 Prüfsumme) gechecked werden. Die Zeilennummer ist gleichzeitig die Adresse, in welche das erste DATA dieser Zeile gePOKEd wird.

Wenn alle DATA-Zeilen korrekt waren, wird der beschriebene Speicherbereich auf Floppy gespeichert. Zum Speichern auf Kassettenrecorder müssen Sie in Zeile 600 das ',8' durch ',1,1' ersetzen. Als Dateiname für den ersten Teil der Basic-Erweiterungen wurde "E1" gewählt, ein kurzer Name, damit er schnell getippt werden kann.

```
100 REM ***  BASIC-LOADER-PROGRAMM  ***            <199>
110 REM ***  FUER MASCHINENROUTINEN ***            <053>
120 REM ***  AUS KAPITEL 2.2        ***            <136>
130 REM                                            <017>
140 REM ***  TEIL 1: BEREICH $C000-$CBFF  ***      <251>
150 REM ***                (49152-52223) ***       <162>
160 REM                                            <047>
170 REM ***  DIE ZEILEN-NUMMERN DER DATA-ZEILEN     <136>
180 REM ***  SIND GLEICH DER ZIELADRESSE !!         <243>
190 REM ***  JE DATA-ZEILE EINE PRUEFSUMME          <068>
210 BA=49152 : REM BEREICHANFANG                   <114>
220 BE=52223 : REM BEREICHENDE                     <234>
230 DEF FN D(X)=PEEK(X)+256*PEEK(X+1)              <100>
240 GS=0     : REM GESAMTPRUEFSUMME                <255>
250 D1=BA    : REM NAECHSTE ZEILENNR.              <077>
260 ZS=0     : REM ZEILENPRUEFSUMME                <045>
270 READ A    : REM 1. ELEMENT LESEN               <128>
280 D=FN D(63): REM NUMMER DER DATA-ZEILE          <234>
290 PRINT"DATA-ZEILE";D;                           <122>
300 IF D<BA THEN PRINT"ZEILENNR. ZU KLEIN":STOP    <055>
310 IF D>BE THEN PRINT"ZEILENNR. ZU GROSS":STOP    <094>
320 IF D=16*INT(D/16)THEN 340                      <113>
330 PRINT"ZEILENNR. NICHT DURCH 16 TEILBAR !"      <052>
340 POKE D,A:ZS=ZS+A                               <206>
350 FOR I=1 TO 15                                  <022>
360 READ A                                         <049>
370 IF A>255 THEN PRINT"FALSCHER WERT":STOP        <100>
380 POKE D+I,A:ZS=ZS+A                             <233>
390 NEXT                                           <009>
400 READ A                                         <089>
410 IF FN D(63)<>D THEN PRINT"ZEILE ZU KURZ":STOP  <051>
420 PRINT"PRUEFSUMME ";                            <198>
430 IF ZS<>A THEN PRINT"FALSCH":STOP               <139>
440 IF ZS=A THEN PRINT"OK"                         <002>
450 IF D>D1 THEN PRINT"ZEILE";D1;"FEHLT":D1=D1+16:
    GOTO 450                                       <154>
460 D1=D1+16                                       <122>
470 GS=GS+ZS                                       <020>
480 IF D+16<BE THEN 260                            <058>
490 PRINT"GESAMTPRUEFSUMME";GS                     <103>
500 END                                            <117>
600 SYS 57812 "E1",8 :REM DATEINAME                <110>
610 POKE 193,BA-256*INT(BA/256)                    <014>
620 POKE 194,INT(BA/256)                           <162>
630 POKE 780,193                                   <119>
640 POKE 781,BE-256*INT(BE/256)                    <055>
650 POKE 782,INT(BE/256)                           <199>
660 SYS 65496         :REM SAVE                    <058>
670 END                                            <032>
```

```
49152 DATA165,203,205, 47,192,240, 36,201, 64,240, 32,162,  0,142,  7,212, 2148
49168 DATA142, 11,212,162,  4,142, 13,212,162,140,142,  8,212,162, 16,142, 1882
49184 DATA 12,212,162, 17,142, 11,212,202,142, 11,212,141, 47,192, 96, 64, 1875
49200 DATA165,212,208, 89,141,142,192,166,203,202,202,202,224,  4,176, 72, 2600
49216 DATA189,144,192,174,141,  2,125,148,192,141,142,192,165,204,208, 61, 2420
49232 DATA165,203,205,143,192,240, 54,165,216,208, 45,165,254, 72,165,253, 2745
49248 DATA 72,174,142,192,202,138, 10, 10, 10, 10,133,253,169,193,105,  0, 1813
49264 DATA133,254,160,  0,177,253,240,  8,153,119,  2,200,192, 15,144,244, 2294
49280 DATA132,198,104,133,253,104,133,254,165,203,141,143,192, 96,  0, 64, 2315
49296 DATA  7,  1,  3,  5,  0,  1,  8,  9, 16, 17, 24, 25,  0,  0,  0,  0, 116
49312 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
49328 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
49344 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
49360 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
49376 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
49392 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
49408 DATA 76, 73, 83, 84, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 329
49424 DATA 79,208, 52, 44, 52, 58, 67,205, 52, 58, 76,201, 13,  0,  0,  0, 1165
49440 DATA 82, 85, 78, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 258
49456 DATA 80,210, 52, 58, 67, 76, 79, 83, 69, 52, 13,  0,  0,  0,  0,  0, 839
49472 DATA 67, 79, 78, 84, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 321
49488 DATA 63, 70, 82, 69, 40, 48, 41, 43, 54, 53, 53, 51, 56, 13,  0,  0, 736
49504 DATA  8, 14,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 22
49520 DATA  8,142,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 150
49536 DATA 76, 79, 65, 68, 34, 34, 20,  0,  0,  0,  0,  0,  0,  0,  0,  0, 376
49552 DATA 80, 79, 75, 69, 53, 51, 50, 56, 48, 44,  0,  0,  0,  0,  0,  0, 605
49568 DATA 34, 44, 56, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 147
49584 DATA 80, 79, 75, 69, 53, 51, 50, 56, 49, 44,  0,  0,  0,  0,  0,  0, 606
49600 DATA 83, 65, 86, 69, 34, 34, 20,  0,  0,  0,  0,  0,  0,  0,  0,  0, 391
49616 DATA 86, 69, 82, 73, 70, 89, 34, 34, 20,  0,  0,  0,  0,  0,  0,  0, 557
49632 DATA 83, 65, 86, 69, 34, 34, 20, 64, 58,  0,  0,  0,  0,  0,  0,  0, 513
49648 DATA 86, 69, 82, 73, 70, 89, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0, 482
49664 DATA 83, 89, 83, 53, 49, 50, 48, 48, 13,  0,  0,  0,  0,  0,  0,  0, 516
49680 DATA 83, 89, 83, 52, 57, 49, 53, 50, 13,  0,  0,  0,  0,  0,  0,  0, 529
```

```
49696 DATA 68, 73, 83, 75, 13,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 312
49712 DATA 86, 73, 69, 87, 34, 36, 34, 44, 56, 13,  0,  0,  0,  0,  0,  0, 532
49728 DATA 76, 65, 68, 48, 44, 34, 69, 50, 34, 44, 56, 44, 49, 13,  0,  0, 694
49744 DATA 80, 79, 75, 69, 36, 67, 56, 48, 57, 44,  0,  0,  0,  0,  0,  0, 611
49760 DATA 70, 50, 51,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 171
49776 DATA 70, 50, 52,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 172
49792 DATA 70, 50, 53,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 173
49808 DATA 70, 50, 54,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 174
49824 DATA 70, 50, 55,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 175
49840 DATA 70, 50, 56,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 176
49856 DATA 70, 50, 57,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 177
49872 DATA 70, 51, 48,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 169
49888 DATA 70, 51, 49,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 170
49904 DATA 70, 51, 50,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 171
49920 DATA 80, 67, 80, 88, 67, 80, 89, 68, 69, 67, 68, 69, 88, 68, 69, 89, 1206
49936 DATA 69, 79, 82, 73, 78, 67, 73, 78, 88, 73, 78, 89, 74, 77, 80, 74, 1232
49952 DATA 83, 82, 76, 68, 65, 76, 68, 88, 76, 68, 89, 76, 83, 82, 78, 79, 1237
49968 DATA 80, 79, 82, 65, 80, 72, 65, 80, 72, 80, 80, 76, 65, 80, 76, 80, 1212
49984 DATA 82, 79, 76, 82, 79, 82, 82, 84, 73, 82, 84, 83, 83, 66, 67, 83, 1267
50000 DATA 69, 67, 83, 69, 68, 83, 69, 73, 83, 84, 65, 83, 84, 88, 83, 84, 1235
50016 DATA 89, 84, 65, 88, 84, 65, 89, 84, 83, 88, 84, 88, 65, 84, 88, 83, 1311
50032 DATA 84, 89, 65, 87, 79, 82, 32,253,174, 32,158,173, 32,163,182,160, 1845
50048 DATA  0,201,  3,208, 42,132,168,169, 57,133,169,160,  0,165,169,197, 1973
50064 DATA168,144, 28, 24,101,168, 74,133,170, 10,101,170,170,160,  0,189, 1810
50080 DATA200,194,209, 34,208, 10,232,200,192,  3,208,243,164,170,200, 96, 2563
50096 DATA176,  7,164,170,200,132,168,208,210,164,170,240,242,136,132,169, 2688
50112 DATA 76,139,195,169,144,133, 56,133, 52,162,  0,138,157,  0,154,232, 1940
50128 DATA208,250,141, 27,192,141, 28,192, 96,175,255,175,255,175,255,175, 2740
50144 DATA 80,  0, 80,  0, 80,  0, 80,  0,  0,  0, 80,  0, 80,  0, 80, 80, 640
50160 DATA255,175,175,175,255,175,255,175,255,175,175,175,175,175,175,255, 3200
50176 DATA  0,169,252,208,  2,169,251,141,  0,196, 32,241,174, 32,141,173, 2181
50192 DATA120,165, 20, 72,165, 21, 72,165,  1, 72, 32,247,183,160,  0,165, 1660
50208 DATA  1, 45,  0,196,133,  1,177, 20,168,104,133,  1,104,133, 21,104, 1341
50224 DATA133, 20, 32,162,179, 88, 96, 32,241,174, 32,141,173, 32,234,197, 1966
```

```
50240 DATA165, 98,133,101,165, 99,133,100,160,  0,177,100,133, 99,200,177, 2040
50256 DATA100,133, 98, 32,228,197, 96, 32,138,173, 32,247,183, 32,253,174, 2148
50272 DATA 32,138,173, 32,234,197,160,  0,165, 99,145, 20,200,165, 98,145, 2003
50288 DATA 20, 96, 32,241,174, 32,141,173, 32,234,197,165, 98,133,101,165, 2034
50304 DATA 99,133,100,160,  0,165,  1, 72,120, 37,252,133,  1,177,100,133, 1683
50320 DATA 99,200,177,100,133, 98,104,133,  1, 88, 32,228,197, 96, 32,235, 1953
50336 DATA183,120,165,  1, 72, 41,249,133,  1,160,  0,138,145, 20,104,133, 1665
50352 DATA  1, 88, 96,120,169,  0,133, 20,169,208,133, 21,165,  1, 72, 41, 1437
50368 DATA249,133,  1,160,  0,162, 16,177, 20,145, 20,200,208,249,230, 21, 1991
50384 DATA202,208,244,104,133,  1, 88, 96,  0,  0,  0,  0,165,  1,141,219, 1602
50400 DATA196,  8,120, 41,252,133,  1,160,  0,140,218,196,177,195,170,177, 2184
50416 DATA193,145,195,138,145,193,200,208,  7,230,196,230,194,238,218,196, 2926
50432 DATA173,218,196,205,217,196,144,228,204,216,196,144,223,173,219,196, 3148
50448 DATA133,  1, 40, 96,  0,208,  0,208, 32,121,  0,240, 64, 32,138,173, 1486
50464 DATA 32,247,183,165, 20,141, 20,197,165, 21,141, 21,197, 32,121,  0, 1703
50480 DATA240, 43, 32,253,174, 32,138,173, 32,247,183,165, 20,141, 22,197, 2092
50496 DATA165, 21,141, 23,197, 32,121,  0,240, 19, 32,253,174, 32,138,173, 1761
50512 DATA 32,247,183,165, 20,141,216,196,165, 21,141,217,196,173, 20,197, 2330
50528 DATA133,195,173, 21,197,133,196,173, 22,197,133,193,173, 23,197,133, 2292
50544 DATA194, 32,220,196, 96,  0,204, 32,138,173, 32,247,183,165, 20,141, 2073
50560 DATA117,197,165, 21,141,118,197, 32,253,174, 32,212,225,169,  0,174, 2227
50576 DATA117,197,172,118,197, 32,213,255,176,  8, 32,183,255, 41,191,208, 2395
50592 DATA  1, 96,162, 29, 76, 55,164, 32,138,173, 32,247,183,165, 20,133, 1706
50608 DATA193,165, 21,133,194, 32,253,174, 32,138,173, 32,247,183,165, 20, 2155
50624 DATA141,117,197,165, 21,141,118,197, 32,253,174, 32,212,225,165,  1, 2191
50640 DATA 72, 41,254,133,  1,169,193,174,117,197,172,118,197, 32,216,255, 2341
50656 DATA104,133,  1, 96,162,144, 56, 76, 73,188,165, 20, 72,165, 21, 72, 1548
50672 DATA 32,247,183,133, 98,132, 99,104,133, 21,104,133, 20, 96, 32,170, 1737
50688 DATA177,133, 98,132, 99, 96,132, 97,160,  0,132, 98,132, 99,196, 97, 1878
50704 DATA240, 42,177, 34, 56,233, 48,201, 10,144, 10,233,  7,201, 10,144, 1790
50720 DATA 27,201, 16,176, 23,  6, 99, 38, 98,  6, 99, 38, 98,  6, 99, 38, 1068
50736 DATA 98,  6, 99, 38, 98,  5, 99,133, 99,200,208,210, 96,132, 97,160, 1778
50752 DATA  0,132, 98,132, 99,196, 97,240, 19,177, 34,201, 48,144, 13,201, 1831
50768 DATA 49,240,  2,176,  7, 38, 99, 38, 98,200,208,233, 96,169,  0,153, 1806
```

```
50784 DATA  0,  1,136,165, 99, 41, 15,201, 10,144,  2,105,  6,105, 48,153, 1231
50800 DATA  0,  1, 70, 98,102, 99, 70, 98,102, 99, 70, 98,102, 99, 70, 98, 1276
50816 DATA102, 99,136, 16,222, 96,169,  0,153,  0,  1,136, 70, 98,102, 99, 1499
50832 DATA169,  0,105, 48,153,  0,  1,136, 16,242, 96, 32,115,  0,165,122, 1400
50848 DATA133, 34,165,123,133, 35,160,  4, 32,  6,198, 32,251,168,160,  0, 1634
50864 DATA132, 13, 32,228,197, 96, 32,115,  0,165,122,133, 34,165,123,133, 1720
50880 DATA 35,160, 16, 32, 61,198, 32,251,168,160,  0,132, 13, 32,228,197, 1715
50896 DATA 96, 32,115,  0, 32,241,174, 32,130,183,133, 97,160,  0,177, 34, 1636
50912 DATA201, 36,240, 19,201, 37,240, 24,165, 97, 76,176,183,164, 97,136, 2092
50928 DATA230, 34,208,  2,230, 35, 96, 32,237,198, 32,  6,198, 76,228,197, 2039
50944 DATA 32,237,198, 32, 61,198, 76,228,197, 32,241,174, 32,141,173, 32, 2084
50960 DATA234,197, 96, 32,  9,199,160, 36,132,255,160,  4, 32, 93,198,169, 2006
50976 DATA255,160,  0, 76,135,180, 32,  9,199,160, 37,132,255,160, 16, 32, 1838
50992 DATA134,198,169,255,160,  0, 76,135,180, 32,250,174, 32,158,173, 32, 2158
51008 DATA163,182,133,169,165, 34,133,167,165, 35,133,168, 32,253,174, 32, 2138
51024 DATA158,173, 32,163,182,133,170, 32,247,174,162,  1, 56,165,170,229, 2247
51040 DATA169,144, 26,105,  0,133, 97,164,169,136, 48, 19,177,167,209, 34, 1797
51056 DATA240,247,230, 34,208,  2,230, 35,232,198, 97,208,234,162,  0,138, 2495
51072 DATA168, 32,162,179, 96, 96,135, 72,  0,  0,  0,169,122,141,  4,221, 1597
51088 DATA169, 38,141,  5,221, 32,138,173,169,134,160,199, 32, 40,186, 32, 1869
51104 DATA247,183,165, 20,141,  6,221,165, 21,141,  7,221,173, 14,221, 41, 1987
51120 DATA209,  9, 17,141, 14,221,173, 15,221, 41,209,  9, 89,141, 15,221, 1745
51136 DATA173, 13,221, 41,  2,208,  5, 32,225,255,208,244, 96,  0,  0,  0, 1723
51152 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
51168 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
51184 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
51200 DATA 76, 33,200, 76, 91,200, 76,184,202,  0, 12,144,  0,192, 48,192, 1726
51216 DATA 32,200, 32,200, 32,200, 32,200, 32,200, 32,200,  0, 36, 10, 73, 1511
51232 DATA 96,169,128,141,  8,  3,169,200,141,  9,  3,169,252,141, 10,  3, 1642
51248 DATA169,201,141, 11,  3,169,142,133, 56,133, 52,120,169,179,141, 20, 1839
51264 DATA  3,169,202,141, 21,  3, 88,169, 79,160,200, 32, 30,171, 96, 80, 1644
51280 DATA 82, 79, 70, 73, 32, 86, 49, 46, 48, 13,  0,169,228,141,  8,  3, 1127
51296 DATA169,167,141,  9,  3,169,134,141, 10,  3,169,174,141, 11,  3,120, 1564
51312 DATA169, 49,141, 20,  3,169,234,141, 21,  3, 88,169,160,133, 56, 96, 1652
```

```
51328 DATA 32,115,  0,144, 30,201, 96,176, 26,201, 65,144, 21,141, 31,200, 1623
51344 DATA162,  0,142, 29,200,160,  0,238, 29,200,189, 36,201,208,  7,173, 1974
51360 DATA 31,200, 56, 76,231,167,209,122,208, 40,200,232,189, 36,201,208, 2406
51376 DATA245, 24,152,101,122,133,122,144,  2,230,123,173, 29,200, 10,170, 1980
51392 DATA189,220,200,141,205,200,189,221,200,141,206,200, 32,  0,158, 76, 2578
51408 DATA174,167,232,189, 36,201,208,250,232, 76,149,200,231,167, 91,200, 2803
51424 DATA 87,196,158,196,179,196, 24,197,119,197,167,197,139,199,  0,158, 2409
51440 DATA  3,158,  6,158,  9,158,  0,203,  0,142,  0,148,  3,148,  6,148, 1290
51456 DATA  9,148, 12,148, 15,148, 18,148, 21,148, 24,148, 27,148, 30,148, 1340
51472 DATA 33,148, 36,148, 39,148, 42,148, 45,148, 48,148,  6,144,  3,144, 1428
51488 DATA  0,144,  9,144, 65, 85, 83,  0, 68, 79, 75, 69,  0, 82, 79, 75, 1057
51504 DATA 69,  0, 67, 72, 65, 82, 75, 79, 80,  0, 83, 87, 65, 80,  0, 76, 980
51520 DATA 65, 68,  0, 83, 80, 69, 73,  0, 80, 65, 85, 83, 69,  0, 68, 73, 961
51536 DATA 83, 75,  0, 76, 69, 83, 84,  0, 76, 69, 83, 80,  0, 86, 73, 69, 1006
51552 DATA 87,  0, 83, 69, 84, 84, 73, 77, 69,  0, 84, 69, 83, 84,  0, 71, 1017
51568 DATA 82, 69, 73, 78,  0, 71, 82, 65, 85, 83,  0, 71, 82, 76, 79, 69, 1065
51584 DATA  0, 70, 65, 82, 66, 69,  0, 71, 82, 65, 70, 69, 73, 78,  0, 80, 940
51600 DATA 85, 78, 75, 84,  0, 67, 72, 65, 82,  0, 76, 73, 78, 73, 69,  0, 977
51616 DATA 82, 69, 67, 72, 84,  0, 66, 76, 79, 67, 75,  0, 75, 82, 69, 73, 1036
51632 DATA 83,  0, 82, 65, 68, 73, 85, 83,  0, 90, 69, 73, 67, 72, 69, 78, 1057
51648 DATA  0, 83, 67, 82, 79, 76, 76,  0, 71, 82, 83, 80, 69, 73,  0, 71, 992
51664 DATA 82, 76, 65, 68,  0, 71, 82, 77, 73, 83, 67, 72,  0, 70, 65, 72, 1023
51680 DATA 82, 84, 65, 85, 83,  0, 71, 82, 69, 78, 90, 69, 78,  0, 70, 65, 1071
51696 DATA 72, 82, 84,  0, 83, 80, 82, 73, 84, 69,  0,  0,169,  0,133, 13, 1024
51712 DATA 32,115,  0,144, 42,201,197,240, 92,201, 96,176, 34,201, 36,240, 2047
51728 DATA 87,201, 37,240, 86,201, 65,144, 21,141, 31,200,162,  0,142, 30, 1788
51744 DATA200,160,  0,238, 30,200,189,130,202,208,  7,173, 31,200, 56, 76, 2100
51760 DATA141,174,209,122,208, 37,200,232,189,130,202,208,245, 24,152,101, 2574
51776 DATA122,133,122,144,  2,230,123,173, 30,200, 10,170,189,110,202,141, 2101
51792 DATA 89,202,189,111,202,141, 90,202, 76,  0,  0,232,189,130,202,208, 2263
51808 DATA250,232, 76, 33,202, 76,209,198, 76,155,198, 76,182,198,141,174, 2476
51824 DATA 19,199, 38,199, 55,196,  1,196,114,196, 57,199,  5,196,139,203, 2012
51840 DATA  0,142, 72, 69, 88, 36,  0, 66, 73, 78, 36,  0, 68, 69, 69, 75, 941
51856 DATA  0, 82, 69, 69, 75,  0, 68, 82, 69, 69, 75,  0, 73, 78, 68, 69, 946
```

```
51872 DATA 88,  0, 67, 72, 69, 69, 75,  0, 84, 73, 77, 69,  0, 84, 69, 83, 979
51888 DATA 84,  0,  0,169,  1,141, 28,200,173, 28,200, 45,  9,200,240,  3, 1521
51904 DATA 32,203,202, 14, 28,200,144,240, 76, 49,234,162,254,232,232, 74, 2376
51920 DATA144,251,189, 11,200,141, 27,200,189, 10,200,141, 26,200,108, 26, 2063
51936 DATA200,  0,  0,  0, 80,  0,  0, 64,  0,  0, 80,  0, 80,  0,  0, 80, 584
51952 DATA255,175,255,175,255,175,255,175,175,255,175,175,175,255,175,255, 3360
51968 DATA 32,158,183,202,134,171, 32,253,174, 32,158,173, 32,163,182,201, 2280
51984 DATA 10,240,  3, 76, 72,178,165,171, 74, 72,169,  0,105,220,133,168, 1856
52000 DATA162,  0,134,167,160, 14,177,167,  9,128,145,167,200,104, 74,  8, 1816
52016 DATA177,167, 42, 40,106,145,167, 32,101,203,248,201, 18,240,  6,144, 2037
52032 DATA  4,233, 18,  9,128,216,160, 11,145,167, 32,101,203,136,145,167, 1875
52048 DATA 32,101,203,136,145,167,161, 34, 41, 15,136,145,167, 96,230, 34, 1843
52064 DATA208,  2,230, 35, 96,161, 34, 10, 10, 10, 10,133,169, 32, 94,203, 1437
52080 DATA161, 34, 41, 15,  5,169,133,169, 32, 94,203,161, 34,201, 58,240, 1750
52096 DATA  5,104,104, 76, 72,178,165,169, 76, 94,203,162,  1, 32,121,  0, 1562
52112 DATA176,  3, 32,158,183,138, 24,105,219,133,168,169,  0,133,167,133, 1941
52128 DATA171,169, 10, 32,244,180,133, 97,134, 98,132, 99,160, 11,177,167, 2014
52144 DATA248,  8, 41,127,201, 18,208,  2,169,  0, 40, 16,  3, 24,105, 18, 1228
52160 DATA216, 32,235,203,160, 10,177,167, 32,235,203,160,  9,177,167, 32, 2215
52176 DATA235,203,160,  8,177,167, 32,224,203, 76,202,180, 74, 74, 74, 74, 2163
52192 DATA 41, 15,  9, 48,164,171,145, 98,230,171, 96, 72, 32,220,203,104, 1819
52208 DATA 32,224,203,169, 58, 32,228,203, 96,138,175,175,175,175,175,255, 2513
55000 REM GESAMTSUMME DER DATAS = 284024
```

Hier nun das Basic-Programm zum Laden des zweiten Teils der Basic-Erweiterungen. Im Bereich bis Adresse 36864 ($9000) ist nur die Integer-Division und das Rahmenprogramm zur Verwendung als TEST-Funktion enthalten. Da diese selten benötigt wird, können Sie die Zeilen 36352 bis 36848 im allgemeinen weglassen. Im folgenden Bereich 36864 bis 37872 haben wir die Sprite-Befehle aus #5# untergebracht. Der Bereich von 36888 bis 40432 beinhaltet die Grafik-Erweiterungen, und zwar zusätzlich zu denjenigen aus Kapitel 2 noch die aus #4# und #8#. Von 40448 bis 40944 stehen dann die Routinen zur Ein-/Ausgabe-Unterstützung.

Insgesamt stehen Ihnen folgende zusätzliche Befehle zur Verfügung, wenn Sie alle DATA-Zeilen eingeben:

Sprite-Befehle

FAHRT,NR,G,RI	Sprite bewegen
FAHRTAUS	Bewegung aller Sprites anhalten
GRENZEN,NR,L,O,R,U,REF	Grenzen für Sprite setzen
SPRITE,NR,PX,PY	Sprite setzen
SPRITE,NR	Sprite löschen

Grafik-Befehle

BLOCK PF,OX,OY,UX,UY	Block zeichnen
GRLAD DN$,8	Grafik laden
GRLOE	Grafik löschen
GRMISCH DN$,8	Grafik mischen
GRSPEI DN$,8	Grafik speichern
KREIS PF,MX,MY,RX,RY	Kreis zeichnen
LINIE PF,AX,AY,EX,EY	Linie ziehen
RADIUS PF,MX,MY,RX,RY,RW	Radius zeichnen
RECHT PF,OX,OY,UX,UY	Rechteck zeichnen
SCROLL AL	Grafikbildschirm rollen
ZEICHEN PF,MX,MY,AD	eigenes Zeichen ausgeben

Dabei haben die Parameter folgende Bedeutung:

Parameter	**Bedeutung**
AD	Anfangsadresse bei eigenen Zeichen
AL	Anzahl der Linien
DN$	Dateiname
G	Geschwindigkeit
L	linke Grenze
MX	X-Koordinate Mittelpunkt

MY	Y-Koordinate Mittelpunkt
NR	Spritenummer
O	obere Grenze
OX	X-Koordinate obere linke Ecke
OY	Y-Koordinate obere linke Ecke
PF	Punktfarbe
PX	X-Koordinate eines Punktes
PY	Y-Koordinate eines Punktes
R	rechte Grenze
REF	Reflexion
RI	Richtung
RW	Radiuswinkel
RX	Radius in X-Richtung
RY	Radius in Y-Richtung
U	untere Grenze
UX	X-Koordinate untere rechte Ecke
UY	Y-Koordinate untere rechte Ecke
ZF	Zeichenfarbe

Raum für Notizen:

```
100 REM ***  BASIC-LOADER-PROGRAMM  ***            <199>
110 REM ***  FUER MASCHINENROUTINEN ***            <053>
120 REM ***  AUS KAPITEL 2.2        ***            <136>
130 REM                                            <017>
140 REM ***  TEIL 2: BEREICH $8E00-$9FFF  ***      <000>
150 REM ***                  (36352-40959) ***     <173>
160 REM                                            <047>
170 REM ***  DIE ZEILEN-NUMMERN DER DATA-ZEILEN    <136>
180 REM ***  SIND GLEICH DER ZIELADRESSE !!        <243>
190 REM ***  JE DATA-ZEILE EINE PRUEFSUMME         <068>
200 POKE 56,142 : REM BASIC-RAM BEGRENZEN          <101>
210 BA=36352 : REM BEREICHANFANG                   <112>
220 BE=40959 : REM BEREICHENDE                     <247>
230 DEF FN D(X)=PEEK(X)+256*PEEK(X+1)              <100>
240 GS=0     : REM GESAMTPRUEFSUMME                <255>
250 D1=BA    : REM NAECHSTE ZEILENNR.              <077>
260 ZS=0     : REM ZEILENPRUEFSUMME                <045>
270 READ A    : REM 1. ELEMENT LESEN               <128>
280 D=FN D(63): REM NUMMER DER DATA-ZEILE          <234>
290 PRINT"DATA-ZEILE";D;                           <122>
300 IF D<BA THEN PRINT"ZEILENNR. ZU KLEIN":STOP    <055>
310 IF D>BE THEN PRINT"ZEILENNR. ZU GROSS":STOP    <094>
320 IF D=16*INT(D/16)THEN 340                      <113>
330 PRINT"ZEILENNR. NICHT DURCH 16 TEILBAR !"      <052>
340 POKE D,A:ZS=ZS+A                               <206>
350 FOR I=1 TO 15                                  <022>
360 READ A                                         <049>
370 IF A>255 THEN PRINT"FALSCHER WERT":STOP        <100>
380 POKE D+I,A:ZS=ZS+A                             <233>
390 NEXT                                           <009>
400 READ A                                         <089>
410 IF FN D(63)<>D THEN PRINT"ZEILE ZU KURZ":STOP  <051>
420 PRINT"PRUEFSUMME ";                            <198>
430 IF ZS<>A THEN PRINT"FALSCH":STOP               <139>
440 IF ZS=A THEN PRINT"OK"                         <002>
450 IF D>D1 THEN PRINT"ZEILE";D1;"FEHLT":D1=D1+16:
    GOTO 450                                       <154>
460 D1=D1+16                                       <122>
470 GS=GS+ZS                                       <020>
480 IF D+16<BE THEN 260                            <058>
490 PRINT"GESAMTPRUEFSUMME";GS                     <103>
500 END                                            <117>
600 SYS 57812 "E2",8 :REM DATEINAME                <111>
610 POKE 193,BA-256*INT(BA/256)                    <014>
620 POKE 194,INT(BA/256)                           <162>
630 POKE 780,193                                   <119>
640 POKE 781,BE-256*INT(BE/256)                    <055>
650 POKE 782,INT(BE/256)                           <199>
660 SYS 65496         :REM SAVE                    <058>
670 END                                            <032>
```

```
36352 DATA 32,250,174, 32,158,173, 32,141,173, 32,247,183,165, 20,141, 71, 2024
36368 DATA143,165, 21,141, 72,143, 32,253,174, 32,138,173, 32,247,174, 32, 1972
36384 DATA247,183,165, 20,141, 69,143,165, 21,141, 70,143, 32,  0,143,173, 1856
36400 DATA 71,143,133, 99,173, 72,143,133, 98, 56,162,144, 76, 73,188, 48, 1812
36416 DATA 48, 48, 48, 48, 36, 48, 48, 48, 48, 32, 48, 66, 89, 84, 32, 48, 819
36432 DATA 36, 32, 36, 67, 50, 48, 48, 36, 32, 36, 67, 50, 48, 48, 32, 36, 702
36448 DATA 67, 50, 48, 48, 45, 36, 66, 89, 84, 32, 48, 32, 36, 67, 50, 48, 846
36464 DATA 48, 45, 36, 44, 66, 89, 84, 32, 48, 37, 68, 85, 80, 32, 36, 67, 897
36480 DATA 50, 48, 48, 45, 36, 44, 66, 89, 84, 32, 48, 32, 51, 49, 54, 53, 829
36496 DATA 32, 32, 32, 32, 32, 32, 51, 49, 54, 53, 32, 51, 49, 54, 53, 37, 675
36512 DATA 68, 85, 80, 32, 36, 67, 50, 48, 48, 45, 36, 44, 66, 89, 84, 32, 910
36528 DATA 48, 48, 68, 48, 68, 32, 32, 32, 32, 32, 32, 32, 36, 67, 49, 70, 726
36544 DATA 54, 48, 68, 36, 48, 48, 48, 68, 32, 49, 51, 66, 89, 84, 32, 49, 870
36560 DATA 51, 32, 32, 32, 32, 32, 66, 89, 84, 32, 49, 51, 32, 32, 32, 32, 710
36576 DATA 32, 59, 82, 69, 84, 85, 82, 78, 32, 51, 49, 54, 49, 32, 32, 32, 902
36592 DATA 32, 32, 32, 51, 49, 54, 49, 32, 51, 49, 54, 49, 66, 89, 84, 32, 805
36608 DATA162,  0,142, 73,143,142, 74,143,173, 69,143, 13, 70,143,240, 46, 1776
36624 DATA162, 16, 46, 71,143, 46, 72,143, 46, 73,143, 46, 74,143, 56,173, 1453
36640 DATA 73,143,237, 69,143,168,173, 74,143,237, 70,143,144,  6,140, 73, 2036
36656 DATA143,141, 74,143,202,208,219, 46, 71,143, 46, 72,143, 96,104,104, 1955
36672 DATA162, 20,108,  0,  3,  0,  0,  0,  0,  0,  0, 48, 48, 37, 69, 78, 573
36688 DATA 68, 53, 48, 48, 48, 48, 36, 32, 36, 67, 51, 48, 48, 66, 89, 84, 870
36704 DATA 32, 48, 70, 83, 72, 84, 65, 66, 32, 32, 36, 56, 48, 56, 48, 70, 898
36720 DATA 84, 65, 76, 84, 32, 32, 32, 70, 84, 65, 83, 84, 54, 32, 32, 70, 979
36736 DATA 84, 78, 82, 32, 32, 32, 32, 70, 84, 65, 83, 84, 56, 32, 32, 70, 948
36752 DATA 84, 65, 83, 84, 57, 32, 32, 70, 84, 65, 76, 84, 32, 32, 32, 70, 982
36768 DATA 84, 65, 83, 84, 57, 32, 32, 70, 84, 78, 82, 32, 32, 32, 32, 70, 949
36784 DATA 83, 72, 84, 65, 66, 32, 32, 70, 84, 65, 83, 84, 84, 65, 66, 70, 1105
36800 DATA 84, 65, 83, 84, 56, 32, 32, 70, 84, 78, 82, 32, 32, 32, 32, 70, 948
36816 DATA 84, 65, 83, 84, 57, 32, 32, 70, 84, 65, 66, 32, 84, 65, 76, 84, 1063
36832 DATA 32, 32, 32, 32, 75, 69, 89, 69, 78, 68, 32, 32, 75, 69, 89, 69, 942
36848 DATA 78, 68, 32, 32, 84, 65, 76, 84, 32, 32, 32, 32, 75, 69, 89, 34, 914
36864 DATA 76,142,144, 76,229,146, 76, 42,147, 76,120,146, 76,  0,145,  0, 1641
36880 DATA  0,  0,  0,  0,  0,  0,  0,  8,  8,  8,  8,  8,  8,  8,  8,250, 314
```

```
36896 DATA250,250,250,250,250,250,250,  0,  0,  0,  0,  0,  0,  0,  0,  0, 1750
36912 DATA  0,  0,  0,  0,  0,  0,  0, 88,  1, 88,  1, 88,  1, 88,  1, 88,  444
36928 DATA  1, 88,  1, 88,  1, 88,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  268
36944 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
36960 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
36976 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
36992 DATA  0,123, 14,250, 53, 18, 49,232,  1,  2,  0,  0,  0,  0, 32,155,  929
37008 DATA183,142,136,144, 32,253,174, 32,138,173,160,144,162,119, 32,212, 2236
37024 DATA187, 32,253,174, 32,138,173,160,144,169,129, 32, 40,186,160,144, 2153
37040 DATA162,124, 32,212,187, 32,100,226,160,144,169,119, 32, 40,186, 32, 1957
37056 DATA155,188,173,136,144, 10,170,165,101,157, 71,144,165,100,157, 72, 2108
37072 DATA144,160,144,169,124, 32,162,187, 32,107,226, 32,180,191,160,144, 2194
37088 DATA169,119, 32, 40,186, 32,155,188,173,136,144, 10,170,165,101,157, 1977
37104 DATA 87,144,165,100,157, 88,144,173,  9,200,  9,  1,141,  9,200, 96, 1723
37120 DATA216,162,  1,142,138,144,202,138, 10,168,185, 72,144, 48, 53,208, 2031
37136 DATA 24,185, 71,144,208, 19,185, 88,144, 48,119,208,100,185, 87,144, 1959
37152 DATA208, 95,232, 14,138,144,144,223, 96, 32,165,145,173,141,144,217, 2311
37168 DATA 56,144,144, 40,208,  8,173,140,144,217, 55,144,144, 30, 32, 20, 1699
37184 DATA146, 76, 92,145, 32,165,145,185, 40,144,205,141,144,144, 13,208, 2025
37200 DATA  8,185, 39,144,205,140,144,144,  3, 32, 48,146,173,140,144,153, 1848
37216 DATA  0,208,173,141,144,240, 12,173, 16,208, 13,138,144,141, 16,208, 1975
37232 DATA 76, 22,145,173,138,144, 73,255, 45, 16,208,141, 16,208, 76, 22, 1758
37248 DATA145, 32,221,145,221, 31,144,144,  3, 32,238,145,153,  1,208, 76, 1939
37264 DATA 34,145, 32,221,145,221, 23,144,240,  2,176,  3, 32,  1,146,153, 1718
37280 DATA  1,208, 76, 34,145, 24,189,103,144,121, 71,144,157,103,144,185, 1849
37296 DATA  0,208,121, 72,144,141,140,144,169,  0,141,139,144,141,141,144, 1989
37312 DATA173, 16,208, 45,138,144,240,  3,238,139,144,185, 72,144, 41,128, 2058
37328 DATA240,  2,169,255,109,139,144,240,  3,238,141,144, 96, 24,189,111, 2244
37344 DATA144,121, 87,144,157,111,144,185,  1,208,121, 88,144, 96,189, 15, 1955
37360 DATA144, 41,  8,240,  3, 32, 98,146,169,  0,157,111,144,189, 31,144, 1657
37376 DATA 96,189, 15,144, 41,  2,240,  3, 32, 98,146,169,  0,157,111,144, 1587
37392 DATA189, 23,144, 96,189, 15,144, 41,  4,240,  3, 32, 76,146,169,  0, 1511
37408 DATA157,103,144,185, 56,144,141,141,144,185, 55,144,141,140,144, 96, 2120
37424 DATA189, 15,144, 41,  1,240,  3, 32, 76,146,169,  0,157,103,144,185, 1645
```

```
37440 DATA 40,144,141,141,144,185, 39,144,141,140,144, 96, 24,185, 71,144, 1923
37456 DATA 73,255,105,  1,153, 71,144,185, 72,144, 73,255,105,  0,153, 72, 1861
37472 DATA144, 96, 24,185, 87,144, 73,255,105,  1,153, 87,144,185, 88,144, 1915
37488 DATA 73,255,105,  0,153, 88,144, 96, 32,155,183,142,136,144,169,  0, 1875
37504 DATA 56, 42,202, 16,252,141,137,144,160,  0,177,122,201, 44,208, 54, 1956
37520 DATA 32,115,  0, 32,235,183,173,136,144, 10,168,138,153,  1,208,165, 1893
37536 DATA 20,153,  0,208,165, 21,240,  9,173, 16,208, 13,137,144, 76,185, 1768
37552 DATA146,173,137,144, 73,255, 45, 16,208,141, 16,208,173, 21,208, 13, 1977
37568 DATA137,144,141, 21,208, 96,173,137,144, 73,255, 45, 21,208,141, 21, 1965
37584 DATA208,173,136,144, 10,168,169,  0,153, 71,144,153, 72,144,153, 87, 1985
37600 DATA144,153, 88,144, 96, 32,155,183,142,136,144, 32,253,174, 32,235, 2143
37616 DATA183,172,136,144,138,153, 23,144,152, 10,168,165, 20,153, 39,144, 1944
37632 DATA165, 21,153, 40,144, 32,253,174, 32,235,183,172,136,144,138,153, 2175
37648 DATA 31,144,152, 10,168,165, 20,153, 55,144,165, 21,153, 56,144, 32, 1613
37664 DATA155,183,172,136,144,138,153, 15,144, 96,173,  9,200, 41,254,141, 2154
37680 DATA  9,200, 96,239,175,255,175,255, 47,254,174,255,175,255,175,124, 2863
37696 DATA  4, 84,173, 84,  5,212,173, 84, 45, 84,167, 84,134, 84,175,246, 1838
37712 DATA163,243,129,211,171,249,163,251,129, 82,128,209,171, 91,170,209, 2769
37728 DATA  4,212, 13,255,  5,246, 36,244, 37,246, 39,254,164,246,175,255, 2431
37744 DATA  0,243,128,241,128,251,128,217,128,123,  0,217,  0,219,128,241, 2392
37760 DATA 92,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0, 652
37776 DATA255,175,255,175,255,175,255,175,255,175,255,175,255,175,255,175, 3440
37792 DATA 89,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0,209,128, 906
37808 DATA255,175,255,175,255,175,255,175,255,174,127,175,255, 47,255, 45, 3053
37824 DATA127,  0,249,  4,116,  4,126,  4,125,  4,126, 12,255,  4,255,132, 1543
37840 DATA251,171,249, 43,251,171,123,129,209,128,210,129,217, 11,217,137, 2646
37856 DATA 93, 13,255, 47,247,  4,126, 37,117,164,126, 47,118, 39,255,175, 1863
37872 DATA 80,171,208,171, 88,171, 81,137, 80,160, 80,161, 80,171,208,169, 2216
37888 DATA 76,106,148, 76,  6,149, 76,157,148, 76,188,148, 76,116,148, 76, 1770
37904 DATA 39,149, 76, 51,150, 76,212,150, 76, 82,152, 76,  0,153, 76,106, 1624
37920 DATA153, 76, 73,154, 76,209,154, 76,127,155, 76,190,156, 76, 94,157, 2002
37936 DATA 76, 18,157,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 251
37952 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
37968 DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 72,165, 237
```

```
37984 DATA  1,  9,  7,133,  1,104, 88,108,250,255, 32,157,148, 32,188,148, 1661
38000 DATA 32,116,148, 96,169, 94,141,250,255,169,148,141,251,255,169,148, 2582
38016 DATA141,  0,221,169, 59,141, 17,208,169, 59,141, 24,208,173, 22,208, 1960
38032 DATA 41,239, 44, 93,148, 16,  2,  9, 16,141, 22,208, 96,169,224,133, 1601
38048 DATA254,169,  0,133,253,168,145,253,200,208,251,230,254,164,254,192, 3128
38064 DATA255,208,242,168,145,253,200,192, 64,208,249, 96, 32,158,183,138, 2791
38080 DATA 10, 10, 10, 10,141, 92,148, 32,241,183,138, 13, 92,148,160,  0, 1428
38096 DATA140, 93,148,153,  0,204,153,250,204,153,244,205,153,238,206,200, 2744
38112 DATA192,250,208,239, 32,121,  0,240, 28, 32,241,183,169,255,141, 93, 2424
38128 DATA148,138,160,  0,153,  0,216,153,250,216,153,244,217,153,238,218, 2657
38144 DATA200,192,250,208,239, 96,169,151,141,  0,221,169, 27,141, 17,208, 2429
38160 DATA169, 21,141, 24,208,173, 22,208, 41,239,141, 22,208, 96,165,  1, 1879
38176 DATA  9,  7,133,  1, 76, 72,178, 32,239,149, 32,253,174, 32,235,183, 1805
38192 DATA224,200,176,234, 44, 93,148, 16,  3, 76,196,149,165, 21,240, 10, 1995
38208 DATA201,  1,208,218,165, 20,201, 64,176,212,165, 20, 41,  7,168, 56, 1923
38224 DATA169,  0,106,136, 16,252,133,171,165, 20, 41,248,133, 20,  8,120, 1738
38240 DATA165,  1, 72, 41,253,133,  1,138, 41,  7, 24,101, 20,133, 20,165, 1315
38256 DATA 21,105,224,133, 21,138, 41,248,133, 89,169,  0,133, 90,169, 40, 1754
38272 DATA133, 91, 32,  5,150, 24,165, 20,101, 87,133, 20,165, 21,101, 88, 1336
38288 DATA133, 21,165,171,160,  0, 44, 93,148,174, 92,148,112, 18, 16,  4, 1499
38304 DATA 81, 20, 80, 25,240,  4, 17, 20, 80, 19, 73,255, 49, 20, 80, 13, 1076
38320 DATA 72, 73,255, 49, 20,133,171,104, 61,235,149,  5,171,145, 20,104, 1767
38336 DATA133,  1, 40, 96,165, 21,240,  3, 76, 30,149,165, 20,201,160,176, 1676
38352 DATA247, 41,  3,168,185,231,149,133,171,165, 20, 41,252, 10,133, 20, 1969
38368 DATA144,  2,230, 21, 76, 94,149,192, 48, 12,  3,  0, 85,170,255, 44, 1525
38384 DATA 93,148, 48, 10, 32,138,173, 32, 43,188,141, 92,148, 96, 32,158, 1572
38400 DATA183,142, 92,148, 96,169,  0,133, 87,133, 88,160,  8, 24, 38, 91, 1592
38416 DATA144, 13, 24,165, 87,101, 89,133, 87,165, 88,101, 90,133, 88,136, 1644
38432 DATA240,  6, 38, 87, 38, 88,144,230, 96,128, 64, 32, 16,  8,  4,  2, 1221
38448 DATA  1,  0,  0, 32,239,149, 32,253,174, 32,235,183,134,166,165, 20, 1815
38464 DATA133,164,165, 21,133,165, 32,253,174, 32,138,173,230, 97,230, 97, 2237
38480 DATA230, 97, 32,247,183,132,253, 24,105,208,133,254,120,165,  1, 41, 2225
38496 DATA251,133,  1,169,  7,141, 50,150,169,  7,141, 49,150,174, 49,150, 1791
38512 DATA172, 50,150,177,253, 61, 41,150,240, 23, 24,165,164,109, 49,150, 1978
```

```
38528 DATA133, 20,165,165,105,   0,133, 21,165,166,109, 50,150,170,  32, 48, 1632
38544 DATA149,206, 49,150, 16,215,206, 50,150, 16,205,165,   1,   9,   4,133, 1724
38560 DATA   1, 88, 96,169,   0,133, 92,133, 93,165, 89,208,   7,165, 90,208, 1737
38576 DATA   3, 76,138,187,165, 88,208,   4,165, 87,240, 23, 56,165, 87,229, 1921
38592 DATA 89,133, 87,165, 88,229, 90,133, 88,144,   8,230, 92,208,229,230, 2243
38608 DATA 93,208,225, 96, 32,239,149, 32,253,174, 32,235,183,134,166,165, 2416
38624 DATA 20,133,164,165, 21,133,165, 32,253,174, 32,235,183,134,169,166, 2179
38640 DATA 20,134,167,165, 21,133,168,166,167,165,168,197,165,144,   8,208, 2196
38656 DATA 30,228,164,240, 88,176, 24,165,164,166,167,133,167,134,164,165, 2375
38672 DATA165,166,168,133,168,134,165,165,166,166,169,133,169,134,166, 56, 2423
38688 DATA165,167,229,164,141, 76,148,165,168,229,165,141, 77,148,162,   0, 2345
38704 DATA142, 78,148, 56,165,169,229,166,240, 70,238, 78,148,141, 79,148, 2295
38720 DATA176, 11, 73,255,105,   1,202,142, 78,148,141, 79,148,173, 77,148, 1957
38736 DATA208, 80,173, 76,148,205, 79,148,176, 72, 76,   0,152,165,169,197, 2124
38752 DATA166,176,   6,166,166,134,169,133,166,166,166,138, 72,165,164,133, 2286
38768 DATA 20,165,165,133, 21, 32, 48,149,104,170,232,228,169,144,236, 96, 2112
38784 DATA165,164,133, 20,165,165,133, 21,166,169, 32, 48,149,230,164,208, 2132
38800 DATA  2,230,165,165,168,197,165,144,   8,208,229,165,167,197,164,176, 2550
38816 DATA223, 96,169,   0,141, 80,148,141, 81,148, 24,173, 80,148,133, 89, 1874
38832 DATA101,164,133, 20,173, 81,148,133, 90,101,165,133, 21,173, 79,148, 1863
38848 DATA133, 91, 32,   5,150,173, 76,148,133, 89,173, 77,148,133, 90, 32, 1683
38864 DATA163,150,165,166, 44, 78,148, 48,   5, 24,101, 92,144,   3, 56,229, 1616
38880 DATA 92,170, 32, 48,149,238, 80,148,208,   3,238, 81,148,173, 77,148, 2033
38896 DATA205, 81,148,144, 10,208,179,173, 76,148,205, 80,148,176,171, 96, 2248
38912 DATA169,   0,141, 82,148,133, 91,165,166, 44, 78,148, 48,   6, 24,109, 1552
38928 DATA 82,148,144,   4, 56,237, 82,148, 72,173, 76,148,133, 89,173, 77, 1842
38944 DATA148,133, 90, 32,   5,150,173, 79,148,133, 89,169,   0,133, 90, 32, 1604
38960 DATA163,150, 24,165, 92,101,164,133, 20,165, 93,101,165,133, 21,104, 1794
38976 DATA170, 32, 48,149,238, 82,148,173, 82,148,205, 79,148,240,182,144, 2268
38992 DATA180, 96, 32,239,149, 32,253,174, 32,235,183,142, 85,148,165, 20, 2165
39008 DATA141, 83,148,165, 21,141, 84,148, 32,253,174, 32,235,183,142, 88, 2070
39024 DATA148,166, 20,142, 86,148,165, 21,141, 87,148,173, 83,148,133,164, 1973
39040 DATA173, 84,148,133,165,173, 85,148,133,166,173, 86,148,133,167,173, 2288
39056 DATA 87,148,133,168,173, 85,148,133,169, 32,247,150,173, 86,148,133, 2213
```

```
39072 DATA164,173, 87,148,133,165,173, 85,148,133,166,173, 86,148,133,167, 2282
39088 DATA173, 87,148,133,168,173, 88,148,133,169, 32,247,150,173, 86,148, 2256
39104 DATA133,164,173, 87,148,133,165,173, 88,148,133,166,173, 83,148,133, 2248
39120 DATA167,173, 84,148,133,168,173, 88,148,133,169, 32,247,150,173, 83, 2269
39136 DATA148,133,164,173, 84,148,133,165,173, 88,148,133,166,173, 83,148, 2260
39152 DATA133,167,173, 84,148,133,168,173, 85,148,133,169, 32,247,150, 96, 2239
39168 DATA 32,239,149, 32,253,174, 32,235,183,142, 85,148,165, 20,141, 83, 2113
39184 DATA148,165, 21,141, 84,148, 32,253,174, 32,235,183,142, 88,148,166, 2160
39200 DATA 20,142, 86,148,165, 21,141, 87,148,173, 88,148,205, 85,148,176, 1981
39216 DATA  9,174, 85,148,142, 88,148,141, 85,148,174, 85,148,142, 82,148, 1947
39232 DATA173, 83,148,133,164,173, 84,148,133,165,173, 82,148,133,166,133, 2239
39248 DATA169,173, 86,148,133,167,173, 87,148,133,168, 32,247,150,238, 82, 2334
39264 DATA148,172, 88,148,204, 82,148,176,215, 96, 32,239,149, 32,253,174, 2356
39280 DATA 32,235,183,142, 91,148,165, 20,141, 89,148,165, 21,141, 90,148, 1959
39296 DATA 32,253,174, 32,138,173,160,148,162, 51, 32,212,187, 32,253,174, 2213
39312 DATA 32,138,173,160,148,162, 56, 32,212,187,169, 51,160,148, 32, 80, 1940
39328 DATA184, 32, 43,188, 48, 10,169, 51,160,148, 32,162,187, 76,183,153, 1826
39344 DATA169, 56,160,148, 32,162,187,169,188,160,185, 32, 15,187,162, 66, 2078
39360 DATA160,148, 32,212,187,169, 32,141, 62,148,169,  0,141, 61,148,141, 1951
39376 DATA 63,148,141, 64,148,141, 65,148,169, 61,160,148, 32,162,187, 32, 1869
39392 DATA107,226,169, 51,160,148, 32, 40,186,169, 17,160,191, 32,103,184, 1975
39408 DATA 32,155,188, 24,165,101,109, 89,148,133, 20,165,100,109, 90,148, 1776
39424 DATA133, 21,169, 61,160,148, 32,162,187, 32,100,226,169, 56,160,148, 1964
39440 DATA 32, 40,186,169, 17,160,191, 32,103,184, 32,155,188, 24,165,101, 1779
39456 DATA109, 91,148,170, 32, 48,149,169, 61,160,148, 32,162,187,169, 66, 1901
39472 DATA160,148, 32,103,184,162, 61,160,148, 32,212,187,169,  9,160,227, 2154
39488 DATA 32, 80,184, 32, 43,188, 16,144, 96, 32,239,149, 32,253,174, 32, 1726
39504 DATA235,183,134,166,165, 20,133,164,165, 21,133,165, 32,253,174, 32, 2175
39520 DATA138,173,160,148,162, 51, 32,212,187, 32,253,174, 32,138,173,160, 2225
39536 DATA148,162, 56, 32,212,187, 32,253,174, 32,138,173,160,148,162, 71, 2140
39552 DATA 32,212,187, 32,107,226,169, 51,160,148, 32, 40,186,169, 17,160, 1928
39568 DATA191, 32,103,184, 32,155,188, 24,165,101,101,164,133,167,165,100, 2005
39584 DATA101,165,133,168,169, 71,160,148, 32,162,187, 32,100,226,169, 56, 2079
39600 DATA160,148, 32, 40,186,169, 17,160,191, 32,103,184, 32,155,188, 24, 1821
```

```
39616 DATA165,101,101,166,133,169, 32,247,150, 96,  0,  0,  0,  0,  0,  0, 1360
39632 DATA  0, 32,239,149, 32,253,174, 32,235,183,142,204,154,165, 20,141, 2155
39648 DATA202,154,165, 21,141,203,154, 32,253,174, 32,138,173, 32,247,183, 2304
39664 DATA120,165,  1, 41,254,133,  1,160,  0,177, 20, 10,141,205,154,200, 1782
39680 DATA177, 20, 42,141,206,154,165, 20, 24,105,  2,133,253,169,  0,101, 1712
39696 DATA 21,133,254,160,  0,140,207,154,160,  0,173,206,154,205,207,154, 2328
39712 DATA240, 23, 32, 86,155,140,208,154, 32, 48,149,172,208,154,200,208, 2209
39728 DATA241,238,207,154,230,254, 76, 24,155,204,205,154,240, 16, 32, 86, 2516
39744 DATA155,140,208,154, 32, 48,149,172,208,154,200, 76, 57,155,165,  1, 2074
39760 DATA  9,  1,133,  1, 88, 96, 24,177,253, 48, 15,109,202,154,133, 20, 1463
39776 DATA169,  0,109,203,154,133, 21, 76,118,155,109,202,154,133, 20,169, 1925
39792 DATA255,109,203,154,133, 21,200, 24,177,253,109,204,154,170, 96, 32, 2294
39808 DATA158,183,142,167,155,138, 74, 74, 74,240, 10, 72, 32, 58,156,104, 1837
39824 DATA 56,233,  1,208,246,173,167,155, 41,  7,240, 10, 72, 32,168,155, 1964
39840 DATA104, 56,233,  1,208,246, 96,  0, 32,170,156,169,  0,133,253,169, 2026
39856 DATA224,133,254,169,199,133, 20,169,222,133, 21,162,  0,160,  1,152, 2152
39872 DATA 41,  7,201,  0,240, 95,177,253,136,145,253,200,200,192,201,208, 2549
39888 DATA238,232,169,200, 24,101,253,133,253,169,  0,101,254,133,254,169, 2683
39904 DATA200, 24,101, 20,133, 20,169,  0,101, 21,133, 21, 32,180,156, 32, 1343
39920 DATA170,156,224, 40,208,199,169,  7,133, 20,169,254,133, 21,162,  0, 2065
39936 DATA160,  0,169,  0,145, 20,152, 24,105,  8,168,192,168,208,243,165, 1927
39952 DATA 20, 24,105,160,133, 20,165, 21,105,  0,133, 21,232,224,  2,208, 1573
39968 DATA223, 32,180,156, 96,165, 21,201,222,240,161,201,223,208,  4,192, 2525
39984 DATA112,144,153,177,253,145, 20, 76,204,155, 32,170,156,169,  0,133, 2099
40000 DATA 20,169,224,133, 21,169, 64,133,253,169,225,133,254,162,  0,160, 2289
40016 DATA  0,177,253,145, 20,200,192,192,208,247,165,253, 24,105,192,133, 2506
40032 DATA253,165,254,105,  0,133,254,165, 20, 24,105,192,133, 20,165, 21, 2009
40048 DATA105,  0,133, 21, 32,180,156, 32,170,156,232,224, 40,208,208,169, 2066
40064 DATA  0,133,253,169,254,133,254,162,  0,160,  0,169,  0,145,253,200, 2285
40080 DATA192, 40,208,249,165,253, 24,105, 40,133,253,165,254,105,  0,133, 2319
40096 DATA254,232,224,  8,208,227, 32,180,156, 96,120, 72,165,  1, 41,253, 2269
40112 DATA133,  1,104, 96, 72,165,  1,  9,  2,133,  1,104, 88, 96, 32,212, 1249
40128 DATA225,169, 97,133,185,165,183,208,  3, 76, 16,247, 32,213,243, 32, 2227
40144 DATA143,246,165,186, 32,177,255,165,185, 32,147,255,169,  0, 32,168, 2357
```

```
40160 DATA255,133,172,169,224, 32,168,255,133,173, 32,170,156,160,  0,177, 2409
40176 DATA172, 32,180,156, 32,168,255,230,172,208,  2,230,173,165,173,201, 2549
40192 DATA255,144,231,165,172,201, 64,144,225, 32,174,255, 76, 66,246, 76, 2526
40208 DATA  4,247, 32,212,225,169, 96,133,185,165,183,208,  3, 76, 16,247, 2201
40224 DATA 32,213,243, 32,175,245,165,186, 32,180,255,165,185, 32,150,255, 2545
40240 DATA 32,165,255,133,174, 32,183,255,208,213, 32,165,255,133,175, 32, 2442
40256 DATA183,255,208, 23, 32,165,255, 32,170,156,160,  0, 17,174,145,174, 2149
40272 DATA 32,180,156,230,174,208,232,230,175,208,228, 76, 66,246, 32,212, 2685
40288 DATA225,169,  0,162,  0,160,224, 76,213,255,169, 50,160,148, 32, 40, 2083
40304 DATA186,169, 17,160,191, 32,103,184, 32,155,188, 24,165,101,109, 72, 1888
40320 DATA148,141, 75,148, 32, 11,153, 96, 72,165,  1,  9,  7,133,  1,104, 1296
40336 DATA 88,108,250,255,160,175,255,175,255,175,255,175,255,175,255,175, 3186
40352 DATA 89,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0, 80,  0,208,  0, 777
40368 DATA255,175,255,175,255,175,255,175,255,175,255,175,255,175,255, 41, 3306
40384 DATA123,  4,253,  1, 88,  0,126,  4,248,  4,254,  8,246,  4,255,164, 1782
40400 DATA251,171,249, 42, 91,171,123,  1,208,130, 81,129,217, 11,217,161, 2253
40416 DATA 93, 13,126, 15,117,  4,126, 45,125, 37,127, 47,126, 39,254,175, 1469
40432 DATA 80,171,208,171, 89,171, 81,169,208,130, 80,129, 80, 11,208,169, 2155
40448 DATA 76,173,158, 76, 69,159, 76,238,158, 76,107,159,  0,  0,  0,  0, 1525
40464 DATA160,  2,177, 71,133, 35,136,177, 71,133, 34,136,177, 71,133,170, 1816
40480 DATA 96,160,  2,165, 35,145, 71,136,165, 34,145, 71,136,165,170,145, 1841
40496 DATA 71, 96,165,171,197,170,240,  9,144,  7, 32,244,180,134, 34,132, 2026
40512 DATA 35,133,170,168,240, 20,165,  1, 72, 41,254,133,  1,136,185,  0, 1754
40528 DATA160,145, 34,192,  0,208,246,104,133,  1, 96,170, 16, 42,201,255, 2003
40544 DATA240, 38, 36, 15, 48, 34, 56,233,127,170,160,255,202,240,  8,200, 2062
40560 DATA185,158,160, 16,250, 48,245,200,185,158,160, 48,  9,166,171,230, 2389
40576 DATA171,157,  0,160,208,241, 41,127,201, 34,208,  8,165, 15, 73,255, 2064
40592 DATA133, 15,169, 34, 96,162,  0, 32,121,  0,201, 35,208, 12, 32,115, 1365
40608 DATA  0, 32,158,183, 32, 30,225, 32,253,174,134, 19, 96,169,  0,133, 1670
40624 DATA171,169,  8, 32,180,255,169,111, 32,150,255, 32,165,255,164,171, 2319
40640 DATA153,  0,160,230,171, 36,157, 16,  3, 32,210,255,201, 13,208,235, 2080
40656 DATA 32,171,255,198,171,169, 68,133, 69,169,211,133, 70,169,255,133, 2406
40672 DATA 13, 32,231,176, 32, 16,158, 32, 50,158, 32, 33,158, 96,169,  0, 1386
40688 DATA133, 15,133,171, 32,149,158, 32,139,176, 32, 16,158, 32, 36,225, 1637
```

```
40704 DATA141, 14,158, 32, 36,225,141, 15,158,168,240, 39, 32, 36,225,141, 1801
40720 DATA 12,158, 32, 36,225,141, 13,158, 32, 36,225, 32, 91,158,164,171, 1684
40736 DATA153,  0,160,168,240, 13,165,144,208,  9,230,171,208,234,162, 23, 2288
40752 DATA 76, 55,164, 32, 50,158, 32, 33,158,166, 19,240,  7, 32,204,255, 1681
40768 DATA162,  0,134, 19, 96,169,  0,133,171, 32,149,158, 32,139,176, 32, 1602
40784 DATA 16,158, 32, 18,225,164,171,153,  0,160,201, 13,240,213,165,144, 2073
40800 DATA208,209,230,171,208,236,162, 23, 76, 55,164, 32,212,225,169, 96, 2476
40816 DATA133,185,165,183,240,115, 32,213,243,165,186, 32,180,255,165,185, 2677
40832 DATA 32,150,255,160,  3,132, 97, 32,165,255,133, 98,164,144,208, 89, 2117
40848 DATA 32,165,255,164,144,208, 82,198, 97,208,236,166, 98, 32,205,189, 2479
40864 DATA 32, 63,171,169,  0,133,171,133, 15, 32,165,255,164,144,208, 57, 1912
40880 DATA170,240, 12, 32, 91,158,166,171,230,171,157,  0,160,208,234,165, 2365
40896 DATA  1, 72, 41,254,133,  1,160,  0,132, 97,230, 97,185,  0,160, 32, 1595
40912 DATA210,255,164, 97,196,171,208,240,104,133,  1,169, 13, 32,210,255, 2458
40928 DATA160,  2,132, 97, 32,225,255,208,158, 32, 66,246, 96,  0,  0,  0, 1709
40944 DATA175,175,175,175,175,175,175,175,175,175,175,175, 65, 83, 83, 49, 2380
55000 REM GESAMTSUMME DER DATAS = 535381
```

Anhang 4: Hex-Listing der Basic-Erweiterungen

Haben Sie ein Monitor-Programm, können Sie die Basic-Erweiterungen auch hexadezimal mit Hilfe des unten abgebildeten Listings eingeben. Sie können dabei natürlich ebenfalls die Bereiche weglassen, die Sie nicht benötigen. Die zusätzlichen Befehle aus #4#, #5# und #8# sind hier ebenfalls enthalten. Die Beschreibung dieser Befehle lesen Sie bitte im Anhang 3 nach; die entsprechenden Speicherbereiche sind hexadezimal (vgl. auch Kapitel 2.2):

TEST-Funktion	: $8E00 bis $8EFF
Integer-Division	: $8F00 bis $8FFF
Sprite-Befehle aus #5#	: $9000 bis $93FF
Grafik-Befehle aus Kapitel 2	: $9400 bis $96A2
Grafik-Befehle aus #4# und #8#	: $96A3 bis $9DFF
Ein-/Ausgabe-Befehle	: $9E00 bis $9FFF

```
8E00  20 FA AE 20 9E AD 20 8D AD 20 F7 B7 A5 14 8D 47
8E10  8F A5 15 8D 48 8F 20 FD AE 20 8A AD 20 F7 AE 20
8E20  F7 B7 A5 14 8D 45 8F A5 15 8D 46 8F 20 00 8F AD
8E30  47 8F 85 63 AD 48 8F 85 62 38 A2 90 4C 49 BC 30
8E40  30 30 30 30 24 30 30 30 30 20 30 42 59 54 20 30
8E50  24 20 24 43 32 30 30 24 20 24 43 32 30 30 20 24
8E60  43 32 30 30 2D 24 42 59 54 20 30 20 24 43 32 30
8E70  30 2D 24 2C 42 59 54 20 30 25 44 55 50 20 24 43
8E80  32 30 30 2D 24 2C 42 59 54 20 30 20 33 31 36 35
8E90  20 20 20 20 20 20 33 31 36 35 20 33 31 36 35 25
8EA0  44 55 50 20 24 43 32 30 30 2D 24 2C 42 59 54 20
8EB0  30 30 44 30 44 20 20 20 20 20 20 20 24 43 31 46
8EC0  36 30 44 24 30 30 30 44 20 31 33 42 59 54 20 31
8ED0  33 20 20 20 20 20 42 59 54 20 31 33 20 20 20 20
8EE0  20 3B 52 45 54 55 52 4E 20 33 31 36 31 20 20 20
8EF0  20 20 20 33 31 36 31 20 33 31 36 31 42 59 54 20
8F00  A2 00 8E 49 8F 8E 4A 8F AD 45 8F 0D 46 8F F0 2E
8F10  A2 10 2E 47 8F 2E 48 8F 2E 49 8F 2E 4A 8F 38 AD
8F20  49 8F ED 45 8F A8 AD 4A 8F ED 46 8F 90 06 8C 49
8F30  8F 8D 4A 8F CA D0 DB 2E 47 8F 2E 48 8F 60 68 68
```

```
8F40  A2 14 6C 00 03 00 00 00 00 00 00 30 30 25 45 4E
8F50  44 35 30 30 30 30 24 20 24 43 33 30 30 42 59 54
8F60  20 30 46 53 48 54 41 42 20 20 24 38 30 38 30 46
8F70  54 41 4C 54 20 20 20 46 54 41 53 54 36 20 20 46
8F80  54 4E 52 20 20 20 20 46 54 41 53 54 38 20 20 46
8F90  54 41 53 54 39 20 20 46 54 41 4C 54 20 20 20 46
8FA0  54 41 53 54 39 20 20 46 54 4E 52 20 20 20 20 46
8FB0  53 48 54 41 42 20 20 46 54 41 53 54 54 41 42 46
8FC0  54 41 53 54 38 20 20 46 54 4E 52 20 20 20 20 46
8FD0  54 41 53 54 39 20 20 46 54 41 42 20 54 41 4C 54
8FE0  20 20 20 20 4B 45 59 45 4E 44 20 20 4B 45 59 45
8FF0  4E 44 20 20 54 41 4C 54 20 20 20 20 4B 45 59 22
9000  4C 8E 90 4C E5 92 4C 2A 93 4C 78 92 4C 00 91 00
9010  00 00 00 00 00 00 00 08 08 08 08 08 08 08 08 FA
9020  FA FA FA FA FA FA FA 00 00 00 00 00 00 00 00 00
9030  00 00 00 00 00 00 00 58 01 58 01 58 01 58 01 58
9040  01 58 01 58 01 58 01 00 00 00 00 00 00 00 00 00
9050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9080  00 7B 0E FA 35 12 31 E8 01 02 00 00 00 00 20 9B
9090  B7 8E 88 90 20 FD AE 20 8A AD A0 90 A2 77 20 D4
90A0  BB 20 FD AE 20 8A AD A0 90 A9 81 20 28 BA A0 90
90B0  A2 7C 20 D4 BB 20 64 E2 A0 90 A9 77 20 28 BA 20
90C0  9B BC AD 88 90 0A AA A5 65 9D 47 90 A5 64 9D 48
90D0  90 A0 90 A9 7C 20 A2 BB 20 6B E2 20 B4 BF A0 90
90E0  A9 77 20 28 BA 20 9B BC AD 88 90 0A AA A5 65 9D
90F0  57 90 A5 64 9D 58 90 AD 09 C8 09 01 8D 09 C8 60
9100  D8 A2 01 8E 8A 90 CA 8A 0A A8 B9 48 90 30 35 D0
9110  18 B9 47 90 D0 13 B9 58 90 30 77 D0 64 B9 57 90
9120  D0 5F E8 0E 8A 90 90 DF 60 20 A5 91 AD 8D 90 D9
9130  38 90 90 28 D0 08 AD 8C 90 D9 37 90 90 1E 20 14
9140  92 4C 5C 91 20 A5 91 B9 28 90 CD 8D 90 90 0D D0
9150  08 B9 27 90 CD 8C 90 90 03 20 30 92 AD 8C 90 99
9160  00 D0 AD 8D 90 F0 0C AD 10 D0 0D 8A 90 8D 10 D0
9170  4C 16 91 AD 8A 90 49 FF 2D 10 D0 8D 10 D0 4C 16
9180  91 20 DD 91 DD 1F 90 90 03 20 EE 91 99 01 D0 4C
9190  22 91 20 DD 91 DD 17 90 F0 02 B0 03 20 01 92 99
91A0  01 D0 4C 22 91 18 BD 67 90 79 47 90 9D 67 90 B9
91B0  00 D0 79 48 90 8D 8C 90 A9 00 8D 8B 90 8D 8D 90
91C0  AD 10 D0 2D 8A 90 F0 03 EE 8B 90 B9 48 90 29 80
91D0  F0 02 A9 FF 6D 8B 90 F0 03 EE 8D 90 60 18 BD 6F
91E0  90 79 57 90 9D 6F 90 B9 01 D0 79 58 90 60 BD 0F
91F0  90 29 08 F0 03 20 62 92 A9 00 9D 6F 90 BD 1F 90
9200  60 BD 0F 90 29 02 F0 03 20 62 92 A9 00 9D 6F 90
9210  BD 17 90 60 BD 0F 90 29 04 F0 03 20 4C 92 A9 00
9220  9D 67 90 B9 38 90 8D 8D 90 B9 37 90 8D 8C 90 60
9230  BD 0F 90 29 01 F0 03 20 4C 92 A9 00 9D 67 90 B9
9240  28 90 8D 8D 90 B9 27 90 8D 8C 90 60 18 B9 47 90
9250  49 FF 69 01 99 47 90 B9 48 90 49 FF 69 00 99 48
9260  90 60 18 B9 57 90 49 FF 69 01 99 57 90 B9 58 90
```

```
9270  49 FF 69 00 99 58 90 60 20 9B B7 8E 88 90 A9 00
9280  38 2A CA 10 FC 8D 89 90 A0 00 B1 7A C9 2C D0 36
9290  20 73 00 20 EB B7 AD 88 90 0A A8 8A 99 01 D0 A5
92A0  14 99 00 D0 A5 15 F0 09 AD 10 D0 0D 89 90 4C B9
92B0  92 AD 89 90 49 FF 2D 10 D0 8D 10 D0 AD 15 D0 0D
92C0  89 90 8D 15 D0 60 AD 89 90 49 FF 2D 15 D0 8D 15
92D0  D0 AD 88 90 0A A8 A9 00 99 47 90 99 48 90 99 57
92E0  90 99 58 90 60 20 9B B7 8E 88 90 20 FD AE 20 EB
92F0  B7 AC 88 90 8A 99 17 90 98 0A A8 A5 14 99 27 90
9300  A5 15 99 28 90 20 FD AE 20 EB B7 AC 88 90 8A 99
9310  1F 90 98 0A A8 A5 14 99 37 90 A5 15 99 38 90 20
9320  9B B7 AC 88 90 8A 99 0F 90 60 AD 09 C8 29 FE 8D
9330  09 C8 60 EF AF FF AF FF 2F FE AE FF AF FF AF 7C
9340  04 54 AD 54 05 D4 AD 54 2D 54 A7 54 86 54 AF F6
9350  A3 F3 81 D3 AB F9 A3 FB 81 52 80 D1 AB 5B AA D1
9360  04 D4 0D FF 05 F6 24 F4 25 F6 27 FE A4 F6 AF FF
9370  00 F3 80 F1 80 FB 80 D9 80 7B 00 D9 00 DB 80 F1
9380  5C 00 50 00 50 00 50 00 50 00 50 00 50 00 50 00
9390  FF AF FF AF FF AF FF AF FF AF FF AF FF AF FF AF
93A0  59 00 50 00 50 00 50 00 50 00 50 00 50 00 D1 80
93B0  FF AF FF AF FF AF FF AF FF AE 7F AF FF 2F FF 2D
93C0  7F 00 F9 04 74 04 7E 04 7D 04 7E 0C FF 04 FF 84
93D0  FB AB F9 2B FB AB 7B 81 D1 80 D2 81 D9 0B D9 89
93E0  5D 0D FF 2F F7 04 7E 25 75 A4 7E 2F 76 27 FF AF
93F0  50 AB D0 AB 58 AB 51 89 50 A0 50 A1 50 AB D0 A9
9400  4C 6A 94 4C 06 95 4C 9D 94 4C BC 94 4C 74 94 4C
9410  27 95 4C 33 96 4C D4 96 4C 52 98 4C 00 99 4C 6A
9420  99 4C 49 9A 4C D1 9A 4C 7F 9B 4C BE 9C 4C 5E 9D
9430  4C 12 9D 00 00 00 00 00 00 00 00 00 00 00 00 00
9440  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9450  00 00 00 00 00 00 00 00 00 00 00 00 00 00 48 A5
9460  01 09 07 85 01 68 58 6C FA FF 20 9D 94 20 BC 94
9470  20 74 94 60 A9 5E 8D FA FF A9 94 8D FB FF A9 94
9480  8D 00 DD A9 3B 8D 11 D0 A9 3B 8D 18 D0 AD 16 D0
9490  29 EF 2C 5D 94 10 02 09 10 8D 16 D0 60 A9 E0 85
94A0  FE A9 00 85 FD A8 91 FD C8 D0 FB E6 FE A4 FE C0
94B0  FF D0 F2 A8 91 FD C8 C0 40 D0 F9 60 20 9E B7 8A
94C0  0A 0A 0A 0A 8D 5C 94 20 F1 B7 8A 0D 5C 94 A0 00
94D0  8C 5D 94 99 00 CC 99 FA CC 99 F4 CD 99 EE CE C8
94E0  C0 FA D0 EF 20 79 00 F0 1C 20 F1 B7 A9 FF 8D 5D
94F0  94 8A A0 00 99 00 D8 99 FA D8 99 F4 D9 99 EE DA
9500  C8 C0 FA D0 EF 60 A9 97 8D 00 DD A9 1B 8D 11 D0
9510  A9 15 8D 18 D0 AD 16 D0 29 EF 8D 16 D0 60 A5 01
9520  09 07 85 01 4C 48 B2 20 EF 95 20 FD AE 20 EB B7
9530  E0 C8 B0 EA 2C 5D 94 10 03 4C C4 95 A5 15 F0 0A
9540  C9 01 D0 DA A5 14 C9 40 B0 D4 A5 14 29 07 A8 38
9550  A9 00 6A 88 10 FC 85 AB A5 14 29 F8 85 14 08 78
9560  A5 01 48 29 FD 85 01 8A 29 07 18 65 14 85 14 A5
9570  15 69 E0 85 15 8A 29 F8 85 59 A9 00 85 5A A9 28
9580  85 5B 20 05 96 18 A5 14 65 57 85 14 A5 15 65 58
9590  85 15 A5 AB A0 00 2C 5D 94 AE 5C 94 70 12 10 04
```

```
95A0  51 14 50 19 F0 04 11 14 50 13 49 FF 31 14 50 0D
95B0  48 49 FF 31 14 85 AB 68 3D EB 95 05 AB 91 14 68
95C0  85 01 28 60 A5 15 F0 03 4C 1E 95 A5 14 C9 A0 B0
95D0  F7 29 03 A8 B9 E7 95 85 AB A5 14 29 FC 0A 85 14
95E0  90 02 E6 15 4C 5E 95 C0 30 0C 03 00 55 AA FF 2C
95F0  5D 94 30 0A 20 8A AD 20 2B BC 8D 5C 94 60 20 9E
9600  B7 8E 5C 94 60 A9 00 85 57 85 58 A0 08 18 26 5B
9610  90 0D 18 A5 57 65 59 85 57 A5 58 65 5A 85 58 88
9620  F0 06 26 57 26 58 90 E6 60 80 40 20 10 08 04 02
9630  01 00 00 20 EF 95 20 FD AE 20 EB B7 86 A6 A5 14
9640  85 A4 A5 15 85 A5 20 FD AE 20 8A AD E6 61 E6 61
9650  E6 61 20 F7 B7 84 FD 18 69 D0 85 FE 78 A5 01 29
9660  FB 85 01 A9 07 8D 32 96 A9 07 8D 31 96 AE 31 96
9670  AC 32 96 B1 FD 3D 29 96 F0 17 18 A5 A4 6D 31 96
9680  85 14 A5 A5 69 00 85 15 A5 A6 6D 32 96 AA 20 30
9690  95 CE 31 96 10 D7 CE 32 96 10 CD A5 01 09 04 85
96A0  01 58 60 A9 00 85 5C 85 5D A5 59 D0 07 A5 5A D0
96B0  03 4C 8A BB A5 58 D0 04 A5 57 F0 17 38 A5 57 E5
96C0  59 85 57 A5 58 E5 5A 85 58 90 08 E6 5C D0 E5 E6
96D0  5D D0 E1 60 20 EF 95 20 FD AE 20 EB B7 86 A6 A5
96E0  14 85 A4 A5 15 85 A5 20 FD AE 20 EB B7 86 A9 A6
96F0  14 86 A7 A5 15 85 A8 A6 A7 A5 A8 C5 A5 90 08 D0
9700  1E E4 A4 F0 58 B0 18 A5 A4 A6 A7 85 A7 86 A4 A5
9710  A5 A6 A8 85 A8 86 A5 A5 A6 A6 A9 85 A9 86 A6 38
9720  A5 A7 E5 A4 8D 4C 94 A5 A8 E5 A5 8D 4D 94 A2 00
9730  8E 4E 94 38 A5 A9 E5 A6 F0 46 EE 4E 94 8D 4F 94
9740  B0 0B 49 FF 69 01 CA 8E 4E 94 8D 4F 94 AD 4D 94
9750  D0 50 AD 4C 94 CD 4F 94 B0 48 4C 00 98 A5 A9 C5
9760  A6 B0 06 A6 A6 86 A9 85 A6 A6 A6 8A 48 A5 A4 85
9770  14 A5 A5 85 15 20 30 95 68 AA E8 E4 A9 90 EC 60
9780  A5 A4 85 14 A5 A5 85 15 A6 A9 20 30 95 E6 A4 D0
9790  02 E6 A5 A5 A8 C5 A5 90 08 D0 E5 A5 A7 C5 A4 B0
97A0  DF 60 A9 00 8D 50 94 8D 51 94 18 AD 50 94 85 59
97B0  65 A4 85 14 AD 51 94 85 5A 65 A5 85 15 AD 4F 94
97C0  85 5B 20 05 96 AD 4C 94 85 59 AD 4D 94 85 5A 20
97D0  A3 96 A5 A6 2C 4E 94 30 05 18 65 5C 90 03 38 E5
97E0  5C AA 20 30 95 EE 50 94 D0 03 EE 51 94 AD 4D 94
97F0  CD 51 94 90 0A D0 B3 AD 4C 94 CD 50 94 B0 AB 60
9800  A9 00 8D 52 94 85 5B A5 A6 2C 4E 94 30 06 18 6D
9810  52 94 90 04 38 ED 52 94 48 AD 4C 94 85 59 AD 4D
9820  94 85 5A 20 05 96 AD 4F 94 85 59 A9 00 85 5A 20
9830  A3 96 18 A5 5C 65 A4 85 14 A5 5D 65 A5 85 15 68
9840  AA 20 30 95 EE 52 94 AD 52 94 CD 4F 94 F0 B6 90
9850  B4 60 20 EF 95 20 FD AE 20 EB B7 8E 55 94 A5 14
9860  8D 53 94 A5 15 8D 54 94 20 FD AE 20 EB B7 8E 58
9870  94 A6 14 8E 56 94 A5 15 8D 57 94 AD 53 94 85 A4
9880  AD 54 94 85 A5 AD 55 94 85 A6 AD 56 94 85 A7 AD
9890  57 94 85 A8 AD 55 94 85 A9 20 F7 96 AD 56 94 85
98A0  A4 AD 57 94 85 A5 AD 55 94 85 A6 AD 56 94 85 A7
98B0  AD 57 94 85 A8 AD 58 94 85 A9 20 F7 96 AD 56 94
98C0  85 A4 AD 57 94 85 A5 AD 58 94 85 A6 AD 53 94 85
```

```
98D0  A7 AD 54 94 85 A8 AD 58 94 85 A9 20 F7 96 AD 53
98E0  94 85 A4 AD 54 94 85 A5 AD 58 94 85 A6 AD 53 94
98F0  85 A7 AD 54 94 85 A8 AD 55 94 85 A9 20 F7 96 60
9900  20 EF 95 20 FD AE 20 EB B7 8E 55 94 A5 14 8D 53
9910  94 A5 15 8D 54 94 20 FD AE 20 EB B7 8E 58 94 A6
9920  14 8E 56 94 A5 15 8D 57 94 AD 58 94 CD 55 94 B0
9930  09 AE 55 94 8E 58 94 8D 55 94 AE 55 94 8E 52 94
9940  AD 53 94 85 A4 AD 54 94 85 A5 AD 52 94 85 A6 85
9950  A9 AD 56 94 85 A7 AD 57 94 85 A8 20 F7 96 EE 52
9960  94 AC 58 94 CC 52 94 B0 D7 60 20 EF 95 20 FD AE
9970  20 EB B7 8E 5B 94 A5 14 8D 59 94 A5 15 8D 5A 94
9980  20 FD AE 20 8A AD A0 94 A2 33 20 D4 BB 20 FD AE
9990  20 8A AD A0 94 A2 38 20 D4 BB A9 33 A0 94 20 50
99A0  B8 20 2B BC 30 0A A9 33 A0 94 20 A2 BB 4C B7 99
99B0  A9 38 A0 94 20 A2 BB A9 BC A0 B9 20 0F BB A2 42
99C0  A0 94 20 D4 BB A9 20 8D 3E 94 A9 00 8D 3D 94 8D
99D0  3F 94 8D 40 94 8D 41 94 A9 3D A0 94 20 A2 BB 20
99E0  6B E2 A9 33 A0 94 20 28 BA A9 11 A0 BF 20 67 B8
99F0  20 9B BC 18 A5 65 6D 59 94 85 14 A5 64 6D 5A 94
9A00  85 15 A9 3D A0 94 20 A2 BB 20 64 E2 A9 38 A0 94
9A10  20 28 BA A9 11 A0 BF 20 67 B8 20 9B BC 18 A5 65
9A20  6D 5B 94 AA 20 30 95 A9 3D A0 94 20 A2 BB A9 42
9A30  A0 94 20 67 B8 A2 3D A0 94 20 D4 BB A9 09 A0 E3
9A40  20 50 B8 20 2B BC 10 90 60 20 EF 95 20 FD AE 20
9A50  EB B7 86 A6 A5 14 85 A4 A5 15 85 A5 20 FD AE 20
9A60  8A AD A0 94 A2 33 20 D4 BB 20 FD AE 20 8A AD A0
9A70  94 A2 38 20 D4 BB 20 FD AE 20 8A AD A0 94 A2 47
9A80  20 D4 BB 20 6B E2 A9 33 A0 94 20 28 BA A9 11 A0
9A90  BF 20 67 B8 20 9B BC 18 A5 65 65 A4 85 A7 A5 64
9AA0  65 A5 85 A8 A9 47 A0 94 20 A2 BB 20 64 E2 A9 38
9AB0  A0 94 20 28 BA A9 11 A0 BF 20 67 B8 20 9B BC 18
9AC0  A5 65 65 A6 85 A9 20 F7 96 60 00 00 00 00 00 00
9AD0  00 20 EF 95 20 FD AE 20 EB B7 8E CC 9A A5 14 8D
9AE0  CA 9A A5 15 8D CB 9A 20 FD AE 20 8A AD 20 F7 B7
9AF0  78 A5 01 29 FE 85 01 A0 00 B1 14 0A 8D CD 9A C8
9B00  B1 14 2A 8D CE 9A A5 14 18 69 02 85 FD A9 00 65
9B10  15 85 FE A0 00 8C CF 9A A0 00 AD CE 9A CD CF 9A
9B20  F0 17 20 56 9B 8C D0 9A 20 30 95 AC D0 9A C8 D0
9B30  F1 EE CF 9A E6 FE 4C 18 9B CC CD 9A F0 10 20 56
9B40  9B 8C D0 9A 20 30 95 AC D0 9A C8 4C 39 9B A5 01
9B50  09 01 85 01 58 60 18 B1 FD 30 0F 6D CA 9A 85 14
9B60  A9 00 6D CB 9A 85 15 4C 76 9B 6D CA 9A 85 14 A9
9B70  FF 6D CB 9A 85 15 C8 18 B1 FD 6D CC 9A AA 60 20
9B80  9E B7 8E A7 9B 8A 4A 4A 4A F0 0A 48 20 3A 9C 68
9B90  38 E9 01 D0 F6 AD A7 9B 29 07 F0 0A 48 20 A8 9B
9BA0  68 38 E9 01 D0 F6 60 00 20 AA 9C A9 00 85 FD A9
9BB0  E0 85 FE A9 C7 85 14 A9 DE 85 15 A2 00 A0 01 98
9BC0  29 07 C9 00 F0 5F B1 FD 88 91 FD C8 C8 C0 C9 D0
9BD0  EE E8 A9 C8 18 65 FD 85 FD A9 00 65 FE 85 FE A9
9BE0  C8 18 65 14 85 14 A9 00 65 15 85 15 20 B4 9C 20
9BF0  AA 9C E0 28 D0 C7 A9 07 85 14 A9 FE 85 15 A2 00
```

```
9C00  A0 00 A9 00 91 14 98 18 69 08 A8 C0 A8 D0 F3 A5
9C10  14 18 69 A0 85 14 A5 15 69 00 85 15 E8 E0 02 D0
9C20  DF 20 B4 9C 60 A5 15 C9 DE F0 A1 C9 DF D0 04 C0
9C30  70 90 99 B1 FD 91 14 4C CC 9B 20 AA 9C A9 00 85
9C40  14 A9 E0 85 15 A9 40 85 FD A9 E1 85 FE A2 00 A0
9C50  00 B1 FD 91 14 C8 C0 C0 D0 F7 A5 FD 18 69 C0 85
9C60  FD A5 FE 69 00 85 FE A5 14 18 69 C0 85 14 A5 15
9C70  69 00 85 15 20 B4 9C 20 AA 9C E8 E0 28 D0 D0 A9
9C80  00 85 FD A9 FE 85 FE A2 00 A0 00 A9 00 91 FD C8
9C90  C0 28 D0 F9 A5 FD 18 69 28 85 FD A5 FE 69 00 85
9CA0  FE E8 E0 08 D0 E3 20 B4 9C 60 78 48 A5 01 29 FD
9CB0  85 01 68 60 48 A5 01 09 02 85 01 68 58 60 20 D4
9CC0  E1 A9 61 85 B9 A5 B7 D0 03 4C 10 F7 20 D5 F3 20
9CD0  8F F6 A5 BA 20 B1 FF A5 B9 20 93 FF A9 00 20 A8
9CE0  FF 85 AC A9 E0 20 A8 FF 85 AD 20 AA 9C A0 00 B1
9CF0  AC 20 B4 9C 20 A8 FF E6 AC D0 02 E6 AD A5 AD C9
9D00  FF 90 E7 A5 AC C9 40 90 E1 20 AE FF 4C 42 F6 4C
9D10  04 F7 20 D4 E1 A9 60 85 B9 A5 B7 D0 03 4C 10 F7
9D20  20 D5 F3 20 AF F5 A5 BA 20 B4 FF A5 B9 20 96 FF
9D30  20 A5 FF 85 AE 20 B7 FF D0 D5 20 A5 FF 85 AF 20
9D40  B7 FF D0 17 20 A5 FF 20 AA 9C A0 00 11 AE 91 AE
9D50  20 B4 9C E6 AE D0 E8 E6 AF D0 E4 4C 42 F6 20 D4
9D60  E1 A9 00 A2 00 A0 E0 4C D5 FF A9 32 A0 94 20 28
9D70  BA A9 11 A0 BF 20 67 B8 20 9B BC 18 A5 65 6D 48
9D80  94 8D 4B 94 20 0B 99 60 48 A5 01 09 07 85 01 68
9D90  58 6C FA FF A0 AF FF AF FF AF FF AF FF AF FF AF
9DA0  59 00 50 00 50 00 50 00 50 00 50 00 50 00 D0 00
9DB0  FF AF FF AF FF AF FF AF FF AF FF AF FF AF FF 29
9DC0  7B 04 FD 01 58 00 7E 04 F8 04 FE 08 F6 04 FF A4
9DD0  FB AB F9 2A 5B AB 7B 01 D0 82 51 81 D9 0B D9 A1
9DE0  5D 0D 7E 0F 75 04 7E 2D 7D 25 7F 2F 7E 27 FE AF
9DF0  50 AB D0 AB 59 AB 51 A9 D0 82 50 81 50 0B D0 A9
9E00  4C AD 9E 4C 45 9F 4C EE 9E 4C 6B 9F 00 00 00 00
9E10  A0 02 B1 47 85 23 88 B1 47 85 22 88 B1 47 85 AA
9E20  60 A0 02 A5 23 91 47 88 A5 22 91 47 88 A5 AA 91
9E30  47 60 A5 AB C5 AA F0 09 90 07 20 F4 B4 86 22 84
9E40  23 85 AA A8 F0 14 A5 01 48 29 FE 85 01 88 B9 00
9E50  A0 91 22 C0 00 D0 F6 68 85 01 60 AA 10 2A C9 FF
9E60  F0 26 24 0F 30 22 38 E9 7F AA A0 FF CA F0 08 C8
9E70  B9 9E A0 10 FA 30 F5 C8 B9 9E A0 30 09 A6 AB E6
9E80  AB 9D 00 A0 D0 F1 29 7F C9 22 D0 08 A5 0F 49 FF
9E90  85 0F A9 22 60 A2 00 20 79 00 C9 23 D0 0C 20 73
9EA0  00 20 9E B7 20 1E E1 20 FD AE 86 13 60 A9 00 85
9EB0  AB A9 08 20 B4 FF A9 6F 20 96 FF 20 A5 FF A4 AB
9EC0  99 00 A0 E6 AB 24 9D 10 03 20 D2 FF C9 0D D0 EB
9ED0  20 AB FF C6 AB A9 44 85 45 A9 D3 85 46 A9 FF 85
9EE0  0D 20 E7 B0 20 10 9E 20 32 9E 20 21 9E 60 A9 00
9EF0  85 0F 85 AB 20 95 9E 20 8B B0 20 10 9E 20 24 E1
9F00  8D 0E 9E 20 24 E1 8D 0F 9E A8 F0 27 20 24 E1 8D
9F10  0C 9E 20 24 E1 8D 0D 9E 20 24 E1 20 5B 9E A4 AB
9F20  99 00 A0 A8 F0 0D A5 90 D0 09 E6 AB D0 EA A2 17
```

```
9F30  4C 37 A4 20 32 9E 20 21 9E A6 13 F0 07 20 CC FF
9F40  A2 00 86 13 60 A9 00 85 AB 20 95 9E 20 8B B0 20
9F50  10 9E 20 12 E1 A4 AB 99 00 A0 C9 0D F0 D5 A5 90
9F60  D0 D1 E6 AB D0 EC A2 17 4C 37 A4 20 D4 E1 A9 60
9F70  85 B9 A5 B7 F0 73 20 D5 F3 A5 BA 20 B4 FF A5 B9
9F80  20 96 FF A0 03 84 61 20 A5 FF 85 62 A4 90 D0 59
9F90  20 A5 FF A4 90 D0 52 C6 61 D0 EC A6 62 20 CD BD
9FA0  20 3F AB A9 00 85 AB 85 0F 20 A5 FF A4 90 D0 39
9FB0  AA F0 0C 20 5B 9E A6 AB E6 AB 9D 00 A0 D0 EA A5
9FC0  01 48 29 FE 85 01 A0 00 84 61 E6 61 B9 00 A0 20
9FD0  D2 FF A4 61 C4 AB D0 F0 68 85 01 A9 0D 20 D2 FF
9FE0  A0 02 84 61 20 E1 FF D0 9E 20 42 F6 60 00 00 00
9FF0  AF AF AF AF AF AF AF AF AF AF AF AF 41 53 53 31
```

```
C000  A5 CB CD 2F C0 F0 24 C9 40 F0 20 A2 00 8E 07 D4
C010  8E 0B D4 A2 04 8E 0D D4 A2 8C 8E 08 D4 A2 10 8E
C020  0C D4 A2 11 8E 0B D4 CA 8E 0B D4 8D 2F C0 60 40
C030  A5 D4 D0 59 8D 8E C0 A6 CB CA CA CA E0 04 B0 48
C040  BD 90 C0 AE 8D 02 7D 94 C0 8D 8E C0 A5 CC D0 3D
C050  A5 CB CD 8F C0 F0 36 A5 D8 D0 2D A5 FE 48 A5 FD
C060  48 AE 8E C0 CA 8A 0A 0A 0A 0A 85 FD A9 C1 69 00
C070  85 FE A0 00 B1 FD F0 08 99 77 02 C8 C0 0F 90 F4
C080  84 C6 68 85 FD 68 85 FE A5 CB 8D 8F C0 60 00 40
C090  07 01 03 05 00 01 08 09 10 11 18 19 00 00 00 00
C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C100  4C 49 53 54 0D 00 00 00 00 00 00 00 00 00 00 00
C110  4F D0 34 2C 34 3A 43 CD 34 3A 4C C9 0D 00 00 00
C120  52 55 4E 0D 00 00 00 00 00 00 00 00 00 00 00 00
C130  50 D2 34 3A 43 4C 4F 53 45 34 0D 00 00 00 00 00
C140  43 4F 4E 54 0D 00 00 00 00 00 00 00 00 00 00 00
C150  3F 46 52 45 28 30 29 2B 36 35 35 33 38 0D 00 00
C160  08 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C170  08 8E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C180  4C 4F 41 44 22 22 14 00 00 00 00 00 00 00 00 00
C190  50 4F 4B 45 35 33 32 38 30 2C 00 00 00 00 00 00
C1A0  22 2C 38 0D 00 00 00 00 00 00 00 00 00 00 00 00
C1B0  50 4F 4B 45 35 33 32 38 31 2C 00 00 00 00 00 00
C1C0  53 41 56 45 22 22 14 00 00 00 00 00 00 00 00 00
C1D0  56 45 52 49 46 59 22 22 14 00 00 00 00 00 00 00
C1E0  53 41 56 45 22 22 14 40 3A 00 00 00 00 00 00 00
C1F0  56 45 52 49 46 59 0D 00 00 00 00 00 00 00 00 00
C200  53 59 53 35 31 32 30 30 0D 00 00 00 00 00 00 00
C210  53 59 53 34 39 31 35 32 0D 00 00 00 00 00 00 00
C220  44 49 53 4B 0D 00 00 00 00 00 00 00 00 00 00 00
C230  56 49 45 57 22 24 22 2C 38 0D 00 00 00 00 00 00
C240  4C 41 44 30 2C 22 45 32 22 2C 38 2C 31 0D 00 00
C250  50 4F 4B 45 24 43 38 30 39 2C 00 00 00 00 00 00
```

```
C260  46 32 33 00 00 00 00 00 00 00 00 00 00 00 00 00
C270  46 32 34 00 00 00 00 00 00 00 00 00 00 00 00 00
C280  46 32 35 00 00 00 00 00 00 00 00 00 00 00 00 00
C290  46 32 36 00 00 00 00 00 00 00 00 00 00 00 00 00
C2A0  46 32 37 00 00 00 00 00 00 00 00 00 00 00 00 00
C2B0  46 32 38 00 00 00 00 00 00 00 00 00 00 00 00 00
C2C0  46 32 39 00 00 00 00 00 00 00 00 00 00 00 00 00
C2D0  46 33 30 00 00 00 00 00 00 00 00 00 00 00 00 00
C2E0  46 33 31 00 00 00 00 00 00 00 00 00 00 00 00 00
C2F0  46 33 32 00 00 00 00 00 00 00 00 00 00 00 00 00
C300  50 43 50 58 43 50 59 44 45 43 44 45 58 44 45 59
C310  45 4F 52 49 4E 43 49 4E 58 49 4E 59 4A 4D 50 4A
C320  53 52 4C 44 41 4C 44 58 4C 44 59 4C 53 52 4E 4F
C330  50 4F 52 41 50 48 41 50 48 50 50 4C 41 50 4C 50
C340  52 4F 4C 52 4F 52 52 54 49 52 54 53 53 42 43 53
C350  45 43 53 45 44 53 45 49 53 54 41 53 54 58 53 54
C360  59 54 41 58 54 41 59 54 53 58 54 58 41 54 58 53
C370  54 59 41 57 4F 52 20 FD AE 20 9E AD 20 A3 B6 A0
C380  00 C9 03 D0 2A 84 A8 A9 39 85 A9 A0 00 A5 A9 C5
C390  A8 90 1C 18 65 A8 4A 85 AA 0A 65 AA AA A0 00 BD
C3A0  C8 C2 D1 22 D0 0A E8 C8 C0 03 D0 F3 A4 AA C8 60
C3B0  B0 07 A4 AA C8 84 A8 D0 D2 A4 AA F0 F2 88 84 A9
C3C0  4C 8B C3 A9 90 85 38 85 34 A2 00 8A 9D 00 9A E8
C3D0  D0 FA 8D 1B C0 8D 1C C0 60 AF FF AF FF AF FF AF
C3E0  50 00 50 00 50 00 50 00 00 00 50 00 50 00 50 50
C3F0  FF AF AF AF FF AF FF AF FF AF AF AF AF AF AF FF
C400  00 A9 FC D0 02 A9 FB 8D 00 C4 20 F1 AE 20 8D AD
C410  78 A5 14 48 A5 15 48 A5 01 48 20 F7 B7 A0 00 A5
C420  01 2D 00 C4 85 01 B1 14 A8 68 85 01 68 85 15 68
C430  85 14 20 A2 B3 58 60 20 F1 AE 20 8D AD 20 EA C5
C440  A5 62 85 65 A5 63 85 64 A0 00 B1 64 85 63 C8 B1
C450  64 85 62 20 E4 C5 60 20 8A AD 20 F7 B7 20 FD AE
C460  20 8A AD 20 EA C5 A0 00 A5 63 91 14 C8 A5 62 91
C470  14 60 20 F1 AE 20 8D AD 20 EA C5 A5 62 85 65 A5
C480  63 85 64 A0 00 A5 01 48 78 25 FC 85 01 B1 64 85
C490  63 C8 B1 64 85 62 68 85 01 58 20 E4 C5 60 20 EB
C4A0  B7 78 A5 01 48 29 F9 85 01 A0 00 8A 91 14 68 85
C4B0  01 58 60 78 A9 00 85 14 A9 D0 85 15 A5 01 48 29
C4C0  F9 85 01 A0 00 A2 10 B1 14 91 14 C8 D0 F9 E6 15
C4D0  CA D0 F4 68 85 01 58 60 00 00 00 00 A5 01 8D DB
C4E0  C4 08 78 29 FC 85 01 A0 00 8C DA C4 B1 C3 AA B1
C4F0  C1 91 C3 8A 91 C1 C8 D0 07 E6 C4 E6 C2 EE DA C4
C500  AD DA C4 CD D9 C4 90 E4 CC D8 C4 90 DF AD DB C4
C510  85 01 28 60 00 D0 00 D0 20 79 00 F0 40 20 8A AD
C520  20 F7 B7 A5 14 8D 14 C5 A5 15 8D 15 C5 20 79 00
C530  F0 2B 20 FD AE 20 8A AD 20 F7 B7 A5 14 8D 16 C5
C540  A5 15 8D 17 C5 20 79 00 F0 13 20 FD AE 20 8A AD
C550  20 F7 B7 A5 14 8D D8 C4 A5 15 8D D9 C4 AD 14 C5
C560  85 C3 AD 15 C5 85 C4 AD 16 C5 85 C1 AD 17 C5 85
C570  C2 20 DC C4 60 00 CC 20 8A AD 20 F7 B7 A5 14 8D
C580  75 C5 A5 15 8D 76 C5 20 FD AE 20 D4 E1 A9 00 AE
```

```
C590  75 C5 AC 76 C5 20 D5 FF B0 08 20 B7 FF 29 BF D0
C5A0  01 60 A2 1D 4C 37 A4 20 8A AD 20 F7 B7 A5 14 85
C5B0  C1 A5 15 85 C2 20 FD AE 20 8A AD 20 F7 B7 A5 14
C5C0  8D 75 C5 A5 15 8D 76 C5 20 FD AE 20 D4 E1 A5 01
C5D0  48 29 FE 85 01 A9 C1 AE 75 C5 AC 76 C5 20 D8 FF
C5E0  68 85 01 60 A2 90 38 4C 49 BC A5 14 48 A5 15 48
C5F0  20 F7 B7 85 62 84 63 68 85 15 68 85 14 60 20 AA
C600  B1 85 62 84 63 60 84 61 A0 00 84 62 84 63 C4 61
C610  F0 2A B1 22 38 E9 30 C9 0A 90 0A E9 07 C9 0A 90
C620  1B C9 10 B0 17 06 63 26 62 06 63 26 62 06 63 26
C630  62 06 63 26 62 05 63 85 63 C8 D0 D2 60 84 61 A0
C640  00 84 62 84 63 C4 61 F0 13 B1 22 C9 30 90 0D C9
C650  31 F0 02 B0 07 26 63 26 62 C8 D0 E9 60 A9 00 99
C660  00 01 88 A5 63 29 0F C9 0A 90 02 69 06 69 30 99
C670  00 01 46 62 66 63 46 62 66 63 46 62 66 63 46 62
C680  66 63 88 10 DE 60 A9 00 99 00 01 88 46 62 66 63
C690  A9 00 69 30 99 00 01 88 10 F2 60 20 73 00 A5 7A
C6A0  85 22 A5 7B 85 23 A0 04 20 06 C6 20 FB A8 A0 00
C6B0  84 0D 20 E4 C5 60 20 73 00 A5 7A 85 22 A5 7B 85
C6C0  23 A0 10 20 3D C6 20 FB A8 A0 00 84 0D 20 E4 C5
C6D0  60 20 73 00 20 F1 AE 20 82 B7 85 61 A0 00 B1 22
C6E0  C9 24 F0 13 C9 25 F0 18 A5 61 4C B0 B7 A4 61 88
C6F0  E6 22 D0 02 E6 23 60 20 ED C6 20 06 C6 4C E4 C5
C700  20 ED C6 20 3D C6 4C E4 C5 20 F1 AE 20 8D AD 20
C710  EA C5 60 20 09 C7 A0 24 84 FF A0 04 20 5D C6 A9
C720  FF A0 00 4C 87 B4 20 09 C7 A0 25 84 FF A0 10 20
C730  86 C6 A9 FF A0 00 4C 87 B4 20 FA AE 20 9E AD 20
C740  A3 B6 85 A9 A5 22 85 A7 A5 23 85 A8 20 FD AE 20
C750  9E AD 20 A3 B6 85 AA 20 F7 AE A2 01 38 A5 AA E5
C760  A9 90 1A 69 00 85 61 A4 A9 88 30 13 B1 A7 D1 22
C770  F0 F7 E6 22 D0 02 E6 23 E8 C6 61 D0 EA A2 00 8A
C780  A8 20 A2 B3 60 60 87 48 00 00 00 A9 7A 8D 04 DD
C790  A9 26 8D 05 DD 20 8A AD A9 86 A0 C7 20 28 BA 20
C7A0  F7 B7 A5 14 8D 06 DD A5 15 8D 07 DD AD 0E DD 29
C7B0  D1 09 11 8D 0E DD AD 0F DD 29 D1 09 59 8D 0F DD
C7C0  AD 0D DD 29 02 D0 05 20 E1 FF D0 F4 60 00 00 00
C7D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C800  4C 21 C8 4C 5B C8 4C B8 CA 00 0C 90 00 C0 30 C0
C810  20 C8 20 C8 20 C8 20 C8 20 C8 20 C8 00 24 0A 49
C820  60 A9 80 8D 08 03 A9 C8 8D 09 03 A9 FC 8D 0A 03
C830  A9 C9 8D 0B 03 A9 8E 85 38 85 34 78 A9 B3 8D 14
C840  03 A9 CA 8D 15 03 58 A9 4F A0 C8 20 1E AB 60 50
C850  52 4F 46 49 20 56 31 2E 30 0D 00 A9 E4 8D 08 03
C860  A9 A7 8D 09 03 A9 86 8D 0A 03 A9 AE 8D 0B 03 78
C870  A9 31 8D 14 03 A9 EA 8D 15 03 58 A9 A0 85 38 60
C880  20 73 00 90 1E C9 60 B0 1A C9 41 90 15 8D 1F C8
C890  A2 00 8E 1D C8 A0 00 EE 1D C8 BD 24 C9 D0 07 AD
C8A0  1F C8 38 4C E7 A7 D1 7A D0 28 C8 E8 BD 24 C9 D0
C8B0  F5 18 98 65 7A 85 7A 90 02 E6 7B AD 1D C8 0A AA
```

```
C8C0  BD DC C8 8D CD C8 BD DD C8 8D CE C8 20 A7 C5 4C
C8D0  AE A7 E8 BD 24 C9 D0 FA E8 4C 95 C8 E7 A7 5B C8
C8E0  57 C4 9E C4 B3 C4 18 C5 77 C5 A7 C5 8B C7 00 9E
C8F0  03 9E 06 9E 09 9E 00 CB 00 8E 00 94 03 94 06 94
C900  09 94 0C 94 0F 94 12 94 15 94 18 94 1B 94 1E 94
C910  21 94 24 94 27 94 2A 94 2D 94 30 94 06 90 03 90
C920  00 90 09 90 41 55 53 00 44 4F 4B 45 00 52 4F 4B
C930  45 00 43 48 41 52 4B 4F 50 00 53 57 41 50 00 4C
C940  41 44 00 53 50 45 49 00 50 41 55 53 45 00 44 49
C950  53 4B 00 4C 45 53 54 00 4C 45 53 50 00 56 49 45
C960  57 00 53 45 54 54 49 4D 45 00 54 45 53 54 00 47
C970  52 45 49 4E 00 47 52 41 55 53 00 47 52 4C 4F 45
C980  00 46 41 52 42 45 00 47 52 41 46 45 49 4E 00 50
C990  55 4E 4B 54 00 43 48 41 52 00 4C 49 4E 49 45 00
C9A0  52 45 43 48 54 00 42 4C 4F 43 4B 00 4B 52 45 49
C9B0  53 00 52 41 44 49 55 53 00 5A 45 49 43 48 45 4E
C9C0  00 53 43 52 4F 4C 4C 00 47 52 53 50 45 49 00 47
C9D0  52 4C 41 44 00 47 52 4D 49 53 43 48 00 46 41 48
C9E0  52 54 41 55 53 00 47 52 45 4E 5A 45 4E 00 46 41
C9F0  48 52 54 00 53 50 52 49 54 45 00 00 A9 00 85 0D
CA00  20 73 00 90 2A C9 C5 F0 5C C9 60 B0 22 C9 24 F0
CA10  57 C9 25 F0 56 C9 41 90 15 8D 1F C8 A2 00 8E 1E
CA20  C8 A0 00 EE 1E C8 BD 82 CA D0 07 AD 1F C8 38 4C
CA30  8D AE D1 7A D0 25 C8 E8 BD 82 CA D0 F5 18 98 65
CA40  7A 85 7A 90 02 E6 7B AD 1E C8 0A AA BD 6E CA 8D
CA50  59 CA BD 6F CA 8D 5A CA 4C 13 C7 E8 BD 82 CA D0
CA60  FA E8 4C 21 CA 4C D1 C6 4C 9B C6 4C B6 C6 8D AE
CA70  13 C7 26 C7 37 C4 01 C4 72 C4 39 C7 05 C4 8B CB
CA80  00 8E 48 45 58 24 00 42 49 4E 24 00 44 45 45 4B
CA90  00 52 45 45 4B 00 44 52 45 45 4B 00 49 4E 44 45
CAA0  58 00 43 48 45 45 4B 00 54 49 4D 45 00 54 45 53
CAB0  54 00 00 A9 01 8D 1C C8 AD 1C C8 2D 09 C8 F0 03
CAC0  20 CB CA 0E 1C C8 90 F0 4C 31 EA A2 FE E8 E8 4A
CAD0  90 FB BD 0B C8 8D 1B C8 BD 0A C8 8D 1A C8 6C 1A
CAE0  C8 00 00 00 50 00 00 40 00 00 50 00 50 00 00 50
CAF0  FF AF FF AF FF AF FF AF AF FF AF AF AF FF AF FF
CB00  20 9E B7 CA 86 AB 20 FD AE 20 9E AD 20 A3 B6 C9
CB10  0A F0 03 4C 48 B2 A5 AB 4A 48 A9 00 69 DC 85 A8
CB20  A2 00 86 A7 A0 0E B1 A7 09 80 91 A7 C8 68 4A 08
CB30  B1 A7 2A 28 6A 91 A7 20 65 CB F8 C9 12 F0 06 90
CB40  04 E9 12 09 80 D8 A0 0B 91 A7 20 65 CB 88 91 A7
CB50  20 65 CB 88 91 A7 A1 22 29 0F 88 91 A7 60 E6 22
CB60  D0 02 E6 23 60 A1 22 0A 0A 0A 0A 85 A9 20 5E CB
CB70  A1 22 29 0F 05 A9 85 A9 20 5E CB A1 22 C9 3A F0
CB80  05 68 68 4C 48 B2 A5 A9 4C 5E CB A2 01 20 79 00
CB90  B0 03 20 9E B7 8A 18 69 DB 85 A8 A9 00 85 A7 85
CBA0  AB A9 0A 20 F4 B4 85 61 86 62 84 63 A0 0B B1 A7
CBB0  F8 08 29 7F C9 12 D0 02 A9 00 28 10 03 18 69 12
CBC0  D8 20 EB CB A0 0A B1 A7 20 EB CB A0 09 B1 A7 20
CBD0  EB CB A0 08 B1 A7 20 E0 CB 4C CA B4 4A 4A 4A 4A
CBE0  29 0F 09 30 A4 AB 91 62 E6 AB 60 48 20 DC CB 68
CBF0  20 E0 CB A9 3A 20 E4 CB 60 8A AF AF AF AF AF FF
```

Anhang 5 : Interpretercodes und Abkürzungen

ABS	A	SHIFT/B	A	SIN	S	SHIFT/I	S
AND	A	SHIFT/N	A	SPC(	S	SHIFT/P	S
ASC	A	SHIFT/S	A	STEP	ST	SHIFT/E	ST
ATN	A	SHIFT/T	A	STOP	S	SHIFT/T	S
CHR$	C	SHIFT/H	C	STR$	ST	SHIFT/R	ST
CLOSE	CL	SHIFT/O	CL	SQR	S	SHIFT/Q	S
CLR	C	SHIFT/L	C	SYS	S	SHIFT/Y	S
CMD	C	SHIFT/M	C	TAB(	T	SHIFT/A	T
CONT	C	SHIFT/O	C	THEN	T	SHIFT/H	T
DATA	D	SHIFT/A	D	USR	U	SHIFT/S	U
DEF	D	SHIFT/E	D	VAL	V	SHIFT/A	V
DIM	D	SHIFT/I	D	VERIFY	V	SHIFT/E	V
END	E	SHIFT/N	E	WAIT	W	SHIFT/A	W
EXP	E	SHIFT/X	E				
FOR	F	SHIFT/O	F				
FRE	F	SHIFT/R	F				
GET	G	SHIFT/E	G				
GOSUB	GO	SHIFT/S	GO				
GOTO	G	SHIFT/O	G				
INPUT#	I	SHIFT/N	I				
LEFT$	LE	SHIFT/F	LE				
LET	L	SHIFT/E	L				
LIST	L	SHIFT/I	L				
LOAD	L	SHIFT/O	L				
MID$	M	SHIFT/I	M				
NEXT	N	SHIFT/E	N				
NOT	N	SHIFT/O	N				
OPEN	O	SHIFT/P	O				
PEEK	P	SHIFT/E	P				
POKE	P	SHIFT/O	P				
PRINT	?		?				
PRINT#	P	SHIFT/R	P				
READ	R	SHIFT/E	R				
RESTORE	RE	SHIFT/S	RE				
RIGHT$	R	SHIFT/I	R				
RND	R	SHIFT/N	R				
RETURN	RE	SHIFT/T	RE				
RUN	R	SHIFT/U	R				
SAVE	S	SHIFT/A	S				
SGN	S	SHIFT/G	S				

Anhang 6 : Stichwortverzeichnis

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Checksummer 64

Um Ihnen das Abtippen der Programme zu erleichtern, haben wir dem 64er-Heft einen Checksummer entnommen, den wir der Vollständigkeit halber an dieser kurz vorstellen wollen:

Der Checksummer überprüft jede eingegebene Basic-Zeile und erspart Ihnen deshalb eine langwierige Fehlersuche. Der Checksummer 64 ist ein kleines Maschinenprogramm, das, wenn es aktiviert ist, Sie sofort davon unterrichtet, ob Sie die jeweilige Programmzeile korrekt eingegeben haben.

1. Tippen Sie den Basic-Lader sorgfältig ein.

2. Vergessen Sie nicht das Abspeichern vor dem Start (RUN)!

3. Wenn Sie den Checksummer 64 zwischenzeitlich nicht benutzen, können Sie ihn jederzeit mit

   POKE 1,55

   desaktivieren. Auch durch Drücken der RUN/STOP- und der RESTORE-Taste wird der Checksummer 64 desaktiviert.

   Wollen Sie den Checksummer 64 wieder einschalten, so geben Sie

   POKE 1,53

   ein.

Das Maschinenprogramm bleibt solange erhalten, bis der Computer ausgeschaltet, oder, wenn von anderen Programmen auf das hinter dem ROM liegende RAM zugegriffen wird. Alle Programme in diesem Buch sind mit einer Checksumme versehen, die am Ende jeder Programmzeile steht. Diese Checksumme steht zwischen 'Kleinerzeichen' und 'Größerzeichen'. Sie wird beim Eintippen des Programms nicht mit eingege-

ben. Die Zahl zwischen den beiden Zeichen stellt lediglich eine Information für Sie dar. Wenn Sie diese Checksumme dennoch mit eintippen, werden Sie schnell bemerken, daß Sie etwas falsch gemacht haben. Bei aktiviertem Checksummer 64 wird nämlich nach Eingabe einer Basic-Zeile, die mit Return beendet wird, in die linke obere Bildschirmecke die Checksumme eingeblendet, die mit der Summe aus dem veröffentlichen Listing übereinstimmen muß. Ist das nicht der Fall, haben Sie die Zeile anders eingegeben, als sie im Listing dargestellt ist.

Der Checksummer 64 ist so ausgelegt, daß er abhängig von der Zeilennummer und dem Text der Zeile eine Checksumme ausgibt. Beim Bilden dieser Checksumme werden Spaces (Leertaste) überlesen, was für Sie bedeutet, daß es egal ist, wieviel Leerzeichen Sie zwischen den Worten lassen, da Sie für den Programmablauf ohnehin keine Bedeutung haben.

Hinweise zum Lesen von Listings

Die Listings haben sich ein wenig im Ausdruckformat verändert, um Ihnen das Eingeben von Programmen wesentlich zu erleichtern.

- Cursorsteuerzeichen und andere Steuerzeichen, die schwer zu lesen sind, werden von nun an in Klartext in speziellen geschweiften Klammern gesetzt.

  Tritt mehrmals hintereinander dasselbe Steuerzeichen auf, so wird diese Steuerzeichen-Sequenz zusammengefaßt, indem zuerst die Anzahl der Wiederholungen dieses Steuerzeichens und dann das Steuerzeichen in Klartext ausgegeben wird.

- Alle Commodore-Grafikzeichen, die über Shift zu erreichen sind, werden nicht mehr als Grafikzeichen, sondern als Klartextzeichen dargestellt. Dabei wird aus dem Zeichen, das Sie auf dem Bildschirm sehen, wenn Sie die Tastenkombination Shift/A ansprechen, wieder ein 'A' (nur im abgedruckten Listing). Um dieses 'A' vom normalen 'A' unterscheiden zu können, ist es etwas kleiner als das gewöhnliche 'A' und ist außerdem mit einem Unter-

streichungszeichen versehen. Diese Vereinbarung gilt auch für sämtliche andere Commodore-Grafikzeichen, die über die SHIFT-Taste zu erreichen sind.

- Entsprechendes gilt für sämtliche Commodore-Grafikzeichen, die über die Commodore-Taste zu erreichen sind. Hier wird jedoch das jeweilige Klartextzeichen nicht unterstrichen, sondern überstrichen.

```
10 REM ********************************     <175>
20 REM *                              *     <247>
30 REM *       CHECKSUMMER 64         *     <162>
33 REM *                              *     <004>
36 REM *       (VERSION  2.0)         *     <014>
40 REM *                              *     <011>
50 REM *           64'ER              *     <061>
60 REM *                              *     <031>
70 REM *        COMMODORE 64          *     <056>
80 REM *                              *     <051>
90 REM ********************************     <255>
100 PRINT"{CLR,13SPACE,RVSON}CHECKSUMMER 6
    4{RVOFF}"                                <025>
110 PRINT                                    <007>
121 SA=820:FOR I=SA TO SA+6:READ A:POKE I,
    A:NEXT I                                 <073>
122 DATA 133,95,134,96,76,191,163            <179>
130 POKE 88,0:POKE 89,192:POKE 90,0:POKE 9
    1,192:POKE 780,0:POKE 781,160:SYS SA     <244>
140 POKE 88,0:POKE 89,0:POKE 90,0:POKE 91,
    0:POKE 780,0:POKE 781,224:SYS SA         <039>
150 POKE 1,53:POKE 42289,96:POKE 42290,228   <001>
160 FOR I=58464 TO 58554:READ A:POKE I,A:N
    EXT I                                    <100>
190 PRINT"{4DOWN,9SPACE}CHECKSUMMER AKTIVI
    ERT."                                    <247>
200 PRINT"{2DOWN}AUSSCHALTEN : POKE1,55"     <050>
210 PRINT"{DOWN}ANSCHALTEN{2SPACE}: POKE1,
    53":NEW                                  <171>
320 DATA 160,2,169,0,133,2,177,95            <103>
330 DATA 240,15,201,32,208,3,200,208         <239>
340 DATA 245,24,101,2,133,2,76,110           <153>
350 DATA 228,192,4,48,241,198,214,165        <090>
360 DATA 214,72,162,3,169,32,157,1           <191>
370 DATA 4,189,183,228,32,210,255,202        <096>
380 DATA 16,242,166,2,169,0,32,205           <206>
390 DATA 189,169,62,32,210,255,104,133       <168>
400 DATA 214,32,108,229,169,141,32,210       <168>
410 DATA 255,76,128,164,92,72,32,201         <093>
420 DATA 255,170,104,144,1,138,96,9          <051>
430 DATA 60,18,19                            <195>
```

Weitere Fachbücher aus unserem Verlagsprogramm

COMMODORE 64

Einführungskurs: Commodore 64
Mai 1984, 276 Seiten
Die Programmiersprache Basic · Einsatzgebiete des Commodore 64-Basic: Grafik, Musik, Dateiverwaltung · mit vielen Beispielprogrammen, häufig benötigten Tabellen und nützlichen Tips · für Einsteiger und Fortgeschrittene.
Best.-Nr. MT 685, ISBN: 3-89090-017-8
(Sfr. 35,—/öS 296,40) **DM 38,—**

Commodore 64 — leicht verständlich
Juni 1984, 154 Seiten
Informationen für den Computer-Neuling · Installation und Inbetriebnahme · Programmieren in Basic · Grafik und Töne · Auswahl von Hardware und Zubehör · Software für Ihren Computer · die ideale Einführung in das Arbeiten mit Ihrem Commodore 64.
Best.-Nr. MT 700, ISBN: 3-89090-022-4
(Sfr. 27,50/öS 232,40) **DM 29,80**

Ihr Heimcomputer Commodore 64
August 1984, 296 Seiten
Alles Wissenswerte im Umgang mit dem Commodore 64 · Planung, Kauf und Inbetriebnahme der Anlage · Einsatz fertig gekaufter oder selbst erstellter Programme · Schwächen und Stärken der altbewährten und neuesten Programmiersprachen · die gängigsten Software-Angebote für jeden Einsteiger.
Best.-Nr. MT 701, ISBN: 3-89090-044-5
(Sfr. 35,—/öS 296,40) **DM 38,—**

Das Commodore 64-LOGO-Arbeitsbuch
September 1984, 225 Seiten
Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.
Best.-Nr. MT 720, ISBN: 3-89090-063-1
(Sfr. 31,30/öS 265,20) **DM 34,—**

35 ausgesuchte Spiele für Ihren Commodore 64
September 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**

Lehrspielzeug Computer: C 64/VC-20
Juli 1984, 139 Seiten
Speziell für Kinder entwickelt führt dieses Buch spielerisch in die Basic-Welt des Commodore 64/VC-20 ein · mit vielen lehrreichen Spielprogrammen und Grafikmöglichkeiten · kleinere Kinder benötigen die Hilfe ihrer sachkundigen Eltern.
Best.-Nr. MT 695, ISBN: 3-89090-011-9
(Sfr. 23,—/öS 193,40) **DM 24,80**

Basic mit dem Commodore 64
April 1984, 320 Seiten
Ein Basic-Lehrbuch für den jugendlichen Anfänger · übersichtlich gegliederte Lernprogramme · Alles über INPUT-GOTO · Let-Befehle · Editorfunktionen · POKE-Befehle für die Grafik · geeignet auch als Leitfaden für Lehrer und Eltern.
Best.-Nr. MT 657, ISBN: 3-922120-91-1
(Sfr. 44,20/öS 374,40) **DM 48,—**

Computerspiele & Wissenswertes
Februar 1984, 156 Seiten
Eine Sammlung von interessanten und nützlichen Maschinenprogrammen · schnelle binäre Arithmetik · Basic-Erweiterungen · mit unterstützendem Assembler-Listing · für den fortgeschrittenen Programmierer.
Best.-Nr. MT 601, ISBN: 3-922120-62-8
(Sfr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 602 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das große Spielebuch — Commodore 64
Februar 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.**

Spiele für den Commodore 64
November 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 792, 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.**

Computer für Kinder — Ausgabe Commodore 64
1984, 112 Seiten
Ein Buch für Kinder und ihre Lehrer · ideal für die erste Begegnung mit Computern, ihren Eigenwilligkeiten und ihren unerschöpflichen Möglichkeiten · leichtverständliche Erläuterungen rund um den Commodore 64 · alle Programmbeispiele in BASIC.
Best.-Nr. PW 709, ISBN: 3-921803-41-1
(Sfr. 27,50/öS 232,40) **DM 29,80**

Grafik & Musik auf dem Commodore 64
Oktober 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,—**

Mehr als 32 Basic-Programme für den Commodore 64
März 1984, 279 Seiten
Programme speziell für den Commodore 64 · umfassende praktische Anwendungen · jede Menge Lehr- und Lernhilfen · super Spiele · für Basic-Neulinge und Experten.
Best.-Nr. MT 613, ISBN: 3-922120-66-0
(Sfr. 45,10/öS 382,20) **DM 49,—**
Best.-Nr. MT 614 (Beispiele auf Diskette)
(Sfr. 48,—/öS 432,—) **DM 48,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag Aktiengesellschaft Buchverlag, Hans-Pinsel-Str. 2, 8013 Haar

Weitere Fachbücher aus unserem Verlagsprogramm

Commodore 64 Listings — Band 1: Spiele
Oktober 1984, 199 Seiten
Mit ausführlicher Dokumentation · Spielanleitung · Variablen für die Änderung der Spiele · vollständige Listings für: Bürger Joe · Nibbler · Zingel Zangel · Universe · Würfelpoker · Maze-Mission · der magische Kreis · Todeskommando Atlantik · Enterprise.
Best.-Nr. MT 748, ISBN: 3-89090-068-2
(Sfr. 23,—/öS 193,40) **DM 24,80**
Best.-Nr. MT 804 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Commodore 64 Listings
Band 2: Dateiverwaltung · Schule · Hobby
Oktober 1984, 179 Seiten
Ein Buch mit Programmen für die ganze Familie · DATAVE — Eine Dateiverwaltung · mathematische Funktionen · Konjugation und Deklination in Latein · Regressionsanalyse · Bundesligatabelle.
Best.-Nr. MT 766, ISBN: 3-89090-071-2
(Sfr. 23,—/öS 193,40) **DM 24,80**

Der sensible Commodore 64
Januar 1985, ca. 130 Seiten
Eine Softwaresammlung zu den technologischen Neuerscheinungen im Commodore 64 · für Erstbenutzer wie für Experten ein Buch zur optimalen Softwarenutzung.
Best.-Nr. PW 727, ISBN: 3-921803-45-4
(Sfr. 27.50/öS 232,40) **DM 29,80**

Die Floppy 1541 April 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 806, ISBN 3-89090-098-4
(Sfr. 45,10/öS 382,20) **DM 49,—**
Best.-Nr. MT 710 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Commodore 64 — Multiplan
April 1984, 230 Seiten
Multiplan jetzt auch für den Commodore 64 · der volle Leistungsumfang der 16-Bit-Version · Einführung in die Arbeitsweise von Tabellenkalkulationsprogrammen · praxisnahe Beispiele · Beschreibung alle Befehle und Funktionen · nicht nur für Anfänger.
Best.-Nr. MT 655, ISBN: 3-922120-89-X
(Sfr. 44,20/öS 374,40) **DM 48,—**

Das Commodore 64-Buch, Bd. 1: Ein Leitfaden für Erstanwender
Mai 1984, 270 Seiten
Der Commodore 64 und seine Handhabung · Einführung in die Grafik · Balkendiagramme · Einführung in die Spritetechnik · Basic-Erweiterungen in Assembler · Ein Leitfaden für Erstanwender, die sich bereits BASIC-Kenntnisse angeeignet haben. Alle Beispiele auf Diskette erhältlich!
Best.-Nr. MT 591, ISBN: 3-922120-61-X
(Sfr. 44,20/öS 374,40) **DM 48,—**
Best.-Nr. MT 592 (Beispiele auf Diskette)
(Sfr. 58,—/öS 522,—) **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 2: Basic-Spiele
Mai 1984, 181 Seiten
Spiele nicht nur zum Abtippen · Programmlisting · Programmbeschreibung · Variablenübersicht · Programme nach Anleitung frei ergänzbar · das ideale Buch, um Programmieren spielend zu lernen · für Anfänger.
Best.-Nr. MT 593, ISBN: 3-922120-68-7
(Sfr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 594 (Beispiele auf Diskette)
(Sfr. 58,—/öS 522,—) **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 3: Ein Leitfaden für Fortgeschrittene
Februar 1984, 206 Seiten
Alles über Sprites · Wissenswertes über Multi-Color-Grafik · Assembler/Disassembler · jede Menge Basic-Erweiterungen · Umgang mit dem Soundgenerator · ein Leitfaden für Fortgeschrittene.
Best.-Nr. MT 595, ISBN: 3-922120-69-5
(Sfr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 596 (Beispiele auf Diskette)
(Sfr. 58,—/öS 522,—) **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 4: Ein Leitfaden für Systemprogrammierer
März 1984, 261 Seiten
Einführung in Maschinenprogrammierung · Verknüpfung von Maschinenprogrammen mit Basic-Programmen · alles über Assembler/Disassembler · Basic-Kenntnisse werden vorausgesetzt!
Best.-Nr. MT 597, ISBN: 3-922120-70-9
(Sfr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 598 (Beispiele auf Diskette)
(Sfr. 58,—/öS 522,—) **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 5: Ein Leitfaden durch Simon's Basic
Juli 1984, 322 Seiten
Ausführliche Besprechung aller Befehle · viele erklärende Beispiele · mit kommentierter Assembler-Listing · das richtige Nachschlagewerk für den geübten Commodore 64-Benutzer.
Best.-Nr. MT 599, ISBN: 3-922120-71-7
(Sfr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 600 (Beispiele auf Diskette)
(Sfr. 58,—/öS 522,—) **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 6: Spiele
Mai 1984, 190 Seiten
Programmieren auf dem Commodore 64 spielend gelernt · leicht verständliche Spielanleitungen · Programmlisting mit anschließender Programmbeschreibung · Variablenübersicht · Tips zum Ändern und Ergänzen des Programms.
Best.-Nr. MT 619, ISBN: 89090-072-5
(Sfr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 620 (Beispiele auf Diskette)
(Sfr. 58,—/öS 522,—) **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 7: Ein Leitfaden für Profis
August 1984, 210 Seiten
Der Commodore 64 als Klaviatur · Noten schreiben mit hochauflösender Grafik · relative Dateien am Beispiel einer kleinen Adreßverwaltung · Joystick und Paddles · Grafikspeicher unter Kernal · Interrupt-Manager · für Profis, die die letzten Möglichkeiten ihres Commodore 64 ausreizen wollen.
Best.-Nr. MT 731, ISBN: 3-89090-067-4
(Sfr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 784 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag Aktiengesellschaft Buchverlag, Hans-Pinsel-Str. 2, 8013 Haar

Weitere Fachbücher aus unserem Verlagsprogramm

ATARI

Spiel und Spaß mit dem Atari
Mai 1984, 338 Seiten
Einfache Programme in Basic · wie man ein Spiel entwickelt · Lernstoff trainieren · Zahlen und Logik · Grafik · Farben · Töne und Musik · den Atari-Computer spielend erforschen.
Best.-Nr. MT 672, ISBN: 3-89090-002-X
(Sfr. 38,60/öS 327,60) **DM 42,—**

Lehrspielzeug Computer: Atari
Juli 1984, 139 Seiten
Das neue Computer-Kinderbuch für den Atari 400, 800 und 1200 · Spielprogramme und grafische Darstellungen für Kinder ab 8 Jahren · viele Rechenaufgaben für den kleinen Einstein · so macht Lernen Freude!
Best.-Nr. MT 696, ISBN: 3-89090-012-5
(Sfr. 23,—/öS 193,40) **DM 24,80**

Spiele für den Atari
September 1984, 216 Seiten
Eine unterhaltsame Einweisung in die Atari-BASIC-Programmierung anhand von bereits bewährten sowie raffinierten neuen Computerspielen · wie man ein Programm strukturiert · Einsatz von Unterprogrammen · Tabellenverarbeitung · bewegte Grafiken · Testen von Programmen · noch nie hat Home-Computing so viel Spaß gemacht!
Best.-Nr. MT 678, ISBN: 3-89090-051-8
(Sfr. 29,50/öS 249,60) **DM 32,—**

Das Atari-Buch, Band 1
Juli 1984, 158 Seiten
Die grundlegenden Programmiermöglichkeiten für Ihren Atari · mit einem Spiel zum Eingewöhnen · Erstellung von Text und Grafik · Player Missiles · Basic-Besonderheiten · ausführliche Assemblerlistings im Anhang · ein Einsteiger-Buch, vollgepackt mit Informationen.
Best.-Nr. MT 703, ISBN: 3-89090-039-9
(Sfr. 29,50/öS 249,60) **DM 32,—**
Best.-Nr. MT 783 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Atari-Buch, Band 2
Oktober 1984, 197 Seiten
Spezielle Programmiermöglichkeiten und Maschinenprogramme · Basic-Kenntnisse und das Studium des Handbuchs (Das Atari-Buch, Bd. 1) werden vorausgesetzt · für alle, die die hervorragenden Grafik- und Soundeigenschaften des Atari ausnutzen wollen!
Best.-Nr. MT 704, ISBN: 3-89090-072-0
(Sfr. 29,50/öS 249,60) **DM 32,—**
Best.-Nr. MT 775 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Atari-Abenteuerspiele
Juli 1984, 148 Seiten
Wege zur Entwicklung und Ausführung spannender Spielprogramme: Schatzsuche · Kampf mit Monstern · Das Auge des Sternenkriegers.
Best.-Nr. MT 727, ISBN: 3-89090-035-6
(Sfr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 728 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Strategische Computerspiele für Ihren Atari
Mai 1984, 148 Seiten
Aufbau eines Spielfeldes · der Bewegungsablauf · Musteröffnungen · das Endspiel · Dame, Schach, Warp Trog als Beispiele strategischer Spiele · Anleitung zur systematischen Fehlersuche · für Fortgeschrittene.
Best.-Nr. MT 681, ISBN: 3-89090-004-6
(Sfr. 29,50/öS 249,60) **DM 32,—**
Best.-Nr. MT 682 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Mein Atari-Computer
1983, ca. 400 Seiten
Alles über Aufbau und Bedienung des Atari-Computers · Programmieren in Basic · Grafikfunktionen · Tonerzeugung · abgeleitete Trigonometrische Funktionen · Tabellen zur Zahlenumwandlung · das Standardwerk für Anfänger.
Best.-Nr. PW 554, ISBN: 3-921803-18-7
(Sfr. 54,30/öS 460,20) **DM 59,—**

Sprühende Ideen mit Atari-Grafik
Januar 1985, ca. 250 Seiten
Eine Einführung in die Grafikmöglichkeiten des Atari · die Gestaltgesetze von Objekten, Farbgebung, Bildschirmentwürfe · Basic-Kenntnisse erforderlich.
Best.-Nr. PW 716, ISBN: 3-921803-39-X
(Sfr. 45,10/öS 382,20) **DM 49,—**

Computer für Kinder — Ausgabe ATARI
Februar 1985, 114 Seiten
Ein BASIC-Programmierbuch ausdrücklich für Kinder geschrieben · mit einem besonderen Abschnitt für Lehrer und Eltern.
Best.-Nr. PW 728, ISBN: 3-921803-43-8
(Sfr. 27,50/öS 232,40) **DM 29,80**

Der Atari als Musikbox
November 1984, 196 Seiten
Eine musikalische Einführung in die Computerprogrammierung · was Sie über Resonanz und Harmonie wissen müssen · Musikprogramme in BASIC für zwei, drei und vier Stimmen sowie für einen Kanon · besondere Geräuscheffekte · eine Lieder-Bibliothek · für Anfänger.
Best.-Nr. MT 797, ISBN: 3-89090-075-5
(Sfr. 27,50/öS 232,40) **DM 29,80**

Lerne Basic auf dem Atari
November 1984, 321 Seiten
Dieses Buch führt sowohl Kinder als auch Erwachsene in die Grundlagen des Atari-Basic ein · Action-Spiele · Brettspiele · Wortspiele · Hinweise · Erklärungen · Übungen · amüsant und leicht verständlich präsentiert · zum Selbststudium geeignet.
Best.-Nr. MT 692, ISBN: 3-89090-007-0
(Sfr. 35,—/öS 296,40) **DM 38,—**

Ausgesuchte Atari-Programme mit Listings
Oktober 1984, 171 Seiten
Mehr als 25 Programme — vom alltäglichen Kleinkram bis zu geschäftlichen Anwendungen und Dienstprogrammen · Girokontoführung · Adressenverzeichnis · Joggingkontrolle · für Anfänger, die den Umgang mit dem Computer und die Grundbegriffe des Programmierens lernen wollen.
Best.-Nr. MT 759, ISBN: 3-89090-070-4
(Sfr. 29,50/öS 249,60) **DM 32,—**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag Aktiengesellschaft, Buchverlag, Hans-Pinsel-Str. 2, 8013 Haar

Weitere Fachbücher aus unserem Verlagsprogramm

Das Atari-Programmierhandbuch
März 1985, 403 Seiten
Alles was Sie über die Bedienung und die Programmierung Ihres Computers in BASIC wissen müssen · Speicherarten · grafische Symbole · spezielle Funktionen · Zubehörteile · Organisation eines Programms einschließlich Flußdiagramm und ihr Gebrauch · der 6502-Prozessor · mit vielen Programmierbeispielen für den ATARI 800 (400/600) · ein unentbehrliches Buch für die richtige Kaufentscheidung!
Best.-Nr. MT 753, ISBN 3-89090-062-3
(Sfr. 47,80/öS 405,60) **DM 52,—**

SCHNEIDER CPC 464

Der CPC 464 für Ein- und Umsteiger
Februar 1985, ca. 260 Seiten
Eine praxisorientierte Spiel- und Arbeitshilfe für den Schneider CPC 464 · BASIC · Grafik · Sound · Tastaturanwendung · Kassettenrecordereinsatz · alle Befehle kompakt und systematisch dargestellt · modular aufgebaute Beispielprogramme auch zur Textverarbeitung und Datenverwaltung · der ideale Grundstock für Ihre CPC 464-Programmbibliothek!
Best.-Nr. MT 801, ISBN: 3-89090-090-9
(Sfr. 42,30/öS 358,80) **DM 46,—**

SINCLAIR

Maschinencode-Programme für den ZX Spectrum
Juni 1984, 204 Seiten
Nützliche Maschinencode-Programme mit Ihrem ZX Spectrum · Sortierung von Fließkommazahlen · Übernahme von Parametern direkt von einem Basic-Pro gramm · Flußdiagramme · für Profis und solche, die es werden wollen.
Best.-Nr. MT 702, ISBN: 3-89090-023-2
(Sfr. 29,50/öS 249,60) **DM 32,—**

ZX-Spectrum Abenteuerspiele
September 1984, 208 Seiten
Die Entstehungsgeschichte der Abenteuerspiele mit repräsentativen Beispielen für jede »Epoche« · Ein Programm speziell für Ihren ZX-Spectrum: ›Das Auge des Sternenkriegers‹, ein Grafik-Abenteuerspiel, das Sie in Atem hält!
Best.-Nr. MT 712, ISBN: 3-89090-047-X
(Sfr. 27,50/öS 232,40) **DM 29,80**

Astronomie-Programme für den ZX-Spectrum
September 1984, 268 Seiten
Eine phantastische Reise in die Welt des Kosmos mit Ihrem ZX-Spectrum: Der Julianische Kalender · Die Mondphasen · Eigene Satelliten starten · Kepler's Umlaufbahnen · Die Umlaufbahn Plutos · Interessant nicht nur für Hobby-Astronome.
Best.-Nr. MT 732, ISBN: 3-89090-048-8
(Sfr. 27,50/öS 232,40) **DM 29,80**

Schnelles Rechnen mit dem ZX81
Oktober 1984, 276 Seiten
Das Betriebssystem · der BASIC-Interpreter · Gleitkomma-Macro-Befehle zur Verkürzung der Rechenzeiten · alle Programmbeispiele sind lauffähig auf dem ZX81 mit dem 1K-RAM-Speicher, ein 16K-Speicher vereinfacht die Programmentwicklung.
Best.-Nr. MT 706, ISBN: 3-89090-073-9
(Sfr. 27,50/öS 232,40) **DM 29,80**

ZX-Spectrum Hardware
Januar 1985, 147 Seiten
Dieses Buch vermittelt Ihnen ein fundiertes Basiswissen über Aufbau und Entwicklung eigener Hardware · Ausführliche Beschreibung der einzelnen ICs mit Abbildungen und 2-System-Schaltplänen · Anschluß einer PIO-Ansteuerung von Dezimalanzeigen · Leuchtdioden · Relais · DIL-Schalter · Eine akkugepufferte Hardwareuhr mit vierstelliger Anzeige · Soundgenerator mit drei Kanälen.
Best.-Nr. MT 737, ISBN: 3-89090-092-5
(Sfr. 27,50/öS 232,40) **DM 29,80**

TI 99/4A

21 LISTige Programme für den TI-99/4A
November 1984, 224 Seiten
Umfangreiche Spiele aller Art für den TI-99/4A · nützliche Utilities · Adressenverwaltung · Vokabel-Programm · für manche Programme ist das Extended-BASIC-Modul, die Speichererweiterung (32 K), ein Disketten-Laufwerk oder Joysticks erforderlich!
Best.-Nr. MT 754, ISBN: 3-89090-065-8
(Sfr. 23,—/öS 193,40) **DM 24,80**

VC 20

Lerne Basic auf dem VC-20
August 1984, 320 Seiten
Das neue Basic-Lehrbuch für den Commodore VC-20 · einfach erklärte Basic-Befehle mit Übungen · viele heiße Actionspiele · nützliche Programmiertricks · mit ausführlichem Begriffslexikon · der Renner für junge Computer-Freaks!
Best.-Nr. MT 691, ISBN: 3-89090-008-9
(Sfr. 35,—/öS 296,40) **DM 38,—**

Lehrspielzeug Computer: C 64/VC-20
Juli 1984, 139 Seiten
Speziell für Kinder entwickelt führt dieses Buch spielerisch in die Basic-Welt des Commodore 64/VC-20 ein · mit vielen lehrreichen Spielprogrammen und Grafikmöglichkeiten · kleinere Kinder benötigen die Hilfe ihrer sachkundigen Eltern.
Best.-Nr. MT 695, ISBN: 3-89090-011-9
(Sfr. 23,—/öS 193,40) **DM 24,80**

Computer für Kinder — Ausgabe Commodore VC 20
1984, 92 Seiten
»Computer für Kinder« richtet sich an Kinder im Alter von 8 bis 13 Jahren, die Interesse für Computer zeigen · unterhaltsame und leichtverständliche Einführungstexte und Beispiele · mit einem besonderen Abschnitt für Lehrer und Eltern.
Best.-Nr. PW 708, ISBN: 3-921803-40-3
(Sfr. 27,50/öS 232,40) **DM 29,80**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag Aktiengesellschaft Buchverlag, Hans-Pinsel-Str. 2, 8013 Haar

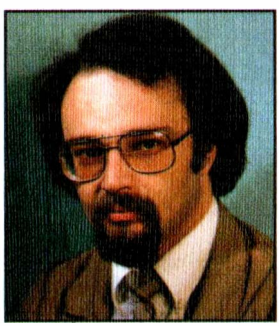
Das C64-Profihandbuch

In diesem Buch sind alle wichtigen und nützlichen Informationen für professionelle Anwendungen mit dem Commodore 64 zusammengetragen. Nicht nur für Profis, auch für Anwender, die mehr über ihren 64er erfahren wollen, ist dieses Buch eine Hilfe.

Das erste Kapitel beschäftigt sich mit allgemeinen Algorithmen, die nicht nur speziell für den 64er angewendet werden können, sondern auch auf andere Rechner übertragbar sind.

Im zweiten Kapitel sind einige Utilities (getrennt nach BASIC- und Maschinenprogrammen) aufgeführt. Da sich BASIC-Erweiterungen im Commodore 64 ohne Hardwareänderungen sehr leicht durchführen lassen, wurden die im zweiten Teil vorgestellten Maschinenprogramme auch als BASIC-Erweiterung ausgebaut. Besonders nützlich für den Profi sind die erweiterten PEEK- und POKE-Funktionen.

Das dritte Kapitel widmet sich den sechs Verbindungen des Rechners nach außen hin zu, und im vierten Kapitel sind einige nützliche Tips und Tricks aufgezählt, die von einzelnen PEEKs und POKEs über SYS-Aufrufe bis hin zu kurzen Programmen reichen. Das letzte Kapitel stellt die Internas des Commodore 64 in tabellarischer Form sowie in einigen Abbildungen zusammen. Besonders wird hier auf den Speicher und die Registerverteilung (VIC, SID, CIA) eingegangen, aber auch die im ROM implementierten Routinen werden angesprochen. Im Buch befindet sich auch eine ausführliche Fehlerbeschreibung, sowohl der Fehler des Rechners als auch der Floppy. Neben dem Literatur- und einem umfangreichen Stichwortverzeichnis finden Sie im Anhang zwei Übersichten zur Übernahme der vorgestellten BASIC-Erweiterungen in Ihrem Rechner (Hex-Listing, BASIC-Loader).



HANS LORENZ SCHNEIDER geboren am 15.10.1953 in Köln. Nach dem Abitur 1973 studierte er von 1976 bis 1980 Informatik an der Bundeswehrhochschule in München. Seit 1980 ist Schneider Inhaber und Geschäftsführer eines Software-Hauses, das sich hauptsächlich mit der Erstellung von Individual-Software für Mikrocomputer befaßt.

WERNER EBERL geboren am 23.2.1962 in München, begann gleich nach dem Abitur 1980 sein Physik-Studium. Seine große Leidenschaft waren schon immer die Computer. Seit 1980 ist er auch als freier Mitarbeiter für ein Software-Haus tätig, wo er für die Umsetzung von Konzepten in lauffähige Programme verantwortlich ist.

ISB N 3- 89090-110-7

Markt&Technik
Verlag Aktiengesellschaft

4 001057 901100

DM 52,-
sFr. 47,80
öS 405,60