COMMODORE

Commodore Sachbuchreihe Band 1

ALLES ÜBER DEN COMMODORE 64

Programmierhandbuch

COMMODORE 64 QUICK REFERENCE

EINFACHE VARIABLEN

Typ	*Name*	*Bereich*
Reelle Zahl	XY	±1.70141183E+38 ±2.93873588E+39
Ganze Zahl	XY%	±32767
Zeichenkette	XY$	0 bis 255 Zeichen

X ist ein Buchstabe (A–Z), Y ein Buchstabe oder eine Zahl (0–9). Variablennamen können aus mehr als zwei Zeichen bestehen, es werden jedoch nur die beiden ersten erkannt.

FELDVARIABLEN

Typ	*Name*
Eindimensional	XY(5)
Zweidimensional	XY(5,5)
Dreidimensional	XY(5,5,5)

Felder aus max. 11 Elementen (Index 0–10) werden automatisch dimensioniert. Felder mit mehr als 11 Elementen müssen explizit dimensioniert werden (DIM).

ALGEBRAISCHE OPERATOREN

= Ordnet Wert einer Variablen zu
– Negation
↑ Exponentenrechnung
* Multiplikation
/ Division
+ Addition
– Subtraktion

VERGLEICHS- UND LOGISCHE OPERATOREN

= Gleich
<> Ungleich
< Kleiner
> Größer
<= Kleiner oder gleich
>= Größer oder gleich
NOT Logisches "Nicht"
AND Logisches "Und"
OR Logisches "Oder"

Der Ausdruck ist 1, wenn er wahr ist, oder 0, wenn er falsch ist.

SYSTEMBEFEHLE

LOAD "NAME"	Laden eines Programms von Kassette
SAVE "NAME"	Speichern eines Programms auf Kassette
LOAD "NAME",8	Laden eines Programms von Diskette
SAVE "NAME",8	Speichern eines Programms auf Diskette
VERIFY "NAME"	Prüfen, ob das Programm ohne Fehler gespeichert wurde.
RUN	Programmausführung
RUNxxx	Programmausführung beginnt bei Zeile xxx
STOP	Stop der Ausführung
END	Beendigung der Ausführung
CONT	Fortsetzen der Programmausführung ab der Zeile, in der das Programm unterbrochen wurde.
PEEK(X)	Wiedergabe des Inhalts von Speicherplatz X
POKE X,Y	Ändern des Inhalts von Speicherplatz X zu Wert Y
SYS xxxxx	Sprung zur Auführung eines Maschinenspracheprogramms, beginnend bei xxxxx
WAIT X,Y,Z	Programm wartet, bis Inhalt von Platz X, bei der EOR-Verknüpfung mit Z und AND-Verknüpfung mit Y ungleich 0 ist.
USR(X)	Übergibt Wert X zu einem Maschinensprache-Unterprogramm

EDITIER- UND FORMATIERBEFEHLE

LIST	Auflisten eines gesamten Programms
LIST A–B	Auflisten von Zeile A bis Zeile B
REM Text	Kommentartext kann geschrieben werden, wird jedoch während der Programmausführung nicht berücksichtigt.
TAB(X)	Verwendung in der PRINT-Anweisung. Weiterrücken von X Positionen auf dem Bildschirm.
SPC(X)	Setzt X Leerzeichen in eine Zeile.
POS(X)	Übergibt derzeitige Cursorposition
CLR/HOME	Cursor wird in linke obere Bildschirmecke positioniert.
SHIFT CLR/HOME	Bildschirm wird gelöscht und Cursor in Ausgangsposition gebracht.
SHIFT INST/DEL	Einfügen eines Leerzeichens an derzeitiger Cursorposition
INST/DEL	Löschen des Zeichens an der derzeitigen Cursorposition
CTRL	Beim Einsatz mit numerischer Farbtaste Wahl der Textfarbe. Kann in PRINT-Anweisung benutzt werden.
CRSR	Zur Bewegung des Cursors nach oben, unten, links und rechts.
COMMODORE-TASTE	Zusammen mit SHIFT für die Wahl von Zeichen der oberen/unteren Umschaltstellung und der Graphikanzeige. Beim gemeinsamen Einsatz mit numerischen Farbtasten Wahl der optionalen Textfarbe.

FELDER UND ZEICHENKETTEN

DIM A(A,Y,Z)	Setzt maximale Indizes für A; reserviert Speicherplätze für (X+1)*(Y+1)*(Z+1)–Elemente, beginnend bei A(0,0,0)
LEN (X$)	Übergibt Zeichenanzahl in X$
STR$(X)	Übergibt Zeichenwert von X, in Zeichenkette verwandelt
VAL(X$)	Übergibt numerischen Wert von X$ bis zu ersten nichtnumerischen Zeichen.
CHR$(X)	Übergibt ASCII-Zeichen mit dem Code X
ASC(X$)	Übergibt ASCII-Code für erstes Zeichen von X$
LEFT$(A$,X)	Übergibt die ersten X Zeichen von A$
RIGHT$(A$,X)	Übergibt die letzten X Zeichen von A$
MID$(A$,X,Y)	Übergibt Y Zeichen von A$ beginnend bei Zeichen X

EIN-/AUSGABEBEFEHLE

INPUT A$ OR A	Zeigt "?" auf dem Bildschirm an und wartet, bis vom Bediener eine Zeichenkette oder ein Wert eingegeben wird.
INPUT "ABC";A	Zeigt Meldung an und wartet auf Eingabe eines Werts. Kann auch INPUT A$ sein
GET A$ oder A	Wartet auf Eingabe eines 1-Zeichenwerts; kein RETURN erforderlich
DATA A,"B",C	Initialisiert einen Wertsatz, der über die READ-Anweisung benutzt werden kann
READ A$ oder A	Ordnet nächsten Datenwert A$ oder A zu
RESTORE	Stellt Datenzeiger zurück, um die DATA-Liste erneut mit READ zu lesen
PRINT "A=";A	Zeigt Zeichenkette "A=" und Wert von A an, ";" unterdrückt Leerzeichen, "," setzt Daten in nächstes Tabulatorfeld.

PROGRAMMABLAUF

GOTO X	Verzweigung zu Zeile X
IF A=3 THEN 10	Ist die Behauptung richtig, dann Ausführung des weiteren Anweisungsteils. Ist sie falsch, Ausführung der nächsten Zeilennummer
FOR A=1 TO 10 STEP 2 : NEXT	Führt alle Anweisungen zwischen FOR und NEXT aus, A geht von 1 bis 10 mit der Schrittweite 2. Schrittweite ist 1, wenn STEP fehlt.
NEXT A	Definiert Schleifenende. A ist optional.
GOSUB 2000	Verzweigung zu Unterprogramm beginnend bei Zeile 2000
RETURN	Kennzeichnet Unterprogrammende. Rückkehr zur Anweisung nach dem letzten GOSUB
ON X GOTO A,B	Verzweigung zur X-ten Zeilennummer der Liste. Ist X=1, Verzweigung zu A usw.
ON X GOSUB A,B	Verzweigung zu Unterprogramm bei X-ter Zeilennummer der Liste.

ALLES ÜBER DEN COMMODORE 64

Programmierhandbuch

Herausgeber
Commodore Büromaschinen GmbH
Lyoner Straße 38
6000 Frankfurt/Main 71

Copyright © Terrapin Inc. 1982, 1983 und Techn. Institut von Massachusetts.
Copyright © der deutschen Ausgabe bei Commodore Büromaschinen GmbH, Frankfurt 1984.

Das Kopieren von Software, des Handbuches, ganz oder auszugsweise, verstößt gegen das Gesetz, sofern nicht das Einverständnis der Urheberrechtsinhaber vorliegt.

INHALTSVERZEICHNIS

EINFÜHRUNG IX
- Was ist alles enthalten? X
- Wie diese Programmieranleitung zu benutzen ist XI
- Hinweise zur Anwendung Ihres Commodore 64 XII
- Anwendung XIII

1. BASIC PROGRAMMIERHINWEISE 1
- Einleitung 2
- Bildschirmcodes (BASIC-Zeichensatz) 2
  - Das Betriebssystem (OS) 2
- Programmieren von Zahlen und Variablen 4
  - Ganze Zahlen, Gleitpunktzahlen und Zeichenketten 4
  - Ganze Zahlen, Gleitpunktzahlen und Stringvariablen 7
  - Ganzzahlige, Gleitpunkt- und Stringfelder 8
- Ausdrücke und Operatoren 10
  - Rechenausdrücke 10
  - Rechenoperationen 10
  - Vergleichsoperatoren 12
  - Logische Operatoren 13
  - Priorität der Operationen 15
  - Zeichenkettenoperationen 17
  - Strings 17
- Programmiertechniken 18
  - Datenumsetzung 18
  - Verwendung der Eingabeanweisung 18
  - Arbeiten mit der GET-Anweisung 22
  - Komprimieren von Basic-Programmen 24

2. BASIC-VOKABULAR 29
- Einführung 30
- Beschreibung der Basic-Schlüsselwörter 35
- Tastatur und Merkmale des Commodore 64 92
- Bildschirmeditor 94

3. GRAPHIKPROGRAMMIERUNG MIT DEM COMMODORE 64 99

- Graphikübersicht 100
  - Zeichenanzeige 100
  - Bit-Map-Modus 100
  - Sprites 100
- Lage der Graphikzeichen 101
  - Wahl der Videobank 101
  - Bildschirmspeicher 102
  - Farbspeicher 103
  - Zeichenspeicher 103
- Standardzeichenmodus 107
  - Zeichendefinitionen 107
- Programmierbare Zeichen 108
- Mehrfarbige Graphiken 115
  - Das Mehrfarben-Modus-Bit 115
- Erweiterter Hintergrundfarbmodus 120
- Graphiken durch Bit-Mapping 121
  - Standard-Bit-Mapping mit hoher Auflösung 122
  - Funktionsweise 122
- Mehrfarben-Bit-Mapping 127
- Kontinuierliches Verschieben 128
- Sprites 131
  - Spritedefinition 131
  - Sprite-Pointer 133
  - Einschalten der Sprites 134
  - Ausschalten der Sprites 135
  - Farben 135
  - Mehrfarbenmodus 135
  - Wählen des Mehrfarbenmodus für ein Sprite 136
  - Vergrößerte Sprites 136
  - Spritepositionierung 137
  - Zusammenfassung über die Spritepositionierung 143
  - Sprite-Anzeigeprioritäten 143
  - Kollisionserkennung 144
- Weitere Graphikmöglichkeiten 149
  - Löschen des Bildschirms 149
  - Rasterregister 149
  - Interrupt-Statusregister 149
  - Vorschläge für Bildschirm-Zeichenfarbe-Kombinationen 151

- Programmieren von Sprites – ein anderer Aspekt 152
  - Programmierung der Sprites in Basic – ein kurzes Programm .. 152
  - Komprimieren Ihrer Sprite-Programme 155
  - Positionierung der Sprites auf dem Bildschirm 156
  - Spriteprioritäten 160
  - Zeichnen eines Sprites 161
  - Erstellen eines Sprites . . . Schritt für Schritt 162
  - Bewegen der Sprites auf dem Bildschirm 164
  - Vertikales Rollen 165
  - Die Tanzmaus – ein Sprite-Programmbeispiel 166
  - Tabelle zum einfachen Konstruieren von Sprites 175
  - Hinweise zur Spriteerstellung 176

4. MUSIKPROGRAMMIERUNG MIT DEM COMMODORE 64 181

- Einführung 182
  - Lautstärkeregelung 184
  - Tonfrequenzen 184
- Arbeiten mit mehreren Stimmen 185
  - Steuern mehrerer Stimmen 189
- Ändern der Wellenformen 190
  - Verständnis der Wellenformen 193
- Hüllkurvengenerator 194
- Filtern 197
- Fortschrittliche Techniken 200
  - Synchronisation und Ringmodulation 205

5. MASCHINENSPRACHE 207

- Was ist Maschinensprache? 208
  - Wie sieht der Maschinencode aus? 209
  - Einfache Liste der Speicherbelegung des Commodore 64 210
  - Die Register im Mikroprozessor 6510 211
- Wie schreibt man Maschinensprache-Programme? 213
  - Monitor 64 213
- Hexadezimaldarstellung 214
  - Die erste Maschinensprache-Anweisung 216
  - Schreiben des ersten Programms 218
- Adressierart 219
  - Zero-Page 219
  - Stapel (Stack) 220

- Indizieren ... 221
  - Indirekt indiziert ... 221
  - Indiziert indirekt ... 222
  - Verzweigungen und Überprüfungen ... 223
- Unterprogramme ... 224
- Hinweise für den Anfänger ... 226
- Vorbereitungen für eine große Aufgabe ... 227
- Anweisungssatz von Mikroprozessor MCS6510 – Alphabetische Reihenfolge ... 228
  - Anweisungs-Adressierarten und zugehörige Ausführungszeiten (in Taktzyklen) ... 250
- Speicherverwaltung beim Commodore 64 ... 256
- Kernal ... 264
- Kernal-Funktionen nach Einschalten der Stromversorgung ... 265
  - Arbeiten mit Kernal ... 265
  - Aufrufbare Kernal-Routinen ... 268
  - Fehlermeldungen ... 303
- Arbeiten mit Maschinensprache und Basic ... 304
  - Wo stehen Maschinensprache-Routinen? ... 306
  - Wie wird Maschinensprache eingegeben? ... 306
- Speicherbelegung des Commodore 64 ... 308
  - Ein-/Ausgabeanordnung beim Commodore 64 ... 317

6. EIN-/AUSGABE-ANLEITUNG ... 329

- Einführung ... 330
- Ausgabe auf den Bildschirm ... 330
- Ausgabe auf andere Geräte ... 331
  - Ausgabe zum Drucker ... 332
  - Arbeiten mit Magnetbandkassetten ... 333
  - Datenspeicherung auf Disketten ... 335
- Spiele-Ports ... 336
  - Drehregler ... 339
  - Lichtgriffel ... 341
- RS-232 Interface-Beschreibung ... 341
  - Allgemeiner Überblick ... 341
  - Öffnen eines RS-232-Kanals ... 342
  - Lesen der Daten von einem RS-232-Kanal ... 345
  - Übertragen von Daten über einen RS-232-Kanal ... 346
  - Schließen eines RS-232-Datenkanals ... 347
  - Basic-Programmbeispiel ... 349
  - Zeiger für Empfangs-/Übertragungspuffer ... 350

Zero-Page-Adressen und ihre Anwendung für das System-Interface RS-232 ... 351
Allgemeine RS-232-Speicherung ... 351
- Userport ... 352
Port-Pin-Beschreibung ... 352
- Der serielle Bus ... 355
Anschlüsse des seriellen Busses ... 356
- Erweiterungsanschluß ... 359
- Z-80 Mikroprozessor-Modul ... 362
Arbeiten mit Commodore CP/M ... 362
Ausführung ... 363

ANHANG ... 365
A. Abkürzungen der Basic-Schlüsselwörter ... 366
B. Bildschirm-Anzeige-Codes ... 368
C. ASCII- und CHR$-Codes ... 371
D. Bildschirm- und Farbspeichermappen ... 374
E. Musiknotenwerte ... 376
F. Literaturverzeichnis ... 380
G. Video Interface Controller (VIC) Chip Registerbelegung ... 383
H. Abgeleitete mathematische Funktionen ... 386
I. Steckerbelegung für Anschlüsse für Peripheriegeräte ... 387
J. Übertragung von fremden Basic-Programmen auf Commodore 64 Basic ... 390
K. Fehlermeldungen ... 392
L. Datenblatt Mikroprozessor 6510 ... 394
M. 6526 Complex Interface Adapter (CIA) ... 411
N. 6566/6567 Video-Interface-Controller (VIC-II) ... 429
O. 6581 Sound Interface Device (SID) Chip Specifications ... 450
P. Glossar ... 474

INDEX ... 475

EINFÜHRUNG

Die **Programmieranleitung COMMODORE 64** wurde als Hilfsmittel und Bezugsquelle für all diejenigen entwickelt, die die Fähigkeiten ihres **COMMODORE 64** optimal nutzen wollen. Diese Anleitung enthält alle Informationen, die Sie zur Erstellung von Programmen benötigen – angefangen bei den einfachsten Beispielen bis hin zu komplexen Programmen. Die **Programmieranleitung** ist so aufgebaut, daß sowohl ein BASIC-Anfänger als auch der erfahrene Maschinensprache-Programmierer die erforderlichen Informationen erhält, um eigene Programme zu erstellen. Gleichzeitig werden Sie feststellen, wie vielseitig Ihr **COMMODORE 64** wirklich ist. Das vorliegende Handbuch ist nicht dazu gedacht, Ihnen die **Programmiersprache** BASIC oder die Maschinensprache 6502 beizubringen. Sie finden jedoch ein ziemlich umfangreiches Glossar mit Fachausdrücken, sowie "lehrreiche" Hinweise. Wenn Sie noch nicht mit BASIC vertraut sind, empfehlen wir Ihnen, die **Bedienungsanleitung** des **COMMODORE 64** durchzulesen. In dieser **Anleitung** finden Sie eine leicht verständliche Einführung in die Programmiersprache BASIC. Sollte das Programmieren in BASIC Ihnen danach noch Schwierigkeiten bereiten, so schlagen Sie am Ende dieser Anleitung (oder in Anhang N in der **Bedienungsanleitung**) nach, und stellen Sie anhand des Literaturverzeichnisses fest, wo Sie die erforderlichen Informationen finden können.

Die vorliegende **Programmieranleitung** ist also nur als Referenz gedacht. Wie Sie die gegebenen Informationen nun tatsächlich umsetzen, hängt davon ab, über welches Know-how Sie bereits verfügen. Wenn Sie also in Sachen Programmierung noch ein Anfänger sind, können Sie die in dieser Anleitung gegebenen Informationen nur dann voll verstehen, wenn Sie Ihre derzeitigen Programmier-Kenntnisse ausweiten.

In dieser Anleitung finden Sie zahlreiche Programmier-Informationen, die leicht verständlich im Programmierer-Jargon geschrieben sind. Erfahrene Programmierer finden andererseits alle Informationen, um ihren **COMMODORE 64** optimal einzusetzen.

WAS IST ALLES ENTHALTEN?

- Unser komplettes "BASIC-Lexikon" umfaßt BASIC-Befehle, Anweisungen und Funktionen in alphabetischer Reihenfolge. Wir haben eine Übersicht erstellt, in der alle Wörter und ihre Abkürzungen enthalten sind. In dem folgenden Abschnitt werden die einzelnen Begriffe genau definiert und anhand von Beispielprogrammen ihre Anwendung beschrieben.
- Wenn Sie eine Einführung in die Anwendung der Maschinensprache für BASIC-Programme benötigen, wird für Sie unsere Übersicht hilfreich sein.
- Ein leistungsstarker Bestandteil des Betriebssystems aller COMMODORE-Computer wird KERNAL genannt. Hierdurch wird sichergestellt, daß die Programme, die Sie heute schreiben, auch noch auf den COMMODORE-Computern von morgen laufen können.
- Der Abschnitt über Ein-/Ausgabeprogrammierung zeigt Ihnen, wie Sie Ihren Computer voll nutzen können. In diesem Abschnitt werden die möglichen Ergänzungen beschrieben – angefangen bei Lichtstiften und Joysticks bis hin zu Diskettenstationen, Druckern und Zusatzgeräten für Telekommunikation (Modems).
- Wir zeigen Ihnen, wie man **SPRITES** und Sonderzeichen programmiert. Sie werden lernen, wie man Lauf-Bilder in hochauflösender Farbgraphik erzeugen kann.
- Wir eröffnen Ihnen die Welt der Musik-Synthese und zeigen Ihnen, wie Sie eigene Songs schreiben und Klangeffekte mit dem eingebauten Synthesizer erzielen können.
- Dem erfahrenen Programmierer zeigen wir, wie er den **COMMODORE 64** mit CP/M* und anspruchsvollen Sprachen benutzen kann.

Die **Programmieranleitung COMMODORE 64** soll also ein nützliches Werkzeug sein, damit Ihnen das zukünftige Programmieren auch wirklich Spaß macht.

---

* CP/M ist ein eingetragenes Warenzeichen von Digital Research; Inc.

WIE DIESE PROGRAMMIERANLEITUNG ZU BENUTZEN IST

Zur Beschreibung der Syntax (Struktur des Programmiertextes) von BASIC-Befehlen oder Anweisungen sowie zur Darstellung der benötigten und frei wählbaren Teile der einzelnen BASIC-Schlüsselwörter werden bestimmte allgemeine Schreibweisen benutzt. Für die Interpretation der Anweisungssyntax gelten folgende Regeln:

1. BASIC-Schlüsselwörter werden in Großbuchstaben dargestellt. Sie müssen exakt an der angegebenen Stelle und genau wie in dieser Anleitung geschrieben eingegeben werden.
2. Angaben in Anführungszeichen (" ") geben variable Daten an, die von Ihnen eingegeben werden müssen. Sowohl die Anführungszeichen als auch die Daten müssen genau an der angegebenen Stelle eingegeben werden.
3. Punkte in eckigen Klammern ([ ]) geben einen frei wählbaren Parameter an. Ein Parameter ist eine Einschränkung oder eine zusätzliche Angabe für Ihre Anweisungen. Bei der Verwendung eines frei wählbaren Parameters müssen auch die für diesen Parameter erforderlichen Daten gegeben werden. Auslassungen (. . .) geben an, daß eine bestimmte Angabe so oft wiederholt werden kann, wie es eine Programmierzeile zuläßt.
4. Ist eine Angabe in ([ ]) UNTERSTRICHEN, bedeutet dies, daß diese bestimmten Zeichen in den frei wählbaren Parametern benutzt werden müssen und genau wie angegeben zu schreiben sind.
5. Angaben in spitzen Klammern (< >) geben variable Daten an, die von Ihnen eingegeben werden. Ein Schrägstrich (/) zeigt Ihnen an, daß Sie sich zwischen mehreren Funktionen entscheiden können.

SYNTAX-FORMAT-BEISPIEL:

OPEN <logische Adresse>,<Gerätenummer> [,<Adresse>], ["<Laufwerk>: <Dateiname>] [<Modus>]"

ANWEISUNGSBEISPIELE:

```
10 OPEN 2,8,6,"0:LAGERBESTAND,S,W"
20 OPEN 1,1,2,"SCHECKBUCH"
30 OPEN 3,4
```

In der Praxis kann die Parameterfolge in Ihren Anweisungen von der der Syntax-Beispiele abweichen. Die Beispiele sind also Einzelbeispiele und sollen nicht jede mögliche Folge zeigen, sondern lediglich alle erforderlichen und frei wählbaren Parameter darstellen.

In den gegebenen Programmierbeispielen sind Wörter und Operatoren durch Leerzeichen voneinander getrennt, damit die Beispiele besser lesbar sind. Normalerweise erfordert BASIC jedoch keine Leerzeichen zwischen Wörtern, außer wenn sich durch ein Auslassen eine mehrdeutige oder falsche Syntax ergibt.
Nachfolgend werden einige der Symbole beschrieben, die in den folgenden Kapiteln für verschiedene Anweisungsparameter benutzt werden. Diese Liste zeigt nicht alle Möglichkeiten, sondern soll Ihnen lediglich zeigen, wie Syntax-Beispiele aufgebaut sind.

SYMBOL	BEISPIEL	BESCHREIBUNG
<Logische Adresse>	50	Logische Dateinummer
<Gerätenummer>	4	Hardware-Gerätenummer
<Adresse>	15	Sekundär-Adreßnummer eines seriellen Bus-Anschlußgeräts
<Laufwerk>	0	Diskettenlaufwerknummer
<Dateiname>	"TEST.DATA"	Name einer Daten- oder Programmdatei
<Konstante>	"ABCDEFG"	Vom Programmierer eingegebene beliebige Daten
<Variable>	X145	Ein beliebiger BASIC-Variablenname oder eine Konstante
<String>	AB$	Eine String-Variable ist erforderlich
<Zahl>	12345	Eine numerische Variable ist erforderlich
<Zeilennummer>	1000	Tatsächliche Programmzeilen-Nr.
<Numerisch>	1.5E4	Ganze Zahl oder Gleitpunktvariable

HINWEISE ZUR ANWENDUNG IHRES COMMODORE 64

Als Sie das erstemal an den Kauf eines Computers dachten, haben Sie sich sicherlich gefragt: "Nun kann ich mir einen Computer leisten, aber was kann ich denn alles mit ihm anfangen?"
Das Besondere an Ihrem **COMMODORE 64** ist, daß er all das machen kann, was SIE wollen! Er kann rechnen und für Sie Ihren geschäftlichen oder privaten Haushalt führen. Sie können ihn auch für die Textverarbeitung einsetzen. Sie können mit ihm Aktionsspiele spielen. Sie können ihn singen lassen. Sie können mit ihm Ihre eigenen Zeichentricks erstellen usw. Das Beste am **COMMODORE 64** ist die Tatsache, daß er sein Geld wert ist, selbst wenn Sie ihn nur für eine der nachfolgend aufgeführten Funktionen einsetzen. Der **64** ist jedoch ein vollständiger Computer und kann daher ALLE nachstehend aufgeführten Punkte ausführen. Und wie!

Übrigens können Sie außerdem noch zahlreiche kreative und praktische Anregungen von den örtlichen **COMMODORE-Anwenderclubs** bekommen.

ANWENDUNG	BESCHREIBUNG/ERFORDERNISSE
AKTIONS-SPIELE	Sie können richtige Spiele wie Omega Race, Gorf und Wizard of Wor, aber auch Lernspiele bekommen.
ANZEIGEN UND VERKAUFSFÖRDERUNG	Schließen Sie Ihren **COMMODORE 64** an ein Fernsehgerät an, stellen Sie ihn in ein Schaufenster, und lassen Sie bewegliche Reklame ablaufen. Natürlich ist auch musikalische Untermalung möglich.
TRICKFILM	Mit der SPRITE-Graphik des COMMODOREs können Sie richtige Trickfilme auf acht verschiedenen Ebenen darstellen, so daß sich die Figuren vor- bzw. hintereinander bewegen können.
BASIC-PROGRAMMIERUNG	Die Bedienungsanleitung zum **COMMODORE 64** sowie die BASIC-Lernkassette verschaffen Ihnen einen guten Start.
KALKULATIONS-PROGRAMME	Der **COMMODORE 64** bietet Ihnen die besten Kalkulationsprogramme, die für Personal-Computer existieren.
KOMPONIEREN	Der **COMMODORE 64** ist mit einem technisch ausgereiften, eingebauten Musik-Synthesizer ausgerüstet. Er hat drei vollständig programmierbare Stimmen, verfügt über 9 Oktaven und 4 regelbare Wellenformen. Erstellen Sie mit Hilfe der COMMODORE-Musikmodule Ihre eigenen Songs und erleben Sie, welche Musik- und Klangeffekte möglich sind.
CP/M*	COMMODORE bietet CP/M* als Steckmodul mit Betriebssystem auf Diskette an.
GESCHICKLICH-KEITSÜBUNGEN	Über die verschiedenen COMMODORE-Spiele können Sie die Koordination von Hand/Auge sowie Ihre Geschicklichkeit trainieren.

AUSBILDUNG

Schon das Arbeiten mit einem Computer an sich ist eine Ausbildung. Die COMMODORE-Ausbildungsbücher enthalten allgemeine Informationen für den Einsatz von Computern als Bildungshilfsmittel. Ferner bieten wir eine Vielzahl von Lernprogrammen an. Die Auswahl erstreckt sich dabei von Musik über Mathematik bis hin zu Kunst und Astronomie.

FREMDSPRACHEN

Der programmierbare Zeichensatz des **COMMODORE 64** ermöglicht ein Auswechseln des Standardzeichensatzes gegen benutzerdefinierte Fremdsprachenzeichen.

GRAPHIK UND KUNST

Zusätzlich zu der bereits erwähnten Sprite-Graphikfunktion ist mit dem **COMMODORE 64** die Darstellung mehrfarbiger Graphiken mit hoher Auflösung, programmierbarer Zeichen und die Kombination der verschiedenen Graphik- und Zeichenmodi möglich.

INSTRUMENTEN-STEUERUNG

Ihr **COMMODORE 64** hat einen seriellen Bus, einen RS-232-Port sowie einen Benutzerport für die verschiedensten speziellen Anwendungen. Als Sonderausstattung ist außerdem ein IEEE/488-Steckmodul erhältlich.

JOURNALE UND KREATIVES SCHREIBEN

Der **COMMODORE 64** bietet Ihnen ein außergewöhnliches Textverarbeitungssystem, das mindestens genauso gut und flexibel wie viele "teuere" Word-Prozessoren ist. Natürlich können Sie die Informationen entweder über eine 1541 Diskettenstation oder einen Datasette™-Rekorder speichern und später ausdrucken lassen.

LICHTGRIFFEL-STEUERUNG

Für Anwendungen, die einen Lichtgriffel erfordern, kann ein beliebiger Lichtgriffel benutzt werden, der in den Spieleanschlußstecker des **COMMODORE 64** paßt.

MASCHINENCODE PROGRAMMIERUNG

Die **Programmieranleitung COMMODORE 64** umfaßt auch ein Kapitel über Maschinensprache sowie einen Abschnitt über BASIC/Maschinencode-

Interface. Für diejenigen, die ausführlichere Informationen wünschen, haben wir außerdem ein Literaturverzeichnis zusammengestellt.

LOHNLISTEN UND FORMULARAUSDRUCK

Der **COMMODORE 64** kann für die Handhabung der verschiedensten Buchungsgeschäfte programmiert werden. Mit Groß- und Kleinschreibung und Graphiksymbolen können Formulare leicht entworfen und danach ausgedruckt werden.

AUSDRUCKEN

Der **COMMODORE 64** kann an verschiedene Punktmatrix- und Schönschriftdrucker sowie an Plotter angeschlossen werden.

SIMULATIONEN

Durch die Computersimulationen können Sie gefährliche oder teuere Experimente bei minimalem Risiko und minimalen Kosten ausführen.

Dies sind nur einige Beispiele dafür, wie Sie Ihren **COMMODORE 64** einsetzen können. Sie werden festgestellt haben, daß der **COMMODORE 64** Ihnen für jedes Problem eine praktische Lösung bietet – bei Arbeit und Spiel, zu Hause, in der Schule und im Büro.

Wir möchten Sie darauf hinweisen, daß unsere Kundenunterstützung beim Kauf eines COMMODORE-Computers ERST BEGINNT. Wir unterstützen und ermutigen die Bildung von COMMODORE-Anwenderclubs auf der ganzen Welt. Diese Clubs sind eine ausgezeichnete Informationsquelle für alle COMMODORE-Besitzer – dies gilt sowohl für den Anfänger als auch für unsere Profis. Schließlich bietet Ihnen auch noch Ihr COMMODORE-Händler ausreichend Unterstützung und Information. Werden Sie Mitglied in einem Computer-Club, erfahren Sie Hilfe bei Computerproblemen, "reden" Sie mit anderen COMMODORE-Freunden oder empfangen Sie topaktuelle Informationen über neue Produkte, Software und Ausbildungsmöglichkeiten!

KAPITEL 1

BASIC PROGRAMMIER-HINWEISE

- Einleitung
- Bildschirmcodes (BASIC-Zeichensatz)
- Programmieren von Zahlen und Variablen
- Ausdrücke und Operatoren
- Programmiertechniken

EINLEITUNG

In diesem Kapitel wird beschrieben, wie mit BASIC Daten gespeichert und aufbereitet werden. Es umfaßt folgende Punkte:

1) Kurze Beschreibung der einzelnen Bauteile und Funktionen des Betriebssystems sowie des Zeichensatzes vom COMMODORE 64.
2) Bildung von Konstanten und Variablen. Welche Variablenarten es gibt und wie Konstante und Variablen gespeichert werden.
3) Richtlinien für Rechenoperationen, Verhältnisberechnungen, Handhabung von Strings und logische Operationen. Außerdem werden die Regeln zum Bilden von Ausdrücken und die für das Mischen von BASIC mit anderen Datentypen erforderlichen Datenumwandlungen beschrieben.

BILDSCHIRMCODES (BASIC-ZEICHENSATZ)

DAS BETRIEBSSYSTEM (OS)

Das Betriebssystem befindet sich in den ROMs und ist eine Kombination aus drei getrennten, jedoch zusammengehörigen Programm-Modulen.

1) BASIC-Interpreter
2) KERNAL
3) Bildschirm-Editor

1) Der BASIC-Interpreter ist für die Analyse der BASIC-Anweisungssyntax verantwortlich und führt die erforderlichen Berechnungen und/oder Datenaufbereitungen durch. Der BASIC-Interpreter verfügt über ein Vokabular von 25 "Schlüsselwörtern" mit besonderen Bedeutungen. Sowohl Schlüsselwörter als auch Variablennamen werden durch Buchstaben und die Zahlen 0 bis 9 gebildet. Auch bestimmte Interpunktionszeichen und Sondersymbole haben für den Interpreter eine Bedeutung. Die Sonderzeichen sind in Tabelle 1.1. aufgelistet.

2) Der KERNAL handhabt die Verwaltung auf Interrupt-Ebene (bezüglich Einzelheiten siehe Kapitel 5). Der KERNAL erledigt auch die tatsächliche Datenein-/-ausgabe.

3) Über den Bildschirm-Editor wird die Ausgabe auf dem Bildschirm (Fernsehgerät) gesteuert und der BASIC-Programmtext aufbereitet. Darüber hinaus prüft er die Eingabe über die Tastatur und entscheidet, ob sofort auf die eingegebenen Zeichen reagiert werden soll oder ob diese zum BASIC-Interpreter weitergeleitet werden.

Tabelle 1.1. CMB BASIC-Zeichensatz

ZEICHEN	BEZEICHNUNG UND BESCHREIBUNG
	LEERZEICHEN – trennt Schlüsselwörter und Variablennamen
;	SEMIKOLON – wird in Variablenlisten zur Ausgabeformatierung benutzt
=	GLEICHHEITSZEICHEN – Wertzuordnung und logische Prüfung
+	PLUSZEICHEN – Addition oder Verkettung von Zeichenketten
–	MINUSZEICHEN – Subtraktion, Vorzeichen
*	STERNCHEN – Multiplikation
/	SCHRÄGSTRICH – Division
↑	AUFWÄRTSPFEIL – Exponentenrechnung
(	LINKE KLAMMER – Auswertung von Ausdrücken und Funktionen
)	RECHTE KLAMMER – Auswertung von Ausdrücken und Funktionen
%	PROZENT – Bestimmt Variablennamen als ganze Zahl
#	"Nummer" – Kommt vor logischer Dateinummer bei Ein-/Ausgabeanweisungen
$	DOLLARZEICHEN – Bestimmt Variablenname als String
,	KOMMA – Wird in Variablenlisten zur Ausgabeformatierung benutzt; trennt außerdem Befehlsparameter
.	PUNKT – Dezimalpunkt bei Gleitpunktkonstanten
"	ANFÜHRUNGSZEICHEN – Schließt Strings ein
:	DOPPELPUNKT – Trennt mehrere BASIC-Anweisungen in einer Zeile
?	FRAGEZEICHEN – Abkürzung für das Schlüsselwort PRINT
<	KLEINER ALS – Wird bei logischen Vergleichen benutzt
>	GRÖSSER ALS – Wird bei logischen Vergleichen benutzt
π	PI – Numerische Konstante 3,141592654

Das Betriebssystem ermöglicht Ihnen, auf zwei Arten mit BASIC zu arbeiten:

1) DIREKT-MODUS
2) PROGRAMM-MODUS

1) Im DIREKT-MODUS steht vor BASIC-Anweisungen keine Zeilenzahl. Sie werden nach Drücken der Taste RETURN ausgeführt.
2) Den PROGRAMM-MODUS benutzen Sie zum Ausführen von Programmen.

   Im PROGRAMM-MODUS muß vor jeder BASIC-Anweisung eine Zeilenzahl stehen. In einer Programmzeile kann mehr als eine BASIC-Anweisung stehen. Die Anzahl der Anweisungen ist jedoch begrenzt, da in eine logische Bildschirmzeile nur 80 Zeichen eingegeben werden können. D. h., bei Überschreitung dieser Grenze von 80 Zeichen muß die ganze BASIC-Anweisung, die nicht mehr in die Zeile paßt, mit einer neuen Zeilennummer in eine neue Zeile eingegeben werden.

Der COMMODORE 64 hat zwei vollständige Zeichensätze, die Sie entweder über Tastatur oder in Ihren Programmen benutzen können.
Den SATZ 1, der die Großbuchstaben sowie die Zahlen 0 bis 9 umfaßt, erreicht man ohne Drücken der Taste SHIFT. Wird die Taste SHIFT gedrückt, so sind die Graphikzeichen rechts auf der Tastenvorderseite wirksam. Wird während des Schreibens die Taste C= gedrückt, so sind die Graphikzeichen der linken Seite wirksam. Wird eine Taste ohne Graphiksymbole zusammen mit der Taste SHIFT betätigt, so wird das Symbol ganz oben auf dieser Taste wirksam.
In SATZ 2 stehen die Kleinbuchstaben sowie die Zahlen 0 bis 9 ohne Drücken der Taste SHIFT zur Verfügung. Für die Großbuchstaben wird während des Schreibens die Taste SHIFT gedrückt. Auch bei diesem Satz werden die Graphiksymbole auf der linken Tastenvorderseite durch Drücken der Taste C= angezeigt. Wird eine Taste ohne Graphikzeichen zusammen mit der Taste SHIFT betätigt, so werden die Symbole ganz oben auf dieser Taste wirksam.
Um von einem Zeichensatz auf den anderen umzuschalten, werden die Tasten C= und SHIFT zusammen gedrückt.

PROGRAMMIEREN VON ZAHLEN UND VARIABLEN

GANZE ZAHLEN, GLEITPUNKTZAHLEN UND ZEICHENKETTEN

Konstanten sind Daten, die in den BASIC-Anweisungen enthalten sind. BASIC benutzt diese Werte, um bei der Interpretation der Anweisungen Daten darzustellen. CBM BASIC kann drei verschiedene Arten von Konstanten erkennen und verarbeiten:

1) GANZE ZAHLEN
2) ZAHLEN MIT GLEITPUNKT
3) ZEICHENKETTEN

GANZZAHLIGE KONSTANTEN sind ganze Zahlen (Zahlen ohne Dezimalpunkt). Ganzzahlige Konstanten können aus dem Bereich von −32768 bis +32767 gewählt werden. *Zwischen den einzelnen Stellen stehen weder Dezimalpunkte noch Kommata.* Wird das Pluszeichen (+) ausgelassen, so werden die Konstanten als positive Zahl angesehen. Nullen vor einer Konstante werden nicht berücksichtigt und sollten daher auch nicht benutzt werden, da sie lediglich Speicherkapazität vergeuden und das Programm verlangsamen. Sie führen jedoch zu keinem Fehler. Ganze Zahlen werden im Speicher als Zwei-Byte-Binärzahlen gespeichert. Ganzzahlige Konstanten sind z. B.:

−12
8765
−32768
+44
0
−32767

Anmerkung: In eine Zahl NIE Kommata eingeben. Z. B. wird für die englische Tausenderangabe 32,000 lediglich 32000 eingegeben. Durch die Verwendung eines Kommas in einer Zahl entsteht ein Fehler, und die BASIC-Fehlermeldung **?SYNTAX ERROR** wird angezeigt.

Gleitpunktkonstanten sind positive oder negative Dezimalzahlen. Ein Dezimalbruch wird durch einen Dezimalpunkt angezeigt. Bitte denken Sie daran, daß Kommata NICHT zwischen Zahlen benutzt werden dürfen. Wird das Pluszeichen (+) vor einer Zahl ausgelassen, geht der COMMODORE 64 davon aus, daß diese Zahl positiv ist. Genau wie bei den ganzen Zahlen werden auch hier Nullen vor einer Konstante nicht berücksichtigt. Gleitpunktkonstanten können auf zwei verschiedene Arten benutzt werden:

1) EINFACHE ZAHL
2) TECHNISCH-WISSENSCHAFTLICHE NOTATION

Gleitpunktkonstanten werden auf dem Bildschirm mit bis zu neun Stellen angezeigt. Mit diesen neun Stellen können Werte zwischen −999999999. und +999999999. dargestellt werden. Werden mehr als neun Stellen eingegeben, so wird die Zahl entsprechend der zehnten Stelle auf- bzw. abgerundet. Ist die zehnte Stelle größer oder gleich 5, wird die Zahl aufgerundet; ist sie kleiner als 5, erfolgt eine Abrundung. Dies kann möglicherweise für Endsummen von Bedeutung sein.

Gleitpunktzahlen benötigen einen Speicherplatz von 5 Bytes und werden mit einer Genauigkeit von 10 Stellen verarbeitet.
Beim Ausdruck bzw. bei der Anzeige werden die Zahlen jedoch auf neun Stellen gerundet. Gleitpunktzahlen sind z. B.:

```
 1.23
-.998877
+3.1459
 .7777777
-333.
 .01
```

Zahlen, die kleiner als .01 oder größer als 999999999. sind, werden in *technisch-wissenschaftlicher Notation* angezeigt. Hierbei besteht eine Gleitpunktkonstante aus drei Teilen:

1) MANTISSE
2) BUCHSTABE E
3) EXPONENT

Die *Mantisse* ist eine einfache Gleitpunktzahl. Der Buchstabe E zeigt Ihnen an, daß die Zahl in exponentieller Form dargestellt wird. D. h., *E* bedeutet *10 (z. B. 3E3=3*10↑3=3000). Der *Exponent* gibt an, wie oft die Mantisse mit dem Faktor 10 multipliziert wird.
Sowohl Mantisse als auch Exponent sind Zahlen mit Vorzeichen (+ oder −). Für Exponenten gilt der Bereich von −39 bis +38. Der Exponent gibt die Stellenanzahl an, um die der Dezimalpunkt in der Mantisse bei der Darstellung als ganze Zahl nach links (−) oder rechts (+) verschoben wird.
Für BASIC gilt eine Begrenzung der Gleitpunktzahlen, selbst bei wissenschaftlichen Notationen: Die größte Zahl lautet +1.70141183E+38. Bei Berechnungen, deren Ergebnis zu einer größeren Zahl führt, wird die BASIC-Fehlermeldung **?OVERFLOW ERROR** angezeigt. Die kleinste Gleitpunktzahl lautet +2.93873588E−39, und bei Berechnungen, deren Ergebnis kleiner als dieser Wert ist, wird Null als Ergebnis angezeigt, und es erfolgt keine Fehlermeldung. Gleitpunktzahlen in wissenschaftlichen Notationen (sowie ihre Dezimalwerte) sind z. B.:

```
235.988E-3     (.235988)
2359E6         (2359000000.)
-7.09E-12      (-.00000000000709)
-3.14159E+5    (-314159.)
```

Stringkonstanten sind Gruppen von alphanumerischen Zusammensetzungen wie Buchstaben, Zahlen und Symbolen. Wird eine Zeichenkette (String) über die Tastatur eingegeben, so steht hierfür die restliche Kapazität einer 80-Zeichen-Zeile zur Verfügung (die NICHT von der Zeilennummer oder anderen Teilen der Anweisung beansprucht wird).
Eine Stringkonstante kann Leerzeichen, Buchstaben, Zahlen, Interpunktionszeichen und Farb- oder Cursorsteuerzeichen in beliebiger Kombination enthalten. Hier können zwischen Ziffern auch Kommata eingegeben werden. *Das einzige Zeichen, das innerhalb einer Zeichenkette nicht zulässig ist, ist das Anführungszeichen (")*, da durch dieses häufig Anfang und Ende einer Zeichenkette gekennzeichnet ist. Eine Zeichenkette kann auch leer sein – d. h., keine Zeichendaten enthalten. Das abschließende Anführungszeichen kann bei einer Zeichenkette weggelassen werden, wenn dieses das letzte Zeichen in einer Zeile ist oder wenn danach ein Doppelpunkt folgt (:). Stringkonstanten sind z. B.:

```
""          (Ein Leerstring)
"HALLO"
"$25,000.00"
"ZAHL DER ANGESTELLTEN"
```

Anmerkung: Um Anführungszeichen (") in Zeichenketten einzuschließen, wird CHR$(34) benutzt.

GANZE ZAHLEN, GLEITPUNKTZAHLEN UND STRINGVARIABLEN

Variablen stehen für Daten in BASIC-Anweisungen. Der durch eine Variable dargestellte Wert kann zugeordnet werden, indem er gleich einer Konstanten gesetzt wird. Er kann auch das Ergebnis einer Programmberechnung sein. Variablen können wie Konstanten ganze Zahlen, Gleitpunktzahlen oder Zeichenketten (Strings) sein.
Wird in einem Programm eine Variable verwendet, ohne daß ihr vorher ein Wert zugeordnet wurde, erstellt der BASIC-Interpreter automatisch die Variable mit dem Wert Null, wenn es sich um eine ganze Zahl oder eine Gleitpunktzahl handelt. Wenn es sich um einen String handelt, wird ein Leerstring angenommen.
Variablennamen können eine beliebige Länge haben. Im CBM-BASIC sind jedoch nur die ersten zwei Zeichen signifikant. D. h., bei allen Variablenbezeichnungen müssen die ersten zwei Buchstaben unbedingt unterschiedlich sein. *Variablennamen dürfen NICHT genau wie BASIC-Schlüsselwörter lauten, und sie dürfen auch keine Schlüsselwörter enthalten.* Schlüsselwörter umfassen alle BASIC-Befehle, Anweisungen, Funktionsbezeichnungen und logische Operatorbezeichnungen.

Wird aus Versehen ein Schlüsselwort inmitten eines Variablennamens benutzt, wird auf dem Bildschirm die BASIC-Fehlermeldung **?SYNTAX ERROR** angezeigt.
Variablennamen können aus dem Alphabet und den Zahlen 0–9 gebildet werden. Das erste Zeichen des Namens muß ein Buchstabe sein.
Als letztes Zeichen können die Daten-Vereinbarungszeichen (%) und ($) stehen. Das Prozentzeichen (%) gibt an, daß es sich bei der Variablen um eine ganze Zahl, und das Dollarzeichen ($), daß es sich um eine Stringvariable handelt. Wird keines dieser Vereinbarungszeichen benutzt, nimmt der Interpreter an, daß es sich um eine Gleitpunktvariable handelt. Variablennamen, Wertzuordnung und Datenarten sind z. B.:

```
A$="GROSS SALES"   (Stringvariable)
MTH$="JAN"+A$      (Stringvariable)
K%=5               (Ganzzahlige Variable)
CNT%=CNT%+1        (Ganzzahlige Variable)
FP=12.5            (Gleitpunktvariable)
SUM=FP*CNT%        (Gleitpunktvariable)
```

GANZZAHLIGE, GLEITPUNKT- UND STRINGFELDER

Ein Feld ist eine Tabelle (oder Liste) zusammengehöriger Daten mit einem einzigen Variablennamen. Ein Feld ist also eine Folge zusammengehöriger Variablen. So kann z. B. eine Zahlentabelle als ein solches Feld angesehen werden. Die einzelnen Zahlen der Tabelle werden zu einzelnen Feldelementen.
Mit Feldern läßt sich auf einfache Weise eine große Anzahl zusammengehöriger Variablen beschreiben. Nehmen wir z. B. eine Zahlentabelle. Diese besteht aus zehn Reihen mit jeweils 20 Zahlen. Das ergibt insgesamt 200 Zahlen. Ohne einen gemeinsamen Feldnamen müßten Sie für den Aufruf jedem Wert in der Tabelle einen eigenen Namen zuordnen. Wenn man jedoch Felder benutzt, braucht einem Feld auch nur ein Name zugeordnet zu werden. Alle Elemente im Feld werden durch ihre jeweilige Lage identifiziert.
Feldnamen können vom Typ Ganze Zahl, Gleitpunktzahl oder String sein. Für alle Elemente im Feld gilt dann entsprechend dem Feldnamen der gleiche Datentyp. Felder können nur eine Dimension (in einfacher Liste) oder auch mehrere Dimensionen haben (stellen Sie sich ein durch Reihen und Spalten gekennzeichnetes Gitter oder einen Rubik-Würfel® vor). Jedes einzelne Feldelement wird durch einen Index (oder eine Indexvariable) identifiziert, der nach dem Feldnamen folgt und in Klammern ( ) eingeschlossen ist.
Die Anzahl der Dimensionen eines Feldes darf theoretisch nicht größer als 255 sein, und für jede Dimension ist die Anzahl der Elemente auf 32767 beschränkt. Bei der

praktischen Anwendung sind die Feldgrößen jedoch durch die Speicherkapazität und/oder eine logische Bildschirmzeile von 80 Zeichen beschränkt.
Hat ein Feld nur eine Dimension und überschreitet sein Indexwert nie 10 (11 Elemente: 0 bis 10), dann wird dieses Feld vom Interpreter erstellt und mit Nullen gefüllt, wenn das erste Mal auf ein beliebiges Feldelement Bezug genommen wird. Ansonsten muß die BASIC-Anweisung DIM zur Definition von Form und Größe des Feldes benutzt werden.
Der für ein Feld erforderliche Speicherbedarf läßt sich wie folgt bestimmen:

```
      5 Bytes für den Feldnamen
    + 2 Bytes für jede Felddimension
    + 2 Bytes pro Element für ganze Zahlen
ODER + 5 Bytes pro Element für Gleitpunktzahlen
ODER + 3 Bytes pro Element für Zeichenketten
 UND + 1 Byte pro Zeichen in jedem Stringelement
```

Beim Index kann es sich um ganzzahlige Konstanten, Variablen oder Rechenausdrücke handeln, bei denen das Ergebnis eine ganze Zahl ist. Getrennte Indizes (durch Kommata getrennt) werden für jede Felddimension benötigt. Ein Index kann einen Wert von Null bis zur Elementanzahl in der jeweiligen Felddimension haben. Werte außerhalb dieses Bereichs führen zu der BASIC-Fehlermeldung **?BAD SUBSCRIPT.** Feldnamen, Wertzuordnung und Datentyen sind z. B.:

```
A$(0)="GROSS SALES"          (Stringfeld)
MTH$(K%)="JAN"               (Stringfeld)
G2%(X)=5                     (Ganzzahlen Feld)
CNT%(G2%(X))=CNT%(1)-2       (Ganzzahlen Feld)
FP(12*K%)=24.8               (Gleitpunktfeld)
SUM(CNT%(1))=FP↑K%           (Gleitpunktfeld)
```

`A(5)=0` (Dem fünften Element im eindimensionalen Feld mit der Bezeichnung "A" wird der Wert 0 zugewiesen.)

`B(5,6)=0` (Dem Element in Reihe 5 und Spalte 6 des zweidimensionalen Feldes mit der Bezeichnung "B" wird der Wert 0 zugewiesen.)

`C(1,2,3)=0` (Dem Element in Reihe 1, Spalte 2 und Tiefe 3 der dritten Dimension mit der Bezeichnung "C" wird der Wert 0 zugewiesen.)

AUSDRÜCKE UND OPERATOREN

Ausdrücke werden durch Konstanten, Variablen und/oder Felder gebildet. Ein Ausdruck kann eine einzelne Konstante, ein einfacher Wert oder eine beliebige Feldvariable sein.
Er kann jedoch auch eine Kombination aus Konstanten, Variablen und Operatoren sein, die einen Einzelwert ergeben sollen. Die Operatoren werden nachfolgend erklärt. Es gibt zwei verschiedene Arten von Ausdrücken:

1) RECHENAUSDRÜCKE
2) STRINGS (ZEICHENKETTEN)

Ausdrücke enthalten normalerweise zwei oder mehr Daten, die Operanden genannt werden. Die Operanden werden voneinander getrennt. Im allgemeinen wird der Wert des Ausdrucks einem Variablennamen zugeordnet. Alle Beispiele für Konstanten und Variablen, die bisher gegeben wurde, waren gleichzeitig auch Beispiele für Ausdrücke.
Ein Operator ist ein spezielles Symbol, das für den BASIC-Interpreter in Ihrem COMMODORE 64 einen Vorgang angibt, der mit Variablen oder Konstantendaten ausgeführt werden soll. Ein oder mehr Operatoren zusammen mit einer oder mehreren Variablen und/oder Konstanten bilden einen Ausdruck. Rechen-, Vergleichs- und logische Operatoren werden vom COMMODORE 64 BASIC erkannt.

RECHENAUSDRÜCKE

Bei der Lösung von Rechenausdrücken ergibt sich eine ganze Zahl oder ein Gleitpunktwert. Die Rechenoperatoren (+, −, *, /, ↑) werden für Addition, Subtraktion, Multiplikation, Division und Exponentialberechnungen benutzt.

RECHENOPERATIONEN

Ein Rechenoperator bestimmt eine Rechenoperation, die mit den beiden Operanden beidseits des Operators ausgeführt wird. Rechenoperationen werden unter Verwendung von Gleitpunktzahlen ausgeführt. Ganze Zahlen werden zuvor in Gleitpunktzahlen umgewandelt. Das Ergebnis wird dann wieder zurück in eine ganze Zahl verwandelt, wenn es einem ganzzahligen Variablennamen zugeordnet ist.

ADDITION (+): Das Pluszeichen (+) gibt an, daß der Operand auf der rechten Seite zu dem auf der linken Seite addiert wird.

BEISPIELE:

```
2+2
A+B+C
X%+1
BR+10E-2
```

SUBTRAKTION (−): Das Minuszeichen (−) gibt an, daß der Operand auf der rechten Seite von dem auf der linken Seite subtrahiert wird.

BEISPIELE:

```
4-1
100-64
A-B
55-142
```

Dieses Minuszeichen kann auch als negatives Vorzeichen benutzt werden. Dies entspricht einer Subtraktion dieser Zahl von Null (0).

BEISPIELE:

```
-5
-9E4
-B
4- (-2) entspricht 4+2
```

MULTIPLIKATION: (*) Ein Sternchen (*) gibt an, daß der Operand auf der linken Seite mit dem auf der rechten Seite multipliziert wird.

BEISPIELE:

```
100*2
50*0
A*X1
R%*14
```

DIVISION (/): Der Schrägstrich (/) gibt an, daß der Operand auf der linken durch den auf der rechten Seite dividiert wird.

BEISPIELE:

```
10/2
6400/4
A/B
4E2/XR
```

EXPONENTIALBERECHNUNG (↑): Der Aufwärtspfeil gibt an, daß der Operand auf der linken Seite in die durch den Operand auf der rechten Seite (Exponent) angegebene Potenz erhoben wird. Ist der Operand auf der rechten Seite eine 2, so wird die Zahl auf der linken Seite zum Quadrat erhoben. Ist der Exponent eine 3, so wird die Zahl auf der linken Seite in die dritte Potenz erhoben usw. Der Exponent kann eine beliebige Zahl sein, solange sich beim Rechenergebnis eine zulässige Gleitpunktzahl ergibt.

BEISPIELE:

2↑2	Entspricht: 2*2
2↑3	Entspricht: 2*2*2
7↑4	Entspricht: 7*7*7*7
AB↑CD	
3↑−2	Entspricht: 1/3*1/3

VERGLEICHSOPERATOREN

Die Vergleichsoperatoren (<, =, >, <=, >=, <>) werden hauptsächlich zum Vergleich der Werte von zwei Operanden, aber auch zur Erzielung eines Rechenergebnisses benutzt. Die Vergleichsoperatoren und die logischen Operatoren (UND, ODER und NICHT) führen zu einer Richtig-/Falschbewertung von Ausdrücken, wenn sie bei Vergleichen benutzt werden. Ist der im Ausdruck angegebene Vergleich richtig, so wird dem Ergebnis die ganze Zahl −1 zugeordnet*; ist er falsch, so wird der Wert 0 zugeordnet. Es gibt folgende Vergleichsoperatoren:

<	Kleiner als
=	Gleich
>	Größer als
<=	Kleiner oder gleich
>=	Größer oder gleich
<>	Ungleich

---

* Achtung: Die Zuordnung von −1 bei einer wahren Aussage ist eine Charakteristik des COMMODORE 64.

BEISPIELE:

1=5−4	Richtig (−1)
14>66	Falsch (0)
15<=15	Richtig (−1)

Mit Vergleichsoperatoren können auch Strings verglichen werden. Bei Vergleichszwecken gilt für das Alphabet die Reihenfolge A<B<C<D usw. Strings werden durch Bewertung des Zusammenhangs zwischen den einzelnen Zeichen von links nach rechts verglichen (siehe Stringoperationen).

BEISPIELE:

"A" < "B"	Richtig (−1)
"X" = "YY"	Falsch (0)
BB$ <> CC$	

Numerische Daten können nur mit anderen numerischen Werten verglichen (oder diesen zugeordnet) werden. Das gleiche gilt für den Vergleich von Strings, da sonst die BASIC-Fehlermeldung **?TYPE MISMATCH** angezeigt wird. Beim Vergleich von numerischen Operanden wird zunächst der Wert von einem bzw. beiden Operanden von einer ganzen Zahl gegebenenfalls in einen Gleitpunktausdruck umgewandelt. Dann wird der Zusammenhang zwischen den Gleitpunktwerten entsprechend einer Richtig-/Falschbeurteilung bewertet.
Nach allen Vergleichen erhalten Sie eine ganze Zahl, unabhängig davon, welcher Datentyp für den Operanden gilt (selbst wenn beides Zeichenketten sind). Aus diesem Grund kann der Vergleich von zwei Operanden als Operand bei Berechnungen benutzt werden. Das Ergebnis lautet −1 oder 0 und kann beliebig weiter verwendet werden, außer als Divisor, da eine Division durch Null unzulässig ist.

LOGISCHE OPERATOREN

Die logischen Operatoren (AND, OR, NOT) können zur Änderung der Bedeutung von Vergleichsoperatoren oder für Rechenergebnisse benutzt werden. Logische Operatoren ergeben andere Ergebnisse als −1 und 0; bei der Richtig-/Falschbewertung wird jedes Ergebnis, das nicht 0 ist, als richtig angesehen.
Die logischen Operatoren (gelegentlich auch Boole'sche Operatoren genannt) können auch für logische Operationen von einzelnen Binärstellen (Bits) bei zwei Operanden benutzt werden. Wird jedoch der Operator NOT benutzt, so erfolgt die Operation nur mit dem einen Operanden auf der rechten Seite.
Die Operanden müssen ganze Zahlen (−32768 bis +32767) sein (Gleitpunktzahlen werden in ganze Zahlen umgewandelt), und beim Ergebnis ergibt sich wieder eine ganze Zahl.

Logische Operationen beziehen sich immer auf die entsprechenden Bits der beiden Operanden. Beim logischen AND ist das Bit-Ergebnis nur 1, wenn beide Operandenbits 1 sind. Beim logischen OR kann das Bit-Ergebnis 1 sein, wenn nur ein Operand 1 ist. Das logische NOT ist der entgegengesetzte Wert jedes Bits als einzelner Operand. D. h., es bedeutet "wenn es NOT 1 ist, dann ist es 0. Wenn es NOT 0 ist, dann ist es 1."
Das ausschließende XOR hat keinen logischen Operator, sondern wird als Teil der Anweisung WAIT ausgeführt. Beim ausschließenden ODER ist das Ergebnis 0, wenn die Bits von zwei Operanden gleich sind. Ansonsten lautet das Ergebnis 1.
Logische Operationen werden durch Anweisungsgruppen definiert, die alle zusammen die in Tabelle 1.2. gezeigte Boole'sche "WAHRHEITSTABELLE" bilden.

Tabelle 1.2. Boole'sche Wahrheitstabelle

Das Ergebnis der AND-Operation lautet nur 1, wenn beide Bits 1 sind:

1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

Das Ergebnis der OR-Operation lautet 1, wenn eins der Bits 1 ist:

1 OR 1 = 1
0 OR 1 = 1
1 OR 0 = 1
0 OR 0 = 0

Durch die NOT-Operation werden alle Bits logisch komplementiert:

NOT 1 = 0
NOT 0 = 1

Das ausschließende ODER (XOR) ist Teil der Anweisung WAIT:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Die logischen Operatoren AND, OR und NOT geben eine Boole'sche Rechenoperation an, die mit zwei Operanden ausgeführt werden. Bei NOT wird nur der Operand auf der rechten Seite berücksichtigt. Logische Operationen (oder Boole'sche Rechenoperationen) werden erst ausgeführt, wenn alle Rechen- und Vergleichsoperationen in einem Ausdruck beendet sind.

BEISPIELE:

IF A=100 AND B=100 THEN 10	(Wenn sowohl A als auch B den Wert 100 haben, ist das Ergebnis richtig)
A=96 AND 32: PRINT A	(A = 32)
IF A=100 OR B=100 THEN 20	(Wenn A oder B 100 ist, dann ist das Ergebnis richtig)
A=64 OR 32: PRINT A	(A = 96)
IF NOT X<Y THEN 30	(Wenn X>=Y, ist das Ergebnis richtig)
X= NOT 96	(Das Ergebnis ist −97 (Zweierkomplement)

PRIORITÄT DER OPERATIONEN

Bei allen Ausdrücken werden die verschiedenen Operationen entsprechend einer festgelegten Prioritätenfolge ausgeführt. D. h., bestimmte Operationen werden vor anderen durchgeführt. Die normale Reihenfolge kann geändert werden, indem man zwei oder mehr Operanden in Klammern einschließt ( ) und so einen "Unterausdruck" bildet. Die Werte eines Ausdrucks in Klammern werden auf einen einzelnen Wert reduziert, ehe die Teile außerhalb der Klammern bearbeitet werden.
Werden in Ausdrücken Klammern benutzt, so ist darauf zu achten, daß stets die gleiche Anzahl an linken und rechten Klammern auftritt. Ansonsten wird die BASIC-Fehlermeldung **?SYNTAX ERROR** angezeigt.
Ausdrücke, die Operanden in Klammern enthalten, können ihrerseits auch in Klammern eingeschlossen werden und so ganze Ausdrücke in mehreren Ebenen bilden. Dies wird Verschachtelung genannt. Klammern können in Ausdrücken auf max. zehn Ebenen verschachtelt werden – zehn Klammersätze.
Hierbei wird die ganz innen liegende Klammer zuerst aufgelöst. Ausdrücke sind z. B.:

```
A+B
C↑(D+E)/2
((X−C↑(D+E)/2)*10)+1
GG$>HH$
JJ$+"MORE"
K%=1 AND M<>X
K%=2 OR (A=B AND M<X)
NOT (D=E)
```

Der BASIC-Interpreter führt normalerweise zuerst die Rechenoperationen durch. Danach folgen Vergleichsoperationen und zuletzt logische Operationen.
Sowohl für arithmetische als auch für logische Operatoren gilt eine Prioritätenfolge. Vergleichsoperatoren haben keine solche Folge und werden bei der Ausdrucksbewertung von links nach rechts so ausgeführt, wie sie erscheinen.
Wenn für die anderen Operatoren in einem Ausdruck keine besondere Priorität gilt, so werden sie von links nach rechts ausgeführt. Beim Auflösen einer Klammer wird die normale Prioritätenfolge aufrechterhalten. Die Priorität für arithmetische und logische Operationen wird in Tabelle 1.3., beginnend bei der ersten Priorität, gezeigt.

Tabelle 1.3. Priorität von Ausdrucks-Operationen

Operator	Beschreibung	Beispiel
↑	Exponentialrechnung	BASE↑EXP
−	Negation (negatives Vorzeichen)	−A
* /	Multiplikation Division	AB * CD EF / GH
+ −	Addition Subtraktion	CNT + 2 JK − PQ
> = <	Vergleichsoperationen	A <= B
NOT	Logisches Nicht (Ganzzahliges Zweierkomplement)	NOT K%
AND	Logisches UND	JK AND 128
OR	Logisches ODER	PQ OR 15

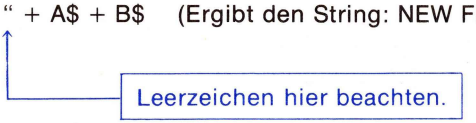
ZEICHENKETTENOPERATIONEN

Zeichenketten können mit den gleichen Vergleichsoperatoren (=, <>, <=, >=, <, >) wie Zahlen verglichen werden. Bei dem Zeichenkettenvergleich wird jeweils ein Zeichen (von links nach rechts) von jeder Zeichenkette genommen und jede Zeichengrundposition vom CBM-Zeichensatz bewertet. Sind die Zeichencodes gleich, so sind auch die Zeichen gleich. Bei abweichenden Zeichencodes ist das Zeichen mit niedrigerer Codenummer auch niedriger im Zeichensatz.
Der Vergleich endet, wenn das Ende einer Zeichenkette erreicht ist. Stimmen alle anderen Punkte überein, so ist die kürzere Zeichenkette niedriger als die längere. Führende oder nachstehende Leerzeichen sind signifikant.
Unabhängig von den Datentypen ist das Ergebnis des Vergleichs stets eine ganze Zahl. Dies gilt selbst dann, wenn beide Operanden Zeichenketten sind. Aus diesem Grund kann ein Vergleich von zwei Zeichenkettenoperanden als Operand bei Berechnungen benutzt werden.
Das Ergebnis ist −1 oder 0 (richtig oder falsch) und kann beliebig eingesetzt werden. Ausgeschlossen ist lediglich eine Division, da eine Teilung durch 0 unzulässig ist.

STRINGS

Ausdrücke werden so behandelt, als ob ein impliziertes "<>0" folgt. D. h., wenn ein Ausdruck richtig ist, werden die nächsten BASIC-Anweisungen auf der gleichen Programmzeile ausgeführt. Ist der Ausdruck falsch, wird der Rest der Zeile überlesen und erst die nächste Programmzeile ausgeführt.
Genau wie mit den Zahlen kann man auch mit Stringvariablen Verknüpfungen durchführen. Der einzige String-Rechenoperator, der vom CBM BASIC erkannt wird, ist ein Pluszeichen (+), das für die Verkettung von Strings benutzt wird. Bei der Verkettung von Strings wird die Kette auf der rechten Seite des Pluszeichens an die auf der linken Seite angefügt. Sie ist eine dritte Zeichenkette. Das Ergebnis kann sofort angezeigt, beim Vergleich benutzt oder einem Variablennamen zugeordnet werden. Wird ein String mit einem numerischen Wert verglichen (oder gleichgesetzt) bzw. umgekehrt, so wird die BASIC-Fehlermeldung **?TYPE MISMATCH** angezeigt. Strings und Verkettungen sind z. B.:

```
10 A$="FILE" : B$="NAME"
20 NAM$ = A$ + B$              (Ergibt den String: FILENAME)
30 RES$ = "NEW " + A$ + B$     (Ergibt den String: NEW FILENAME)
```



PROGRAMMIERTECHNIKEN

DATENUMSETZUNG

Der CBM BASIC-Interpreter kann gegebenenfalls einen numerischen Wert von einer ganzen Zahl in eine Gleitpunktzahl oder umgekehrt umsetzen. Hierbei gelten folgende Regeln:

- Alle arithmetischen und Vergleichsoperationen werden im Gleitpunktformat ausgeführt. Ganze Zahlen werden vor der Verarbeitung der Ausdrücke in Gleitpunktzahlen umgewandelt, und das Ergebnis wird dann wieder in eine ganze Zahl umgesetzt. Logische Operationen wandeln ihre Operanden in ganze Zahlen um und ergeben ein ganzzahliges Resultat.
- Wird eine numerische Variable einem numerischen Wert anderer Art gleichgesetzt, so wird die Zahl umgesetzt und als im Variablennamen angegebener Datentyp gespeichert.
- Wird ein Gleitpunktwert in eine ganze Zahl umgewandelt, so werden die Nachkommastellen abgeschnitten, und das ganzzahlige Ergebnis ist kleiner oder gleich dem Gleitpunktwert. Liegt das Ergebnis außerhalb des Bereichs +32767 bis −32768, wird die BASIC-Fehlermeldung **?ILLEGAL QUANTITY** angezeigt.

VERWENDUNG DER EINGABEANWEISUNG

Nun wissen Sie also, was Variablen sind, und sind in der Lage, zusammen mit der Eingabe-Anweisung INPUT zu programmieren.
Bei unserem ersten Beispiel können Sie sich eine Variable als eine Art "Speicher" vorstellen, in den der COMMODORE 64 die gegebenen Antworten speichert. Beim Schreiben eines Programms, bei dem ein Name eingegeben werden soll, können Sie dem über die Tastatur eingegebenen Namen die Variable N$ zuordnen. Nun wird jedesmal, wenn Sie in Ihr Programm PRINT N$ eingeben, der COMMODORE 64 automatisch den eingegebenen Namen anzeigen.
Geben Sie über die Tastatur des COMMODORE 64 NEW ein, und drücken Sie die Taste RETURN. Probieren Sie dieses Beispiel aus:

```
10 PRINT "IHR NAME":INPUT N$
20 PRINT "HELLO," N$
```

In diesem Beispiel haben Sie die Variable N benutzt, um sich daran zu erinnern, daß diese Variable für "NAME" steht. Das Dollarzeichen ($) zeigt dem Computer an, daß Sie eine Stringvariable benutzen. Es muß unbedingt zwischen zwei Variablenarten unterschieden werden:

1) NUMERISCHE VARIABLE
2) STRINGVARIABLE

Sie erinnern sich sicherlich noch daran, daß numerische Variablen zum Speichern von Zahlenwerten wie z. B. 1, 100, 4000, usw. benutzt werden. Eine numerische Variable kann ein einzelner Buchstabe (A), zwei Buchstaben (AB), ein Buchstabe und eine Zahl (A1) oder zwei Buchstaben und eine Zahl (AB1) sein. Durch die Verwendung kürzerer Variablen wird nicht soviel Speicherkapazität vergeben. Nützlich ist es auch, Buchstaben und Zahlen für unterschiedliche Kategorien im gleichen Programm zu benutzen (A1, A2, A3). Wenn Sie als Antwort ganze Zahlen und nicht Zahlen mit Dezimalpunkten wünschen, brauchen Sie lediglich nach dem Variablennamen ein Prozentzeichen (%) einzugeben (AB%, A1%, usw.).
Nun wollen wir uns einige Beispiele ansehen, bei denen verschiedene Variablenarten und Ausdrücke mit der Anweisung INPUT benutzt werden.
Eine Programmzeile wird durch Drücken der RETURN-Taste an den Computer übergeben.

```
10 PRINT "ZAHL EINGEBEN":INPUT A
20 PRINT A
```

```
10 PRINT "WORT EINGEBEN":INPUT A$
20 PRINT A$
```

```
10 PRINT "ZAHL EINGEBEN":INPUT A
20 PRINT A "MAL 5 IST GLEICH" A*5
```

Anmerkung: Beispiel 3 zeigt, daß MELDUNGEN oder AUFFORDERUNGEN in Anführungszeichen (" ") stehen und die Variablen außerhalb dieser Anführungszeichen liegen. Beachten Sie auch, daß in Zeile 20 die Variable A und dann die Meldung "MAL 5 IST GLEICH" und abschließend die Berechnung A*5 angezeigt wird.

Berechnungen sind in den meisten Programmen wichtig. Sie haben die Wahl, feste Zahlen oder Variablen zu benutzen. Beim Arbeiten mit vom Benutzer vorgegebenen Zahlen müssen jedoch numerische Variablen benutzt werden. Zunächst wird der Benutzer zur Eingabe von zwei Zahlen aufgefordert:

```
10 PRINT "2 ZAHLEN EINGEBEN":INPUT A:INPUT B
```

BEISPIEL FÜR EINEN HAUSHALTSPLAN ÜBER EINKÜNFTE/AUSGABEN

```
5 PRINT "□"   SHIFT CLR/HOME
10 PRINT"MONTHLY INCOME":INPUT IN
20 PRINT
30 PRINT"EXPENSE CATEGORY 1":INPUT E1$
40 PRINT"EXPENSE AMOUNT":INPUT E1
50 PRINT
60 PRINT"EXPENSE CATEGORY 2":INPUT E2$
70 PRINT"EXPENSE AMOUNT":INPUT E2
80 PRINT
90 PRINT"EXPENSE CATEGORY 3":INPUT E3$
100 PRINT"EXPENSE AMOUNT":INPUT E3
110 PRINT "□"   SHIFT CLR/HOME
120 E=E1+E2+E3
130 EP=E/IN
140 PRINT"MONTHLY INCOME: $"IN
150 PRINT"TOTAL EXPENSES: $"E
160 PRINT"BALANCE EQUALS: $"IN-E
170 PRINT
180 PRINT E1$"="(E1/E)*100"% OF TOTAL EXPENSES"
190 PRINT E2$"="(E2/E)*100"% OF TOTAL EXPENSES"
200 PRINTE3$"="(E3/E)*100"% OF TOTAL EXPENSES"
210 PRINT
220 PRINT"YOUR EXPENSES="EP*100"% OF YOUR TOTAL
INCOME"
230 FOR X=1TO5000:NEXT:PRINT
240 PRINT"REPEAT? (Y OR N)":INPUT Y$:IF Y$="Y"THEN5
250 PRINT "□":END
          SHIFT CLR/HOME
```

Anmerkung: IN darf nicht 0 sein, und E1, E2, E3 können NICHT alle gleichzeitig 0 sein.

Zeile(n)	Beschreibung
5	Löscht den Bildschirm
10	Anweisung PRINT/INPUT
20	Einfügen einer Leerzeile
30	Ausgabekategorie 1 = E1$
40	Ausgabebetrag = E1.
50	Einfügen einer Leerzeile
60	Ausgabekategorie 2 = E2$
70	Ausgabebetrag 2 = Es.
80	Einfügen einer Leerzeile
90	Ausgabekategorie 3 = E3$
100	Ausgabebetrag 3 = E3
110	Löscht den Bildschirm
120	Addition der Ausgabebeträge = E
130	Berechnung von Ausgaben/Einkünften %
140	Anzeige des Einkommens
150	Anzeige der Gesamtausgaben
160	Anzeige von Einkommen – Ausgaben
170	Einfügen einer Leerzeile
180–200	Zeile 180–200 berechnet, wieviel % jeder Ausgabebetrag von den Gesamtausgaben beträgt
210	Einfügen einer Leerzeile
220	Anzeige von Einkünften/Ausgaben in %
230	Warteschleife

Multiplizieren Sie nun, wie nachstehend in Zeile 20 gezeigt, zwei Zahlen miteinander, um die neue Variable C zu erhalten:

```
20 C=A*B
```

Um das Ergebnis auf dem Bildschirm anzuzeigen, ist folgende Zeile einzugeben:

```
30 PRINT A "MAL" B "IST GLEICH" C
```

Danach RUN eingeben und die RETURN-Taste drücken. Bitte beachten Sie, daß die Meldungen im Gegensatz zu den Variablen in Anführungszeichen stehen.
Nehmen wir nun an, Sie möchten ein Dollarzeichen ($) vor der durch die Variable C gekennzeichneten Zahl. Das $ muß in Anführungszeichen und vor der Variablen C angezeigt werden. Um $ in Ihr Programm einzufügen, drücken Sie die Tasten RUN/STOP und RESTORE. In Zeile 40 geben Sie nun wie folgt ein:

```
40 PRINT "$" C
```

Nun die Taste RETURN drücken, danach RUN eingeben und wieder RETURN drücken.
Das Dollarzeichen steht in Anführungszeichen, da die Variable C nur eine Zahl angibt und keine $ enthalten kann. Lautete die durch C dargestellte Zahl 100, würde auf dem Bildschirm des COMMODORE 64 $ 100 angezeigt. Wurde jedoch versucht, PRINT $C ohne Anführungszeichen zu benutzen, wird die Meldung **?SYNTAX ERROR** angezeigt.
Ein letzter Tip: Sie können eine Variable zur Darstellung eines Dollarzeichens erstellen, durch die Sie dann $ ersetzen können, wenn Sie es mit numerischen Variablen benutzen wollen. Z. B.:

```
10 Z$="$"
```

Immer wenn Sie jetzt ein Dollarzeichen brauchen, können Sie die Stringvariable Z$ benutzen. Probieren Sie folgendes:

```
10 Z$="$":INPUT A
20 PRINT Z$A
```

Zeile 10 bestimmt $ als die Stringvariable Z$ und gibt danach eine Zahl A ein. In Zeile 20 wird Z$ ($) neben A (Zahl) angezeigt.

ARBEITEN MIT DER GET-ANWEISUNG

In den meisten einfachen Programmen wird die Anweisung INPUT benutzt, um vom Benutzer Daten zu bekommen. Bei komplexeren Anwendungen zum Schutz vor Schreibfehlern usw. bietet Ihnen die Anweisung GET größere Flexibilität. Dieser Abschnitt zeigt Ihnen, wie Sie mit der Anweisung GET umgehen müssen und für Ihre Programme zusätzliche Bildschirm-Aufbereitungsfunktionen erzielen.
Der COMMODORE 64 hat einen Tastaturpuffer mit einer Kapazität von 10 Zeichen. D. h., auch wenn der Computer gerade mit einer Operation beschäftigt und daher nicht zum Lesen Ihrer Eingabe bereit ist, können Sie noch max. 10 Zeichen eingeben, die sofort nach Beendigung der derzeitigen Operation benutzt werden. Geben Sie als Beispiel folgendes Programm in Ihren COMMODORE 64 ein:

```
10 TI$="000000"
20 IF TI$ < "000015" THEN 20
```

Geben Sie nun RUN ein, drücken Sie die Taste RETURN, und geben Sie mit der Tastatur während der Programmausführung HELLO ein.
Bitte beachten Sie, daß ca. 15 s lang nach Beginn des Programms gar nichts passiert. Erst dann erscheint die Meldung HELLO auf dem Bildschirm. Stellen Sie sich vor, Sie stehen in einer Schlange vor einem Kino an. Die erste Person in dieser Schlange ist auch die erste, die eine Karte bekommt und dann aus der Schlange tritt. Entsprechend bekommt die letzte Person in der Schlange auch erst zuletzt die Karte. Die Anweisung GET ist so etwas wie ein Kartenkontrolleur. Zunächst wird geprüft, ob irgendwelche Zeichen "in Schlange stehen" (d. h., ob irgendwelche Tasten angeschlagen wurden).
Lautet die Antwort ja, dann wird dieses Zeichen der entsprechenden Variablen zugeordnet. Wurde keine Taste gedrückt, wird der Variablen ein leerer Wert zugeordnet.
Bitte beachten Sie hierbei unbedingt, daß stets nur 10 Zeichen in den Puffer eingegeben werden können. Alle übrigen Zeichen werden nicht berücksichtigt.
Da die Anweisung GET auch dann weiterläuft, wenn keine Zeichen eingegeben werden, ist es sinnvoll, diese Anweisung in eine Schleife einzugeben, so daß stets bis zum Anschlagen einer Taste oder dem Empfang eines Zeichens während des Programms gewartet werden muß.
Nachstehend sehen Sie eine Anwendung für die Anweisung GET. Zum Löschen des vorherigen Programms geben Sie NEW ein und drücken auf die RETURN-Taste.

```
10 GET A$ : IF A$ = "" THEN 10
```

Bitte beachten Sie, daß zwischen den Anführungszeichen ("") in dieser Zeile KEIN LEERZEICHEN ist. Dies zeigt einen leeren Wert (LEERSTRING) an und schickt das Programm in einer kontinuierlichen Schleife zurück zur Anweisung GET, bis eine Taste angeschlagen wird. Danach wird das Programm mit der Zeile nach Zeile 10 fortgesetzt. Fügen Sie folgende Zeile in Ihr Programm ein:

```
100 PRINT A$;: GOTO 10
```

Lassen Sie nun das Programm laufen. Bitte beachten Sie, daß auf dem Bildschirm kein Cursor ■ erscheint. Die von Ihnen eingegebenen Zeichen werden jedoch auf dem Bildschirm angezeigt. Dieses zweizeilige Programm kann wie nachstehend gezeigt als Teil eines Editor-Programms benutzt werden.
Es gibt viele Dinge, die Sie mit einem Bildschirm-Editor tun können. Es kann ein blinkender Cursor angezeigt werden. Sie können vermeiden, daß durch das Betätigen bestimmter Tasten wie z. B. CLR/HOME aus Versehen der ganze Bildschirm gelöscht wird.

Und Sie können sogar mit Ihren Funktionstasten ganze Wörter oder Sätze darstellen. Folgende Programmzeilen belegen die Funktionstasten. Denken Sie daran, daß dies lediglich ein Programmanfang ist und daß Sie das Programm ganz entsprechend Ihren persönlichen Bedürfnissen gestalten können.

```
10 GET A$ : IF A$ ="" THEN 10
20 IF A$ = CHR$(133) THEN POKE 53280,8:GOTO 10
30 IF A$ = CHR$(134) THEN POKE 53281,4:GOTO 10
40 IF A$ = CHR$(135) THEN A$="DEAR SIR:"+CHR$(13)
50 IF A$ = CHR$(036) THEN A$="SINCERELY,"+CHR$(13)
100 PRINT A$;: GOTO 10
```

Die Zahlen in Klammern stammen aus der Zeichen-Code-Tabelle in Anhang C. Diese Tabelle führt für jedes Zeichen eine bestimmte Zahl auf. Die vier Funktionstasten werden benutzt, um die durch die Anweisungen dargestellten Aufgaben auszuführen, die in jeder Zeile nach dem Wort THEN folgen. Durch Änderung der Zahlen in den Klammern können Sie verschiedene Tasten belegen. Verschiedene Anweisungen werden ausgeführt, wenn die Information nach der Anweisung THEN geändert wird.

KOMPRIMIEREN VON BASIC-PROGRAMMEN

Durch Komprimieren (engl.: crunching) können Sie die max. mögliche Anzahl an Anweisungen in Ihrem Programm eingeben. Außerdem können Sie hierdurch die Programmgröße reduzieren. Bei dem Schreiben von Programmen, bei denen die Eingabe von Daten, wie z. B. Zahlen oder Text, erforderlich ist, läßt ein kürzeres Programm mehr Speicherkapazität für die Daten übrig.

SCHLÜSSELWORTABKÜRZUNGEN

Eine Liste der Schlüsselwortabkürzungen finden Sie in Anhang A. Mit Hilfe dieser Abkürzungen können Sie bedeutend mehr Informationen in eine Zeile eingeben. Die am häufigsten eingesetzte Abkürzung ist das Fragezeichen (?), das die BASIC-Abkürzung für den Befehl PRINT ist. Bei der Auflistung eines Programms, das Abkürzungen enthält, zeigt der COMMODORE 64 Ihnen allerdings die Schlüsselwörter in ganzer Länge an. Enthält eine Programmzeile mit ausgeschriebenen Schlüsselwörtern mehr als 80 Zeichen (zwei Bildschirmzeilen) und soll geändert werden, so müssen Sie vor der Speicherung diese Zeile neu mit Abkürzungen eingeben. Bei Programmspeicherung werden BASIC-Schlüsselwörter vom COMMODORE 64 in Zeichen (Tokens) umgesetzt. Normalerweise werden Abkürzungen nach dem Schreiben eines Programms eingefügt, wenn dieses vor der Speicherung nicht mehr aufgelistet wird.

VERKLEINERN DER PROGRAMMZEILENNUMMERN

Die meisten Programmierer beginnen ihre Programme bei Zeile 10 und numerieren die nachfolgenden Zeilen dann im Zehner-Abstand durch (d. h. 100, 110, 120). Auf diese Weise können nach der Programmentwicklung Extra-Anweisungszeilen eingefügt werden (111, 112, usw.). Um das Programm kürzer zu gestalten, *können für die Zeilennummern die niedrigsten Nummern gewählt werden, die möglich sind* (1, 2, 3). Denken Sie daran, daß längere Zahlen mehr Speicherkapazität beanspruchen als kürzere. So benötigt die Zahl 100 z. B. 3 Bytes (1 Byte für jede Ziffer), die Zahl 1 hingegen nur 1 Byte.

EINGABE VON MEHREREN ANWEISUNGEN IN JEDE ZEILE

In eine numerierte Zeile Ihres Programms können Sie mehrere Anweisungen getrennt durch einen Doppelpunkt eingeben. Hierbei gilt lediglich die Begrenzung, daß die Anweisungen in jeder Zeile einschließlich Doppelpunkten nicht die Standardzeilenlänge von 80 Zeichen überschreiten. Nachfolgend stehen zwei Programmbeispiele vor und nach der Verkürzung.

Vor der Verkürzung

```
10 PRINT"HELLO . . .";
20 FOR T=1 TO 500:NEXT
30 PRINT "HELLO, AGAIN . . ."
40 GOTO 10
```

Nach der Verkürzung

```
10 PRINT "HELLO . . .";:FORT=1TO
   500:NEXT:PRINT"HELLO,
   AGAIN . . .":GOTO10
```

LÖSCHEN DER REM-ANWEISUNGEN

REM-Anweisungen sind eine nützliche Hilfe, um sich selbst oder anderen Programmierern einen bestimmten Programmteil zu erläutern. Wenn das Programm jedoch vollständig und einsatzbereit ist, werden Sie diese REM-Anweisungen wahrscheinlich nicht mehr brauchen.
Sie können daher Speicherkapazität einsparen, indem Sie diese Anweisungen löschen. Soll eine Programmstruktur zukünftig überarbeitet oder genau untersucht werden, so sollten Sie eine Kopie des Programms mit den REM-Anweisungen anfertigen.

ARBEITEN MIT VARIABLEN

Wird eine Zahl, ein Wort oder ein Satz wiederholt in Ihrem Programm benutzt, so sollten Sie diese Wörter oder Zahlen am besten in Variablen ablegen. Zahlen

können einfachen Buchstaben zugeordnet werden. Zur Angabe von Wörtern und Sätzen wählen Sie Stringvariablen mit einem Buchstaben und einem Dollarzeichen. Hier ein Beispiel:

Vor der Verkürzung

```
10 POKE 54296,15
20 POKE 54276,33
30 POKE 54273,10
40 POKE 54273,40
50 POKE 54273,70
60 POKE 54296,0
```

Nach der Verkürzung

```
10 V=54296:F=54273
20 POKEV,15:POKE54276,33
30 POKEF,10:POKEF,40:POKEF,70
40 POKEV,0
```

ARBEITEN MIT READ- UND DATA-ANWEISUNGEN

Umfangreiche Datenmengen können einzeln eingegeben werden . . . Sie haben jedoch auch die Möglichkeit, den Anweisungsteil des Programms zusammenzufassen und alle zu bearbeitenden Daten in einer langen Liste wiederzugeben, die DATA-Anweisung genannt wird. Auf diese Weise lassen sich besonders gut große Zahlenlisten in einem Programm unterbringen.

ARBEITEN MIT FELDERN UND MATRIZEN

Mit Feldern und Matrizen können, wie bei den DATA-Anweisungen, umfangreiche Datenmengen verarbeitet werden. Der Unterschied besteht lediglich darin, daß bei Feldern mehrdimensionale Listen möglich sind.

VERMEIDUNG VON LEERZEICHEN

Am einfachsten können Sie die Programmgröße dadurch reduzieren, daß Sie alle Leerzeichen vermeiden. Auch wenn in den Beispielprogrammen zur besseren Lesbarkeit häufig Leerzeichen enthalten sind, brauchen Sie doch bei der tatsächlichen Programmierung keinerlei Leerzeichen und können entsprechend Speicherkapazität sparen.

ARBEITEN MIT GOSUB-ROUTINEN

Wird eine bestimmte Zeile oder Anweisung wiederholt benutzt, so ist es ratsamer, von mehreren Stellen des Programms zu dieser Zeile über die GOSUB-Anweisung zu gehen, als die ganze Zeile oder Anweisung jedesmal neu zu schreiben.

ARBEITEN MIT TAB UND SPC

Statt eine Zeichenposition auf dem Bildschirm über mehrere Cursor-Befehle zu bestimmen, ist es häufig ratsamer, hierzu die Anweisungen TAB und SPC zu benutzen.

KAPITEL 2

BASIC-VOKABULAR

- Einführung
- BASIC-Schlüsselwörter, Abkürzungen und Funktionsarten
- Beschreibung der BASIC-Schlüsselwörter (alphabetisch)
- Tastatur und Merkmale des COMMODORE 64
- Bildschirm-Editor

EINFÜHRUNG

In diesem Kapitel werden die CBM-BASIC-Schlüsselwörter beschrieben. Zunächst geben wir eine leicht lesbare Liste der Schlüsselwörter mit ihren Abkürzungen und der jeweiligen Bildschirmanzeige. Danach werden Syntax und Funktion jedes Schlüsselworts genau beschrieben und anhand von Beispielen gezeigt, wie Sie diese Schlüsselwörter in Ihrem Programm nutzen können.
Beim COMMODORE 64 BASIC können Sie die meisten Schlüsselwörter abkürzen. Hierzu werden so viele Buchstaben des Schlüsselworts eingegeben, wie zur Unterscheidung von den übrigen Wörtern erforderlich sind, und bei der Eingabe des letzten Buchstaben oder des letzten Graphik-Zeichens die Taste SHIFT gedrückt gehalten.
Abkürzungen in Programmen sparen keinerlei Speicherkapazität, da alle Schlüsselwörter vom BASIC-Interpreter als einzelne Zeichen (Tokens) dargestellt werden. Bei der Auflistung eines Programms mit Abkürzungen erscheinen alle Schlüsselwörter in voll ausgeschriebener Form. Mit Hilfe von Abkürzungen können mehr Anweisungen in eine Programmzeile eingegeben werden, selbst wenn sie nicht auf die logische Bildschirmzeile von 80 Zeichen passen. Der Bildschirm-Editor arbeitet auf der Basis einer 80-Zeichen-Zeile. D. h., wenn in einer Zeile Abkürzungen von mehr als 80 Zeichen benutzt werden, dann kann diese Zeile beim Auflisten nicht editiert werden. Sie müssen daher entweder die ganze Zeile einschließlich sämtlicher Abkürzungen neu eingeben oder die eine Zeile in zwei Zeilen mit jeweils eigener Zeilennummer unterteilen.
Tabelle 2.1. gibt eine vollständige Liste aller Schlüsselwörter, Abkürzungen und der entsprechenden Bildschirmanzeige. Danach folgt eine alphabetische Beschreibung der Anweisungen, Befehle und Funktionen, die mit Ihrem COMMODORE 64 möglich sind.
Ferner werden die BASIC-Funktionen des BASIC-Interpreters erklärt.
Diese integrierten Funktionen können als direkte Anweisungen oder in einem beliebigen Programm ohne weitere Bestimmung der Funktion benutzt werden. Dies gilt NICHT für vom Benutzer definierte Funktionen. Das Ergebnis der integrierten BASIC-Funktionen kann direkt ausgegeben oder einem geeigneten Variablennamen zugeordnet werden. Es gibt zwei verschiedene Arten von BASIC-Funktionen.

1) NUMERISCH
2) STRING (ZEICHENKETTE)

Argumente von integrierten Funktionen sind stets in Klammern eingeschlossen ( ). Die Klammern folgen unmittelbar nach dem Funktionsschlüsselwort, und es stehen KEINE LEERZEICHEN zwischen dem letzten Buchstaben des Schlüsselworts und der linken Klammer (.

Der Typ des benötigten Arguments wird im allgemeinen durch den Datentyp des Resultats bestimmt. Funktionen, bei denen das Ergebnis ein String ist, werden durch ein Dollarzeichen ($) als letztem Schlüsselwortbuchstaben identifiziert. In einigen Fällen enthalten Stringfunktionen ein oder mehrere numerische Argumente. Numerische Funktionen nehmen gegebenenfalls eine Umkehrung von ganzen Zahlen und Gleitpunktzahlen vor. In der nachfolgenden Beschreibung wird der Datentyp bei jeder Funktionsbezeichnung gezeigt. Die Argumenttypen werden ebenfalls durch das Anweisungsformat gegeben.

TABELLE 2.1. COMMODORE 64 BASIC-SCHLÜSSELWÖRTER

BEFEHL	ABKÜRZUNG	BILDSCHIRM-DARSTELLUNG	FUNKTIONSTYP
ABS	A SHIFT B	A	NUMERISCH
AND	A SHIFT N	A	
ASC	A SHIFT S	A	NUMERISCH
ATN	A SHIFT T	A	NUMERISCH
CHR$	C SHIFT H	C	ZEICHENKETTE
CLOSE	CL SHIFT O	CL	
CLR	C SHIFT L	C	
CMD	C SHIFT M	C	
CONT	C SHIFT O	C	
COS	keine	COS	NUMERISCH
DATA	D SHIFT A	D	
DEF	D SHIFT E	D	
DIM	D SHIFT I	D	

BEFEHL	ABKÜRZUNG	BILDSCHIRM-DARSTELLUNG	FUNKTIONSTYP
END	E SHIFT N	E 	
EXP	E SHIFT X	E 	NUMERISCH
FN	keine	FN	
FOR	F SHIFT O	F 	
FRE	F SHIFT R	F 	NUMERISCH
GET	G SHIFT E	G 	
GET#	keine	GET#	
GOSUB	GO SHIFT S	GO 	
GOTO	G SHIFT O	G 	
IF	keine	IF	
INPUT	keine	INPUT	
INPUT#	I SHIFT N	I 	
INT	keine	INT	NUMERISCH
LEFT$	LE SHIFT F	LE 	ZEICHENKETTE
LEN	keine	LEN	NUMERISCH
LET	L SHIFT E	L 	
LIST	L SHIFT I	L 	
LOAD	L SHIFT O	L 	
LOG	keine	LOG	NUMERISCH

BEFEHL	ABKÜRZUNG	BILDSCHIRM-DARSTELLUNG	FUNKTIONSTYP
MID$	M SHIFT I	M	ZEICHENKETTE
NEW	keine	NEW	
NEXT	N SHIFT E	N	
NOT	N SHIFT O	N	
ON	keine	ON	
OPEN	O SHIFT P	O	
OR	keine	OR	
PEEK	P SHIFT E	P	NUMERISCH
POKE	P SHIFT O	P	
POS	keine	POS	NUMERISCH
PRINT	?	?	
PRINT#	P SHIFT R	P	
READ	R SHIFT E	R	
REM	keine	REM	
RESTORE	RE SHIFT S	RE	
RETURN	RE SHIFT T	RE	
RIGHT$	R SHIFT I	R	ZEICHENKETTE
RND	R SHIFT N	R	NUMERISCH
RUN	R SHIFT U	R	

BEFEHL	ABKÜRZUNG	BILDSCHIRM-DARSTELLUNG	FUNKTIONSTYP
SAVE	S SHIFT A	S 	
SGN	S SHIFT G	S 	NUMERISCH
SIN	S SHIFT I	S 	NUMERISCH
SPC(	S SHIFT P	S 	ZEICHENKETTE
SQR	S SHIFT Q	S 	NUMERISCH
STATUS	ST	ST	NUMERISCH
STEP	ST SHIFT E	ST 	
STOP	S SHIFT T	S 	
STR$	ST SHIFT R	ST 	ZEICHENKETTE
SYS	S SHIFT Y	S 	
TAB(	T SHIFT A	T 	ZEICHENKETTE
TAN	keine	TAN	NUMERISCH
THEN	T SHIFT H	T 	
TIME	TI	TI	NUMERISCH
TIME$	TI$	TI$	ZEICHENKETTE
TO	keine	TO	
USR	U SHIFT S	U 	NUMERISCH
VAL	V SHIFT A	V 	NUMERISCH
VERIFY	V SHIFT E	V 	
WAIT	W SHIFT A	W 	

BESCHREIBUNG DER BASIC-SCHLÜSSELWÖRTER

ABS

TYP: Funktion-Numerisch
FORMAT: ABS(<Ausdruck>)

Funktion: Gibt den Absolutwert einer Zahl an. Dies ist ihr Wert ohne Vorzeichen. Der Absolutwert einer negativen Zahl ist diese Zahl multipliziert mit −1.

BEISPIELE DER ABS-FUNKTION:

```
10 X = ABS ( Y )
10 PRINT ABS ( X * J )
10 IF X = ABS (X) THEN PRINT "POSITIV"
```

AND

TYP: Operator
FORMAT: <Ausdruck> AND <Ausdruck>

Funktion: AND wird in Boole'schen Operationen zur Prüfung einzelner Bits und zur Wahrheitsprüfung beider Operanden benutzt.
In der Boole'schen Algebra ist das Ergebnis einer AND-Operation nur dann 1, wenn beide beteiligten Zahlen 1 sind. Das Ergebnis ist 0, wenn eine von beiden 0 ist (falsch) oder beide 0 sind.

BEISPIELE DER 1-BIT-AND-OPERATION:

```
   0        1        0        1
AND 0    AND 0    AND 1    AND 1
---      ---      ---      ---
   0        0        0        1
```

Der COMMODORE 64 führt AND-Operationen bei Zahlen im Bereich von −32768 bis +32767 durch. Brüche dürfen nicht benutzt werden, und Zahlen, die außerhalb dieses Bereichs liegen, führen zur Anzeige der Fehlermeldung **?ILLEGAL QUANTITY.**
Bei der Umwandlung in ein Binärformat ergibt sich ein zulässiger Bereich von 16 Bits für jede Zahl. Entsprechende Bits werden AND-verbunden und ergeben ein 16-Bit-Ergebnis im gleichen Bereich.

BEISPIELE DER 16-BIT-AND-OPERATION:

```
                                17
                            AND 194
                     0000000000010001
                 AND 0000000011000010
            (BINARY) 0000000000000000
        (DECIMAL)                   0


                             32007
                         AND 28761
                     0111110100000111
                 AND 0111000001011001
            (BINARY) 0111000000000001
        (DECIMAL)                28673


                              -241
                         AND 15359
                     1111111100001111
                 AND 0011101111111111
            (BINARY) 0011101100001111
        (DECIMAL)                15119
```

Bei der Richtig-/Falschbewertung einer Zahl nimmt der Computer stets an, daß die Zahl richtig ist, wenn ihr Wert nicht 0 lautet. Bei der Auswertung eines Vergleichs wird bei richtigem Ergebnis der Wert −1 und bei falschem Ergebnis der Wert 0 zugeordnet. Im Binärformat besteht −1 aus lauter Einsen und 0 aus lauter Nullen. Aus diesem Grund ist das Ergebnis bei der AND-Verknüpfung von Richtig-/Falschbewertungen stets richtig, wenn beliebige Bits im Ergebnis richtig sind.

BEISPIELE DER AND-VERKNÜPFUNG MIT RICHTIG-/FALSCHBEWERTUNGEN:

```
50 IF X=7 AND W=3 THEN GOTO 10: REM NUR WAHR, WENN SOWOHL
   X=7 und W=3 WAHR SIND
60 IF A AND Q=7 THEN GOTO 10: REM WAHR, WENN A#0 und Q=7
```

ASC

TYP: Numerisch
FORMAT: ASC (<String>)

Funktion: Durch ASC wird eine Zahl von 0 bis 255 gegeben, die dem COMMODORE ASCII-Wert des ersten Zeichens der Zeichenkette entspricht. In Anhang C wird eine Tabelle der COMMODORE ASCII-Werte gegeben.

BEISPIELE DER ASC-FUNKTION:

```
10 PRINT ASC("Z")
20 X = ASC("ZEBRA")
30 J = ASC(J$)
```

Sind in der Zeichenkette keine Zeichen enthalten, wird die Fehlermeldung **?ILLEGAL QUANTITY** angezeigt. Ist in dem oben gezeigten Beispiel J$="", ist die Anwendung der ASC-Funktion nicht erlaubt. Die Anweisungen GET und GET# lesen CHR$(0) als einen Leerstring. Um dieses Problem zu beseitigen, wird am Ende der Folge CHR$(0) wie nachstehend gezeigt, eingefügt.

BEISPIEL DER ASC-FUNKTION OHNE ANZEIGE DER FEHLERMELDUNG ?ILLEGAL QUANTITY:

```
30 J = ASC(J$ + CHR$(0))
```

ATN

TYP: Funktion-Numerisch
FORMAT: ATN (<Zahl>)

Funktion: Diese mathematische Funktion gibt den Arcustangens der Zahl wieder. Das Ergebnis ist der Winkel (in Bogenmaß), dessen Tangens die gegebene Zahl ist. Das Ergebnis liegt stets in dem Bereich von $-\pi/2$ bis $+\pi/2$.

BEISPIELE DER ATN-FUNKTION:

```
10 PRINT ATN ( 0 )
20 X = ATN ( J ) * 180 / π : REM UMRECHNUNG DES WINKELS IN GRAD
```

CHR$

TYP: Zeichenkette
FORMAT: CHR$ (<Zahl>)

Funktion: Über diese Funktion wird ein COMMODORE ASCII-Code in sein entsprechendes Zeichen umgewandelt. Eine Liste der Zeichen und ihrer entsprechenden Codes finden Sie in Anhang C. Die Zahl muß einen Wert zwischen 0 und 255 haben, da sonst die Fehlermeldung **?ILLEGAL QUANTITY** angezeigt wird.

BEISPIELE DER FUNKTION CHR$:

```
10 PRINT CHR$(65) : REM 65 = GROSSES A
20 A$ = CHR$(13) : REM 13 = RETURN TASTE
50 A = ASC(A$) : A$ = CHR$(A): REM UMWANDLUNG IN C64 ASCII CODE
   UND UMGEKEHRT
```

CLOSE

TYP: Ein-/Ausgabeanweisung
FORMAT: CLOSE <logische Filenummer>

Funktion: Über diese Anweisung kann eine beliebige Datendatei oder ein Gerätekanal geschlossen werden. Die Dateinummer ist hierbei die gleiche wie beim Öffnen der entsprechenden Datei oder des Gerätes (siehe Anweisung OPEN im Abschnitt "Eingabe-/Ausgabeprogrammierung").
Beim Arbeiten mit externen Speichern wie z. B. Kassetten und Disketten wird durch die CLOSE-Anweisung jeder Inhalt des Puffers durch das Gerät gespeichert. Wird dies nicht ausgeführt, so ist die Datei nur unvollständig auf der Kassette und unlesbar auf der Diskette. Bei anderen Geräten ist die CLOSE-Anweisung nicht unbedingt erforderlich, sie setzt jedoch Speicherkapazität für weitere Dateien frei. Bezüglich weiterer Einzelheiten schlagen Sie bitte im Handbuch des entsprechenden Peripheriegeräts nach.

BEISPIELE DER CLOSE-ANWEISUNG:

```
10 CLOSE 1
20 CLOSE X
30 CLOSE 9 * ( 1 + J )
```

CLR

TYP: Anweisung
FORMAT: CLR

Funktion: Über diese Anweisung kann RAM-Speicher verfügbar gemacht werden, der benutzt wurde, jetzt aber nicht mehr gebraucht wird. Die BASIC-Programme im Speicher bleiben unberührt, sämtliche Variablen, Felder, GOSUB-Adressen, FOR . . . NEXT-Schleifen, vom Benutzer definierte Funktionen und Dateien werden jedoch aus dem Speicher gelöscht. Der Speicherplatz steht dann für neue Variablen usw. zur Verfügung.
Dateien auf Disketten oder Kassetten werden nicht richtig durch die CLR-Anweisung geschlossen. Die Dateieninformationen, einschließlich aller nicht vollständigen Puffer, sind für den Computer verloren. Das Disketten-Laufwerk geht immer noch davon aus, daß die Datei offen ist. Wegen Einzelheiten siehe Anweisung CLOSE.

BEISPIEL DER CLR-ANWEISUNG:

```
10 X=25
20 CLR
30 PRINT X

RUN
0

READY
```

CMD

TYP: Ein-/Ausgabeanweisung
FORMAT: CMD <logische Filenummer> [, String]

Funktion: Über diese Anweisung wird die Datenausgabe vom Bildschirm auf das angegebene File umgeschaltet. Dieses File kann der Diskette, der Kassette, dem Drucker oder einer Ein-/Ausgabevorrichtung, wie z. B. einem Modem, zugeordnet sein. Die logische Filenummer muß zuvor in einer OPEN-Anweisung festgelegt werden. Der String wird, wenn er festgelegt ist, zum File geschickt.

Wenn dieser Befehl wirksam ist, werden die PRINT-Anweisungen und LIST-Befehle nicht auf dem Bildschirm angezeigt, sondern übertragen den Text im gleichen Format auf das logische File.
Damit die Ausgabe wieder auf dem Bildschirm angezeigt wird, muß der Befehl PRINT# eine Leerzeile vom CMD-Gerät vor dem Schließen (CLOSE) schicken, damit dieses nicht mehr auf die Datenübertragung wartet (dies nennt man "Un-listening" des Geräts).
Durch Systemfehler (wie z. B. **?SYNTAX ERROR**) wird die Ausgabe wieder zurück auf den Bildschirm geholt. Hierdurch erfolgt kein Un-listening der Geräte, so daß danach eine Leerzeile übertragen werden muß. (Bezüglich Einzelheiten schlagen Sie bitte in dem Handbuch des Druckers oder der Diskette nach.)

BEISPIELE DER CMD-ANWEISUNG:

```
OPEN 4, 4: CMD 4, "TITLE" : LIST: REM DRUCKT PROGRAMMLISTING AUS
PRINT# 4: CLOSE 4: REM BEENDET AUSDRUCK

10 OPEN 1, 1, 1, "TEST": REM ANLEGEN EINER SEQUENTIELLEN DATEI
20 CMD 1: REM AUSGABE AUF KASSETTE
30 FOR L = 1 TO 100
40 PRINT L: REM SCHREIBT DIE ZAHLEN IN PUFFER DES KASSETTEN-
   REKORDERS
50 NEXT
60 PRINT# 1: REM UNLISTEN
70 CLOSE 1: REM SCHREIBT PUFFERINHALT AUF DAS BAND, KORREK-
   TER ABSCHLUSS
```

CONT

TYP: Befehl
FORMAT: CONT

Funktion: Über diesen Befehl wird die Programmausführung wieder gestartet, die durch die Anweisung STOP oder END oder durch Drücken der Taste RUN/STOP abgebrochen wurde. Das Programm wird genau an der Stelle fortgesetzt, an der es unterbrochen wurde.
Bei gestoppter Programmausführung können Variablen überprüft und geändert oder das Programm durchgesehen werden. Bei der Fehlerbeseitigung oder Überprüfung eines Programms können STOP-Anweisungen so gesetzt werden, daß die Über-

prüfung von Variablen und des Programmablaufs möglich wird. Die Fehlermeldung **CAN'T CONTINUE** erscheint, wenn das Programm abgeändert wurde (selbst wenn nur die Taste RETURN gedrückt wurde und der Cursor hinter einer unveränderten Zeile stand), wenn das Programm durch einen Fehler gestoppt wurde, oder wenn Sie vor der Eingabe von CONT zum erneuten Start des Programms einen Fehler verursacht haben.

BEISPIEL DES CONT-BEFEHLS:

```
10 PI=0:C=1
20 PI=PI+4/C-4/(C+2)
30 PRINT PI
40 C=C+4:GOTO 20
```

Dieses Programm berechnet den Wert von PI. Geben Sie über die Tastatur RUN ein, und drücken Sie, wenn das Programm läuft, nach kurzer Zeit die Taste RUN/STOP . Es erscheint folgende Anzeige:

BREAK IN 20 **Anmerkung:** Die Zahl kann verschieden sein

Geben Sie den Befehl PRINT C ein, um festzustellen, wie weit der COMMODORE 64 gekommen ist. Zur Wiederaufnahme des Programms benutzen Sie die Anweisung CONT.

COS

TYP: Funktion
FORMAT: COS (<Zahl>)

Funktion: Über diese mathematische Funktion wird der Kosinus einer Zahl berechnet, wobei diese Zahl als das Bogenmaß eines Winkels aufgefaßt wird.

BEISPIELE DER COS-FUNKTION:

```
10 PRINT COS ( 0 )
20 X = COS ( Y * π / 180 ) : REM UMRECHNUNG VON GRAD IN BOGEN-
   MASS
```

DATA

TYP: Anweisung
FORMAT: DATA <Konstantenliste>

Funktion: Über die DATA-Anweisungen werden Informationen innerhalb eines Programms gespeichert. Das Programm benutzt die Informationen über die Anweisung READ, bei der die Konstanten der Reihe nach aus den DATA-Anweisungen ausgelesen werden.
Alle DATA-Anweisungen in einem Programm werden als kontinuierliche Liste behandelt. Die Daten werden von links nach rechts, von der Zeile mit der niedrigsten Zeilennummer bis zu der mit der höchsten Zeilennummer gelesen. Wenn die READ-Anweisung auf Daten trifft, die nicht dem erforderlichen Typ entsprechen (ist der Typ z. B. eine Zahl und ein String wird gefunden), dann wird eine Fehlermeldung angezeigt.
Als Daten können beliebige Zeichen gewählt werden, bei bestimmten Zeichen müssen die Daten jedoch in Anführungszeichen sein (" "). Hierzu gehören Interpunktionszeichen wie z. B. Komma (,), Doppelpunkt (:), Leerstellen, Buchstaben mit SHIFT, graphische Zeichen und Cursor-Steuerzeichen.

BEISPIELE DER DATA-ANWEISUNG:

```
10 DATA 1, 10, 5, 8
20 DATA JOHN, PAUL, GEORGE, RINGO
30 DATA "DEAR MARY, HOW ARE YOU, LOVE, BILL"
40 DATA -1.7E-9, 3.33
```

DEF FN

TYP: Anweisung
FORMAT: DEF FN <Name> (<Variable>) = <Ausdruck>

Funktion: Auf diese Weise wird vom Anwender eine Funktion definiert, die später im Programm benutzt werden kann. Diese Funktion kann aus einer beliebigen mathematischen Gleichung bestehen. Selbstdefinierte Funktionen sparen Speicherkapazität bei Programmen ein, bei denen eine lange Gleichung an mehreren Stellen auftritt.
Die Gleichung muß nur einmal in der Definitionsanweisung bestimmt werden und wird danach als abgekürzter Funktionsname aufgerufen.

Der Funktionsname setzt sich aus den Buchstaben FN gefolgt von einem beliebigen Variablennamen zusammen. Der Variablennamen kann aus ein oder zwei Zeichen bestehen, wobei der erste ein Buchstabe und der zweite ebenfalls ein Buchstabe oder eine Zahl ist.

BEISPIELE DER ANWEISUNG DEF FN:

```
10 DEF FN A (X) = X + 7
20 DEF FN AA (X) = Y * Z
30 DEF FN A9 (Q) = INT( RND( 1)* Q+ 1)
```

Diese Funktion wird später im Programm mit Hilfe des Funktionsnamens aufgerufen, wobei eine Variable in Klammern steht. Der Funktionsname wird wie jede andere Variable behandelt, und sein Wert wird automatisch berechnet.

BEISPIELE FÜR FN:

```
40 PRINT FN A (9)
50 R = FN AA (9)
60 G = G + FN A9 (10)
```

In Zeile 50 in obigem Beispiel beeinflußt die Zahl 9 in den Klammern nicht das Funktionsergebnis, da die Funktionsdefinition in Zeile 20 nicht die Variable in Klammern benutzt. Das Ergebnis ist Y mal Z, unabhängig vom Wert X. In den beiden anderen Funktionen wird das Ergebnis durch den Wert in Klammern beeinflußt.

DIM

TYP: Anweisung
FORMAT: DIM <Variable> (<Index>) [, <Variable> (<Index>) . . .]

Funktion: Über diese Anweisung wird ein Feld oder eine Matrix von Variablen bestimmt. Auf diese Weise können Sie den Variablennamen mit einem Index benutzen. Der Index weist auf das benutzte Element. Der niedrigste Index eines Feldes ist Null, und die höchste Zahl ist die Zahl, die in der DIM-Anweisung gegeben wird (max. 32767).
Die DIM-Anweisung muß einmal (*und darf nur einmal*) für jedes Feld ausgeführt werden. Bei einer erneuten Ausführung dieser Zeile wird die Fehlermeldung

REDIM'D ARRAY angezeigt. Aus diesem Grund führen die meisten Programme alle DIM-Operationen ganz am Anfang aus.
Ein Feld kann beliebige Dimensionen und 255 verschiedene Indizes enthalten. Dies ist lediglich durch die Kapazität des RAM-Speichers begrenzt, die für die Variablen zur Verfügung steht. Das Feld kann, wie oben gezeigt, aus normalen numerischen Variablen, aus Zeichenketten oder ganzen Zahlen bestehen. Handelt es sich bei den Variablen nicht um normale Zahlen, so benutzen Sie nach dem Variablennamen das Zeichen $ oder %, um eine String- oder Ganzzahlvariable anzugeben.
Wurde für ein Feld, auf das im Programm Bezug genommen wurde, keine DIM-Anweisung gegeben, so werden automatisch 11 für jede Dimension reserviert.

BEISPIELE DER DIM-ANWEISUNG:

```
10 DIM A ( 100 )
20 DIM Z ( 5, 7 ), Y ( 3, 4, 5)
30 DIM Y7% ( Q )
40 DIM PH$ (1000)
50 F (4) =9: REM ES WIRD AUTOMATISCH DIM F (10) FESTGESETZT
```

BERECHNUNG DES DURCH DIM BENUTZTEN SPEICHERS:

5 Bytes für den Feldnamen
2 Bytes für jede Dimension
2 Bytes/Element für ganzzahlige Variablen
5 Bytes/Element für normale numerische Variablen
3 Bytes/Element für Stringvariablen
1 Byte für jedes Zeichen in jedem String

END

TYP: Anweisung
FORMAT: END

Funktion: Hierdurch wird die Programmausführung beendet und die Meldung READY angezeigt. Die Steuerung wird nun wieder an den Benutzer übergeben.
Ein Programm kann beliebig viele END-Anweisungen enthalten. Auch wenn es nicht erforderlich ist, überhaupt eine END-Anweisung in das Programm einzugeben, wird doch eine solche Anweisung empfohlen. Die Anweisung END entspricht der STOP-Anweisung. Der Unterschied besteht lediglich darin, daß durch STOP die

Meldung **BREAK IN LINE XX** und durch END nur READY angezeigt wird. Bei beiden Anweisungen ist eine Wiederaufnahme der Ausführung durch Eingabe des Befehls CONT möglich.

BEISPIELE DER END-ANWEISUNG:

```
10 PRINT "WILLST DU DAS PROGRAMM WIRKLICH STARTEN"
20 INPUT A$
30 IF A$ = "NEIN" THEN END
30 REM REST DES PROGRAMMS
999 END
```

EXP

TYP: Funktion-Numerisch
FORMAT: EXP (<Zahl>)

Funktion: Mit dieser mathematischen Funktion wird die Konstante e (2.71828183) in die Potenz der angegebenen Zahl erhoben. Durch einen Wert, der größer ist als 88.0296919 kommt es zu der Fehlermeldung **?OVERFLOW.**

BEISPIELE DER EXP-FUNKTION:

```
10 PRINT EXP (1)
20 X = Y * EXP (Z * Q)
```

FN

TYP: Funktion-Numerisch
FORMAT: FN <Name> (<Zahl>)

Funktion: Diese Funktion verweist auf die zuvor mit DEF-Anweisung definierte Funktion. Die Zahl wird eingesetzt und dann der Funktionswert berechnet. Das Ergebnis ist ein numerischer Wert.
Diese Funktion kann in der direkten Betriebsart benutzt werden, wenn die Anweisung DEF ausgeführt wurde.
Wird FN vor der entsprechenden DEF-Anweisung ausgeführt, so wird der Fehler UNDEF'D FUNCTION angezeigt.

BEISPIELE DER FN-FUNKTION:

```
PRINT FN A ( Q )
1100 J = FN J (7) + FN J (9)
9990 IF FN B7 (I+1)= 6 THEN END
```

FOR . . . TO . . . [STEP . . .]

TYP: Anweisung
FORMAT: FOR <Variable> = <Start> TO <Grenze>[STEP <Schrittweite>]

Funktion: Dies ist eine BASIC-Anweisung, mit der Sie eine Variable als Zähler benutzen können. Sie müssen bestimmte Parameter angeben: Gleitpunkt-Variablen, Startwert dieser Variablen, Endwert und Schrittweite. Diese Schrittweite kann irgendeine Gleitpunktzahl sein.
Nachstehend sehen Sie ein einfaches BASIC-Programm, das von 1 bis 10 zählt, jede Zahl anzeigt und mit keinen FOR-Anweisungen arbeitet:

```
100 L = 1
110 PRINT L
120 L = L + 1
130 IF L <= 10 THEN 110
140 END
```

Nachfolgend steht das gleiche Programm, diesmal jedoch mit der FOR-Anweisung:

```
100 FOR L = 1 TO 10
110 PRINT L
120 NEXT L
130 END
```

Wie Sie feststellen können, ist das Programm kürzer und leichter verständlich, wenn die FOR-Anweisung benutzt wird.
Beim Ausführen der FOR-Anweisung finden verschiedene Operationen statt. Der <Start-Wert> wird in die vom Zähler benutzte <Variable> eingesetzt. In obigem Beispiel wird in L eine 1 eingesetzt.

Bei Erreichen der Anweisung NEXT wird die Schrittweite zu der <Variablen> addiert. War STEP nicht enthalten, wird die <Schrittweite> auf +1 gesetzt. Kommt das obige Programm das erste Mal zu Zeile 120, wird 1 zu L addiert, so daß sich für L ein neuer Wert von 2 ergibt.
Nun wird der Wert in der <Variablen> mit der <Grenze> verglichen. Ist die <Grenze> noch nicht erreicht, geht das Programm (weiter) zur Zeile hinter der FOR-Anweisung. In diesem Fall ist der Wert 2 von L kleiner als die Grenze 10, so daß das Programm in die Zeile 110 zurück springt.
Wird die <Grenze> durch die <Variable> überschritten, so ist die Schleife beendet, und das Programm setzt mit der Zeile nach der NEXT-Anweisung fort. Ist in unserem Beispiel der Wert L=11 erreicht, also die Grenze 10 überschritten, wird das Programm in Zeile 130 fortgesetzt.
Ist die <Schrittweite> positiv, muß die <Variable> die <Grenze> überschreiten. Ist der Wert negativ, muß sie entsprechend kleiner als die <Grenze> sein.

Anmerkung: Eine Schleife wird stets mindestens einmal ausgeführt.

BEISPIELE FÜR DIE ANWEISUNG FOR . . . TO . . . STEP . . .:

```
100 FOR L = 100 TO 0 STEP -1
100 FOR L = PI TO 6* π STEP .01
100 FOR AA = 3 TO 3
```

FRE

TYP: Funktion
FORMAT: FRE (<Variable>)

Funktion: Über diese Funktion erfahren Sie, wieviel RAM-Kapazität für Ihr Programm und die Variablen zur Verfügung steht. Wird durch ein Programm mehr Speicherkapazität benötigt als vorhanden ist, erscheint die Fehlermeldung **OUT OF MEMORY.**
Die Zahl in Klammern kann einen beliebigen Wert haben und wird in der Berechnung nicht benutzt.

Anmerkung: Ist das Ergebnis von FRE negativ, so addieren Sie 65536 zu der FRE-Zahl, um die Anzahl der verfügbaren Bytes zu erhalten.

BEISPIELE DER FRE-FUNKTION:

```
PRINT FRE ( 0 )
10 X = ( FRE ( K ) – 1000 ) / 7
950 IF FRE ( 0 ) < 100 THEN PRINT "NOT ENOUGH ROOM"
```

Anmerkung: Auf folgende Weise wird Ihnen stets die derzeitig verfügbare RAM-Kapazität angezeigt:
PRINT FRE(0) – (FRE(0)<0)* 65536

GET

TYP: Anweisung
FORMAT: GET <Variablenliste>

Funktion: Über diese Anweisung wird jede vom Benutzer gedrückte Taste gelesen. Bei der Eingabe über die Tastatur werden die Zeichen im Tastaturpuffer des COMMODORE 64 gespeichert. Der Puffer hat eine Kapazität von 10 Zeichen, so daß der Anschlag der elften und aller weiteren Tasten nicht berücksichtigt wird. Durch Lesen eines der Zeichen mit der GET-Anweisung wird Platz für weitere Zeichen geschaffen.
Wenn in der GET-Anweisung numerische Daten spezifiziert werden und der Benutzer eine andere Taste als eine Zahlentaste anschlägt, wird die Fehlermeldung **?SYNTAX ERROR** angezeigt. Aus Sicherheitsgründen sollten die Tastenanschläge als Zeichenketten gelesen und später in Zahlen umgewandelt werden.
Mit der GET-Anweisung können einige Beschränkungen der INPUT-Anweisung vermieden werden. Bezüglich Einzelheiten schlagen Sie bitte im Abschnitt über die Verwendung der GET-Anweisung in dem Kapitel "Programmiertechniken" nach.

BEISPIELE DER GET-ANWEISUNG:

```
10 GET A$: IF A$ = "" THEN 10: REM WARTESCHLEIFE BIS TASTEN-
   DRUCK
20 GET A$, B$, C$, D$, E$: REM LIEST 5 ZEICHEN
30 GET A, A$
```

GET#

TYP: Ein-/Ausgabeanweisung
FORMAT: GET# <logische Filenummer>, <Variablenliste>

Funktion: Mit dieser Anweisung werden die Zeichen einzeln von der angegebenen Datei oder dem angegebenen Gerät gelesen. Sie entspricht der GET-Anweisung. Der Unterschied besteht darin, daß die Daten nicht von der Tastatur kommen. Wird kein Zeichen empfangen, so wird die Variable einem Leerstring zugeordnet (entspricht "") oder einer 0 bei numerischen Variablen. Zeichen, die Daten in Dateien trennen sollen, wie z. B. ein Komma (,) oder der RETURN -Tastencode (ASC-Code 13), werden wie andere Zeichen eingegeben.
Beim Arbeiten mit Gerät #3 (TV-Bildschirm) werden über diese Anweisung die Zeichen einzeln vom Bildschirm gelesen. Bei jeder Verwendung von GET# bewegt sich der Cursor um eine Position nach rechts. Das Zeichen am Ende einer logischen Zeile wird in CHR$ (13), d. h. in den RETURN -Tastencode umgewandelt.

BEISPIELE DER ANWEISUNG GET#:

```
5 GET# 1, A$
10 OPEN 1, 3: GET# 1, Z7$
20 GET# 1, A, B, C$, D$
```

GOSUB

TYP: Anweisung
FORMAT: GOSUB <Zeilennummer>

Funktion: Dies ist eine spezielle Form der GOTO-Anweisung. Es besteht jedoch ein wesentlicher Unterschied. GOSUB speichert, von wo gesprungen wird. Wird die RETURN-Anweisung (unterscheidet sich von dem Anschlag der Taste RETURN) im Programm erreicht, springt das Programm zurück zur Anweisung, die unmittelbar hinter der GOSUB-Anweisung steht.
Ein Unterprogramm (GOSUB gleich GO to a SUB-routine = Sprung zum Unterprogramm) wird hauptsächlich dann eingesetzt, wenn ein kleiner Programmteil von verschiedenen Teilen des Programms benutzt wird. Durch die Verwendung von

Unterprogrammen läßt sich vermeiden, daß die gleichen Zeilen ständig an verschiedenen Programmstellen wiederholt werden müssen. Auf diese Weise entspricht GOSUB auch DEF FN. Mit DEF FN können Sie Platz bei der Verwendung von Gleichungen und mit GOSUB Platz durch die Vermeidung von sich ständig wiederholenden gleichen Programmteilen gewinnen. Nachstehendes Programm arbeitet nicht mit GOSUB.

```
100 PRINT "DIESES PROGRAMM SCHREIBT"
110 FOR L = 1 TO 500 : NEXT
120 PRINT "LANGSAM"
130 FOR L = 1 TO 500 : NEXT
140 PRINT "ES BENUTZT EINE SCHLEIFE"
150 FOR L = 1 TO 500 : NEXT
160 PRINT "ALS VERZÖGERUNG."
170 FOR L = 1 TO 500 : NEXT
```

Hier ist das gleiche Programm noch einmal, diesmal jedoch mit der GOSUB-Anweisung:

```
100 PRINT "DIESES PROGRAMM SCHREIBT"
110 GOSUB 200
120 PRINT "LANGSAM"
130 GOSUB 200
140 PRINT "ES BENUTZT EINE SCHLEIFE"
150 GOSUB 200
160 PRINT "ALS VERZÖGERUNG."
170 GOSUB 200
180 END
200 FOR L = 1 TO 500 : NEXT
210 RETURN
```

Jedesmal, wenn das Programm eine GOSUB-Anweisung ausführt, werden Zeilennummer und Position in der Programmzeile in einem speziellen Bereich, dem Stack (= Stapel) gespeichert. Dieser Stack benutzt 256 Bytes Speicherplatz. Deshalb ist die Datenmenge begrenzt, die im Stapel gespeichert werden kann. Aus diesem Grund ist die Anzahl der speicherbaren Unterprogramm-Rückkehradressen auch begrenzt. Deshalb ist besonders darauf zu achten, daß jede GOSUB-Anweisung auf das entsprechende RETURN trifft, da sonst Speicherprobleme auftreten, obwohl noch viele Bytes frei sind.

GOTO

TYP: Anweisung
FORMAT: GOTO <Zeilennummer>
oder GO TO <Zeilennummer>

Funktion: Über diese Anweisung kann das BASIC-Programm Zeilen außerhalb der numerischen Reihenfolge ausführen. Das Wort GOTO gefolgt von einer Zahl läßt das Programm an die durch diese Zahl bestimmte Zeile springen.
GOTO ohne folgende Zahl entspricht GOTO 0. Die Zeilennummer muß nach dem Wort GOTO folgen.
Mit der GOTO-Anweisung können unendliche Schleifen erstellt werden. Das einfachste Beispiel ist eine Zeile, die eine GOTO-Anweisung für sich selbst enthält, wie z. B. 10 GOTO 10. Solche Schleifen können über die Taste RUN/STOP gestoppt werden.

BEISPIELE DER GOTO-ANWEISUNG:

```
GOTO 100
10 GO TO 50
20 GOTO 999
```

IF . . . THEN . . .

TYP: Anweisung
FORMAT: IF <Ausdruck> THEN <Zeilennummer>
IF <Ausdruck> GOTO <Zeilennummer>
IF <Ausdruck> THEN <Anweisungen>

Funktion: Dies ist eine Anweisung, die hauptsächlich die "Intelligenz" von BASIC ausmacht. Mit ihr können Bedingungen ausgewertet und entsprechend dem jeweiligen Ergebnis verschiedene Maßnahmen ergriffen werden.
Dem Wort IF folgt ein Ausdruck, der Variablen, Zeichenketten, Zahlen, Vergleiche und logische Operatoren enthalten kann. Das Wort THEN erscheint auf der gleichen Zeile und wird entweder von einer Zeilennummer oder einer weiteren BASIC-Anweisung gefolgt. Ist der Ausdruck falsch, wird alles nach dem Wort THEN auf dieser Zeile überlesen, und die Ausführung wird in der nächsten Programmzeile fortgesetzt. Bei einem richtigen Ergebnis erfolgt entweder eine Programmverzwei-

gung zur Zeilennummer nach dem Wort THEN oder die Ausführung der nachfolgenden BASIC-Anweisungen in dieser Zeile.

BEISPIEL DER ANWEISUNG IF . . . GOTO . . .:

```
100 INPUT "GIB EINE ZAHL EIN"; N
110 IF N <= 0 GOTO 200
120 PRINT "QUADRATWURZEL=" SQR(N)
130 GOTO 100
200 PRINT "ZAHL MUSS SEIN >0"
210 GOTO 100
```

Bei diesem Programm wird die Quadratwurzel einer beliebigen positiven Zahl angezeigt. Die IF-Anweisung wird hier benutzt, um die Eingabe zu überprüfen. Ist N <= 0 richtig, springt das Programm zu Zeile 200. Ist N 0, so wird als nächstes Zeile 120 ausgeführt. Bitte beachten Sie, daß THEN GOTO nicht bei IF . . . THEN benötigt wird. So bedeutet in Zeile 110 z. B. GOTO 200 tatsächlich THEN GOTO 200.

BEISPIEL DER ANWEISUNG IF . . . THEN . . .:

```
100 FOR L = 1 TO 100
110 IF RND(1) <.5 THEN X = X+ 1 : GOTO 130
120 Y = Y+ 1
130 NEXT L
140 PRINT "KOPF= " X
150 PRINT "ZAHL= " Y
```

Das IF in Zeile 110 überprüft eine beliebige Zahl, um festzustellen, ob sie kleiner als .5 ist. Ist das Ergebnis richtig, werden alle Anweisungen nach dem Wort THEN ausgeführt: Zunächst wird 1 zu X addiert, und danach springt das Programm in die Zeile 130. Ist das Ergebnis falsch, so geht das Programm zur nächsten Anweisung in Zeile 120 weiter.

INPUT

TYP: Anweisung
FORMAT: INPUT ["<Kommentar>" ;] <Variablenliste>

Funktion: Über diese Anweisung können vom Bediener Informationen in den Computer eingegeben werden. Bei der Ausführung erscheint auf dem Bildschirm ein Fragezeichen (?), und der Cursor erscheint eine Stelle rechts neben dem Fragezeichen. Der Computer wartet nun mit blinkendem Cursor darauf, daß der Bediener die Antwort über die Tastatur eingibt und danach die Taste RETURN drückt. Nach dem Wort INPUT kann ein beliebiger Text in Anführungszeichen (" ") folgen. Dieser Text erscheint auf dem Bildschirm, und danach folgt das Fragezeichen. Nach dem Text folgt ein Semikolon (;) und der Name einer oder mehrerer durch Kommata getrennter Variablen. Diese Variablen geben an, wo der Computer die vom Bediener eingegebene Information speichert. Hierbei kann es sich um beliebige Variablennamen handeln, und für verschiedene Eingaben müssen unterschiedliche Variablennamen gewählt werden.

BEISPIELE DER INPUT-ANWEISUNG:

```
100 INPUT A
110 INPUT B, C, D
120 INPUT "KOMMENTAR"; E
```

Bei der Programmausführung erscheint das Fragezeichen und zeigt dem Bediener so an, daß der COMMODORE 64 für Zeile 100 eine Eingabe erwartet. Jede beliebige eingegebene Zahl wird in A eingesetzt und später im Programm benutzt. Wurde als Antwort keine Zahl eingegeben, erscheint die Fehlermeldung **?REDO FROM START,** die bedeutet, daß zwar eine Zahl erwartet, jedoch ein String empfangen wurde. Wenn der Bediener lediglich die Taste RETURN drückt, bleibt der Variablenwert unverändert.

Nun erscheint das Fragezeichen für Zeile 110. Wenn wir nur eine Zahl eingeben und die Taste RETURN drücken, zeigt der COMMODORE 64 zwei Fragezeichen (??) an, was bedeutet, daß weitere Eingaben erforderlich sind.

Geben Sie also stets so viele Eingaben wie erforderlich, getrennt durch Kommata, ein. Werden zu viele Daten eingegeben, erscheint die Fehlermeldung **?EXTRA IGNORED,** was bedeutet, daß die überflüssigen Werte nicht in Variablen eingegeben wurden.

In Zeile 120 wird das Wort KOMMENTAR vor dem Erscheinen des Fragezeichens angezeigt. Zwischen dem KOMMENTAR und einer beliebigen Variablenliste muß ein Semikolon stehen.

Die INPUT-Anweisung kann nie außerhalb eines Programms benutzt werden.

INPUT#

TYP: Ein-/Ausgabeanweisung
FORMAT: INPUT# <logische Filenummer> , <Variablenliste>

Funktion: Dies ist für gewöhnlich der schnellste und einfachste Weg, um die in einer Disketten- oder Kassetten-Datei gespeicherten Daten zu lesen. Die Daten haben die Form von Variablen mit einer Länge von max. 80 Zeichen (es wird also nicht wie bei der Anweisung GET# jeweils nur ein Zeichen gelesen). Zunächst muß die Datei geöffnet werden. Dann kann INPUT# in die Variablen einlesen.
Der Befehl INPUT# geht davon aus, daß eine Variable beendet ist, wenn ein RETURN-Code (CHR$ (13)), ein Komma (,), Semikolon (;) oder ein Doppelpunkt (:) gelesen wird. Diese Zeichen können gegebenenfalls in Anführungszeichen eingeschlossen werden (siehe Anweisung PRINT#).
Handelt es sich beim Variablentyp um ein numerisches Zeichen und wird ein nichtnumerisches Zeichen empfangen, so wird der Fehler **BAD DATA** angezeigt. INPUT# kann Zeichenketten von max. 80 Zeichen lesen. Wird diese Zeichenanzahl überschritten, dann wird die Fehlermeldung **STRING TOO LONG** angezeigt.
Wird Gerät #3 benutzt (Bildschirm), so liest diese Anweisung eine ganze logische Zeile und bewegt den Cursor dann zur nächsten Zeile.

BEISPIELE DER ANWEISUNG INPUT#:

```
10 INPUT# 1, A
20 INPUT# 2, A$, B$
```

INT

TYP: Ganzzahl-Funktion
FORMAT: INT (<numerisch>)

Funktion: Gibt den ganzzahligen Wert eines Ausdrucks wieder. Ist der Ausdruck positiv, wird der Bruch weggelassen. Ist der Ausdruck negativ, so wird bei einem Bruch die nächstniedrigere ganze Zahl wiedergegeben.

BEISPIELE DER INT-FUNKTION:

```
120 PRINT INT(99.4343), INT(-12.34)

 99          -13
```

LEFT$

TYP: Stringfunktion
FORMAT: LEFT$ (<Zeichenkette>, <Ganze Zahl>)

Funktion: Gibt eine Zeichenkette wieder, die die Zeichen von der äußersten linken Position bis zur angegebenen Zahl umfaßt. Der ganzzahlige Argumentenwert muß im Bereich von 0 bis 255 liegen. Ist die ganze Zahl größer als die Zeichenkettenlänge, so wird der gesamte String wiedergegeben. Hat diese Zahl den Wert Null, so wird ein Leerstring (Länge Null) wiedergegeben.

BEISPIELE DER FUNKTION LEFT$:

```
10 A$ = "COMMODORE COMPUTER"
20 B$ = LEFT$(A$,9): PRINT B$
RUN

COMMODORE
```

LEN

TYP: Ganzzahl-Funktion
FORMAT: LEN (<Zeichenkette>)

Funktion: Gibt die Anzahl der Zeichen in einem String wieder. Nicht angezeigte Zeichen und Leerzeichen werden mitgezählt.

BEISPIELE DER LEN-FUNKTION:

```
CC$ = "COMMODORE COMPUTER": PRINT LEN(CC$)

18
```

LET

TYP: Anweisung
FORMAT: [LET] <Variable> = <Ausdruck>

Funktion: Die LET-Anweisung wird benutzt, um einer Variablen einen Wert zuzuordnen. Das Wort LET ist jedoch optional, und erfahrene Programmierer lassen es daher meist aus, damit es keine Speicherkapazität vergeudet. Das Gleichheitszeichen (=) allein genügt bei der Wertzuordnung eines Ausdrucks zu einem Variablennamen.

BEISPIELE DER LET-ANWEISUNG:

```
10 LET D= 12              (Dies entspricht D = 12)
20 LET E$ = "ABC"
30 F$ = "DEF"
40 SUM$ = E$ + F$         (SUM$ entspricht ABCDEF)
```

LIST

TYP: Befehl
FORMAT: LIST [[<Erste Zeile>]–[<Letzte Zeile>]]

Funktion: Mit dem LIST-Befehl können Sie sich Zeilen im BASIC-Programm anschauen, die derzeitig im Speicher Ihres COMMODORE 64 gespeichert sind. Auf diese Weise können Sie den Bildschirm-Editor zum Editieren von aufgelisteten Programmen schnell und einfach einsetzen.

Mit dem LIST-Systembefehl wird ganz oder teilweise das Programm angezeigt, das derzeitig im Speicher abgelegt ist. LIST wird normalerweise zum Bildschirm geleitet; die CMD-Anweisung kann benutzt werden, um die Ausgabe auf ein externes Gerät, wie z. B. Drucker oder Diskette, umzuschalten. Der LIST-Befehl kann im Programm erscheinen, nach der Ausführung von LIST wird jedoch stets die Meldung READY angezeigt.

Wenn die Programmliste auf dem Bildschirm erscheint, kann das "Rollen" des Bildschirms von unten nach oben durch Drücken der Taste CTRL verlangsamt werden. LIST wird durch Drücken der Taste RUN/STOP abgebrochen.

Werden keine Zeilennummern angegeben, so wird das ganze Programm aufgelistet. Ist nur die erste Zeilennummer angegeben und folgte danach ein Gedankenstrich (–), so werden diese Zeile und alle Zeilen mit größeren Nummern aufgelistet. Ist nur die letzte Zeilennummer angegeben und steht davor ein Gedankenstrich, so werden alle Zeilen vom Programmanfang bis zu dieser Zeile aufgelistet. Sind beide Zahlen angegeben, so wird der gesamte Bereich einschließlich dieser Zahlen angezeigt.

BEISPIELE DES LIST-BEFEHLS:

LIST	(Listet das derzeitig im Speicher befindliche Programm auf.)
LIST 500	(Listet nur Zeile 500 auf.)
LIST 150–	(Listet alle Zeilen von 150 bis zum Ende auf.)
LIST –1000	(Listet alle Zeilen von der niedrigsten bis 1000 auf.)
LIST 150–1000	(Listet die Zeilen 150 bis einschl. 1000 auf.)

```
10 PRINT "THIS IS LINE 10"
20 LIST                    (LIST im Programmier-Modus)
30 PRINT "THIS IS LINE 30"
```

LOAD

TYP: Befehl
FORMAT: LOAD ["<Programmname>"] [,<Gerätenummer>] [,<Sekundäradresse>]

Funktion: Über die LOAD-Anweisung wird der Inhalt einer Programmdatei von Kassette oder Diskette in den Speicher gelesen. Auf diese Weise können Sie die geladenen Informationen benutzen oder sie ändern. Die Gerätenummer ist optional, der Computer wählt jedoch standardmäßig 1 (Kassetteneinheit), wenn keine besondere Eingabe erfolgt. Die Diskettenstation hat normalerweise die Gerätenummer 8. Über den Befehl LOAD werden alle offenen Dateien geschlossen, und im Direktmodus wird vor dem Lesen des Programms ein CLR durchgeführt. Wird LOAD inmitten

eines Programms ausgeführt, so wird das geladene Programm automatisch gestartet, d. h., Sie können LOAD benutzen, um mehrere Programme zu verketten; dabei werden keine Variablen gelöscht.
Beim Arbeiten mit übereinstimmenden Dateinamemustern wird die erste Datei, die mit dem Muster übereinstimmt, geladen. Durch den Asterisk ("*") wird der erste Dateiname im Disketteninhaltsverzeichnis geladen. Existiert der benutzte Dateiname nicht oder handelt es sich nicht um eine Programmdatei, so wird die BASIC-Fehlermeldung **?FILE NOT FOUND** angezeigt.
Beim Programmladen von Kassette kann <Programmname> ausgelassen werden. In diesem Fall wird die nächste Programmdatei auf der Kassette gelesen. Der COMMODORE 64 löscht den Bildschirm nach Drücken der Taste PLAY. Wird das Programm gefunden, so wird die Meldung FOUND angezeigt. Nach Drücken der Taste C= oder nach ca. 15 s wird das Programm geladen. Wird die Leertaste gedrückt, so wird das derzeitig gesuchte Programm übersprungen und versucht, das nächste zu laden. Programme werden beim Laden ab Speicherplatz 2048, wenn keine <Sekundäradresse> 1 benutzt wird, abgelegt. Wird mit der Sekundäradresse 1 gearbeitet, so wird das Programm in den Speicherplatz geladen, aus dem es zuvor abgespeichert wurde.

BEISPIELE DES LOAD-BEFEHLS:

LOAD	(Liest das nächste Programm von der Kassette)
LOAD A$	(Benutzt für die Suche den Namen in A$)
LOAD "*",8	(Lädt das erste Programm von Diskette)
LOAD "",1,1	(Sucht das erste Programm auf der Kassette und lädt es zurück in den gleichen Speicherbereich, aus dem es abgespeichert wurde)
LOAD "STAR TREK" PRESS PLAY ON TAPE FOUND STAR TREK LOADING READY.	(Lädt ein Programm von Kassette)

```
LOAD "FUN",8
SEARCHING FOR FUN
LOADING
READY.
```

(Lädt ein Programm von Diskette)

```
LOAD "GAME ONE",8,1
SEARCHING FOR GAME ONE
LOADING
READY.
```

(Lädt ein Programm in den bestimmten Speicherplatz, von dem aus das Programm auf Diskette gespeichert worden ist)

LOG

TYP: Gleitpunktfunktion
FORMAT: LOG (<numerisch>)

Funktion: Gibt den natürlichen Logarithmus (Logarithmus der Basis e) des Arguments wieder. Ist der Wert des Arguments Null oder negativ, wird die BASIC-Fehlermeldung **?ILLEGAL QUANTITY** angezeigt.

BEISPIELE DER LOG-FUNKTION:

```
25 PRINT LOG(45/7)
 1.86075234

10 NUM = LOG(ARG) / LOG(10)
```

(Berechnet den Logarithmus von ARG mit der Basis 10)

MID$

TYP: Folgefunktion
FORMAT: MID$ (<String>, <numerische Zahl A> [,<numerische Zahl B>])

Funktion: Die Funktion MID$ definiert einen Teilstring, der Teil eines größeren Strings ist. Der Startpunkt des Teilstrings wird durch das Argument <numerische Zahl A> und die Länge durch das Argument <numerische Zahl B> bestimmt. Beide numerischen Argumente können einen Wert von 0 bis 255 haben.

Ist <numerische Zahl A> größer als die Länge des <Strings>, oder ist <numerische Zahl B> Null, dann gibt MID$ einen Leerstring wieder. Wird das Argument <numerische Zahl B> ausgelassen, nimmt der Computer an, daß die Länge des restlichen Strings benutzt werden soll. Hat der Quellen-String weniger Zeichen als <numerische Zahl B> vom Startpunkt bis zum Ende, dann wird der ganze Rest dieses Strings benutzt.

BEISPIEL DER FUNKTION MID$:

```
10 A$="GOOD"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$ + MID$(B$, 8, 8)

GOOD EVENING
```

NEW

TYP: Befehl
FORMAT: NEW

Funktion: Der Befehl NEW wird benutzt, um ein derzeitig im Speicher befindliches Programm und sämtliche Variablen zu löschen. Vor der Eingabe eines neuen Programms muß NEW in der Direktbetriebsart für die Speicherlöschung benutzt werden. NEW kann auch in einem Programm eingesetzt werden. Sie sollten jedoch daran denken, daß alles, was zuvor ausgeführt wurde und noch immer im Computerspeicher ist, gelöscht wird. Dies kann sich als besonders störend bei der Programm-Fehlersuche erweisen.

> **Bitte beachten:** Wird ein altes Programm nicht vor dem Schreiben eines neuen Programms gelöscht, so kann es zu einer Vermischung kommen.

BEISPIELE DES NEW-BEFEHLS:

NEW	(Löscht das Programm und alle Variablen)
10 NEW	(Führt eine NEW-Operation durch und stoppt das Programm)

NEXT

TYP: Anweisung
FORMAT: NEXT [<Zähler>] [,<Zähler>] . . .

Funktion: Die NEXT-Anweisung wird mit FOR benutzt, um das Ende der Schleife FOR . . . NEXT zu bestimmen. Der <Zähler> ist der Variablenname vom Schleifenindex, der mit FOR zum Beginn der Schleife benutzt wird. Durch eine einzelne NEXT-Anweisung können mehrere verschachtelte Schleifen abgeschlossen werden, wenn danach die Variablennamen für jeden FOR-<Zähler> folgen. Hierzu muß jeder Name aufgeführt werden, wobei der der innersten Schleife zuerst und der der äußersten zuletzt folgt. Wird eine einzelne NEXT-Anweisung in dieser Weise benutzt, so müssen die Variablennamen durch Kommata getrennt sein. Schleifen können auf max. neun Ebenen verschachtelt werden. Werden die Zählervariablen ausgelassen, erfolgt eine Inkrementierung des Zählers, der durch die Anweisung FOR mit der derzeitigen Ebene (der verschachtelten Schleifen) verbunden ist.
Bei Erreichen der NEXT-Anweisung wird zum Zählerwert 1 oder ein optionaler STEP-Wert addiert. Er wird dann mit einem End-Wert verglichen, um festzustellen, ob die Schleife beendet werden soll. Eine Schleife wird beendet, wenn eine NEXT-Anweisung gefunden wird, deren Zählerwert größer als der End-Wert ist.

BEISPIELE DER NEXT-ANWEISUNG:

```
10 FORJ=1 TO 5: FOR K = 10 TO 20: FOR N = 5 TO -5 STEP -1

20 NEXT N, K, J          (Verschachtelte Schleifen)

10 FOR L = 1 TO 100
20 FOR M = 1 TO 10
30 NEXT M
400 NEXT L               (Beachten Sie, daß die Schleifen einander
                          nicht überschneiden)

10 FOR A = 1 TO 10
20 FOR B = 1 TO 20
30 NEXT
40 NEXT                  (Beachten Sie, daß keine Variablennamen
                          nötig sind)
```

NOT

TYP: Logischer Operator
FORMAT: NOT <Ausdruck>

Funktion: Der logische Operator NOT "komplementiert" den Wert jedes Bits in seinem einzelnen Operanden. Das Ergebnis ist ein ganzzahliges "Zweier-Komplement". Beim Arbeiten mit Gleitpunktzahlen werden die Operanden in ganze Zahlen umgewandelt und Brüche eliminiert. Der Operator NOT kann auch bei einem Vergleich zur Umkehrung des Richtig-/Falschwertes benutzt werden, der das Ergebnis einer Vergleichsprüfung ist. Aus diesem Grund kehrt er die Bedeutung eines Vergleichs um. Im nachstehenden ersten Beispiel ist der Ausdruck richtig, wenn das "Zweier-Komplement" von "AA" gleich "BB" und wenn "BB" nicht "CC" ist.

BEISPIELE DES NOT-OPERATORS:

```
10 IF NOT AA = BB AND NOT(BB = CC) THEN . . .
NN% = NOT 96: PRINT NN%
-97
```

Anmerkung: Um den Wert von NOT zu finden, benutzen Sie den Ausdruck X=(−(X+1)). (Das Zweier-Komplement einer ganzen Zahl ist das Bit-Komplement + 1.)

ON

TYP: Anweisung
FORMAT: ON <Variable> GOTO / GOSUB <Zeilennummer> [,<Zeilennummer>] . . .

Funktion: Die ON-Anweisung wird benutzt, um je nach Variablenwert zu einer von mehreren angegebenen Zeilennummern überzugehen. Der Wert der Variablen liegt im Bereich von Null bis zu der angegebenen Zeilenzahl. Ist der Wert keine ganze Zahl, so werden die Nachkommastellen weggelassen. Ist der Variablenwert z. B. 3, so geht das Programm durch die ON-Anweisung zu der dritten Zeilennummer in der Liste über. Ist der Wert einer Variablen negativ, wird die BASIC-Fehlermeldung **?ILLEGAL QUANTITY** angezeigt. Ist die Zahl Null oder größer als die Punktezahl in der Liste, wo wird die Anweisung vom Programm einfach "überlesen", und das Programm setzt mit der Anweisung nach der ON-Anweisung fort.

ON ist also eine Variante der Anweisung IF . . . THEN . . . Statt mehrere IF-Anweisungen zu benutzen, die das Programm jeweils an eine bestimmte Zeile schicken, kann eine ON-Anweisung eine Liste von IF-Anweisungen ersetzen. Im nachstehenden ersten Beispiel ersetzt die erste ON-Anweisung vier Anweisungen IF . . . THEN . . .

BEISPIELE DER ON-ANWEISUNG:

```
ON -(A=7)-2*(A=3)- 3*(A<3)-4*(A>7)GOTO 400,900,1000,100

ON X GOTO 100,130,180,220

ON X+3 GOSUB 9000,20,9000

100 ON NUM GOTO 150, 300, 320, 390

500 ON SUM / 2 + 1 GOSUB 50, 80, 20
```

OPEN

TYP: Ein-/Ausgabe-Anweisung
FORMAT: OPEN <logische Filenummer>, [<Gerätenummer>] [,<Sekundäradresse>] [,"<Dateiname> [,<Type>] [,<Modus>]"]

Funktion: Über diese Anweisung wird ein Kanal für die Ein- und Ausgabe zu einem Peripheriegerät geöffnet. Sie brauchen jedoch wahrscheinlich nicht alle Teile für jede OPEN-Anweisung. Einige OPEN-Anweisungen benötigen lediglich zwei Codes:

1) LOGISCHE FILENUMMER
2) GERÄTENUMMER

Die <logische Filenummer> ist die logische Nummer, die die Anweisungen OPEN, CLOSE, CMD, GET#, INPUT# und PRINT#, den Dateinamen und das zu verwendende Gerät miteinander in Beziehung setzt. Sie kann im Bereich von 1 bis 255 liegen.

Anmerkung: Filenummern über 128 haben spezielle Auswirkungen, so daß Sie lediglich Zahlen bis 127 verwenden sollten.

Jedes Peripheriegerät (Drucker, Diskettenstation, Kassetteneinheit) im System hat seine eigene Nummer, auf die es antwortet. Die <Gerätenummer> wird zusammen mit OPEN benutzt, um festzulegen, auf welchem Gerät sich die Datei befindet. Peripheriegeräte wie z. B. Kassetteneinheiten, Diskettenstationen oder Drucker antworten zusätzlich auf mehrere Sekundäradressen. Stellen Sie sich diese als Codes vor, die dem Gerät mitteilen, welche Operation ausgeführt werden soll. Die logische Filenummer des Geräts wird mit jeder Anweisung GET#, INPUT# und PRINT# benutzt.

Wird die <Gerätenummer> ausgelassen, nimmt der Computer automatisch an, daß Sie Informationen zu Gerätenummer 1 übertragen bzw. von dort empfangen wollen, d. h., von der Datasette™. Auch der Dateiname kann ausgelassen werden. In diesem Fall können Sie jedoch später in Ihrem Programm die Datei nicht mit ihrem Namen aufrufen. Beim Speichern von Dateien auf Kassetten nimmt der Computer an, daß die <Sekundäradresse> Null (0) ist, wenn diese ausgelassen wird (eine READ-Operation).

Der Sekundäradressenwert eins (1) öffnet Kassettendateien zum Schreiben. Durch die Sekundäradresse zwei (2) wird ein Kassettenendkennzeichen geschrieben, wenn die Datei später geschlossen wird. Dieses Kennzeichen verhindert, daß aus Versehen über das Dateiende hinaus gelesen und somit die BASIC-Fehlermeldung **?DEVICE NOT PRESENT** angezeigt wird.

Bei Disketten stehen für Daten-Files die Sekundäradressen 2 bis 14 zur Verfügung. Andere Zahlen haben eine besondere Bedeutung in den DOS-Befehlen. Wenn Sie mit der Disketten-Station arbeiten, müssen Sie eine Sekundäradresse benutzen. (Bezüglich Einzelheiten über die DOS-Befehle schlagen Sie bitte in Ihrem Handbuch der Diskettenstation nach.)

Der <Dateiname> besteht aus einem String von 1 bis 16 Zeichen. Beim <Modus>=R werden sequentielle Dateien zum Lesen und beim <Modus>=W zum Schreiben geöffnet.

Wird versucht, auf eine Datei vor dem Öffnen zuzugreifen, so wird die BASIC-Fehlermeldung **?FILE NOT OPEN** angezeigt. Wird versucht, eine nicht existierende Datei zum Lesen zu öffnen, so wird die Fehlermeldung **?FILE NOT FOUND** angezeigt. Wird eine Datei auf Diskette zum Schreiben geöffnet und der Dateiname existiert bereits, dann erscheint die DOS-Fehlermeldung **FILE EXISTS.** Für Dateien auf Kassetten gibt es keinerlei Überprüfungsmöglichkeit, so daß Sie stets sicherstellen müssen, daß die Kassette richtig eingelegt ist. Andernfalls können bereits gespeicherte Daten versehentlich überschrieben werden. Wird eine bereits geöffnete Datei neu geöffnet, so wird die BASIC-Fehlermeldung **FILE OPEN** angezeigt.

BEISPIELE DER OPEN-ANWEISUNGEN:

10 OPEN 2, 8, 4 "DISK-OUTPUT SEQ, W"	(Öffnet sequentielle Datei auf Diskette)
10 OPEN 1, 1, 2, "TAPE-WRITE"	(Schreiben des Dateiendekennzeichens)
10 OPEN 50, 0	(Eingabe über die Tastatur)
10 OPEN 12, 3	(Bildschirmausgabe)
10 OPEN 130, 4	(Druckerausgabe)
10 OPEN 1, 1, 0, "NAME"	(Lesen von Kassette)
10 OPEN 1, 1, 1, "NAME"	(Schreiben auf Kassette)
10 OPEN 1, 2, 0, CHR$ (10)	(Kanal zu RS-232 öffnen)
10 OPEN 1, 4, 0, "STRING"	(Großbuchstaben/Graphiken zum Drucker senden)
10 OPEN 1, 4, 7, "STRING"	(Klein-/Großschrift zum Drucker schicken)
10 OPEN 1, 5, 0, "STRING"	(Großbuchstaben/Graphiken zum Drucker mit der Gerätenummer #5 schicken)
10 OPEN 1, 8, 15, "COMMAND"	(Einen Befehl zur Diskette schicken)

OR

TYP: Logischer Operator
FORMAT: <Operand> OR <Operand>

Funktion: So wie Vergleichsoperatoren für Entscheidungen hinsichtlich des Programmablaufs benutzt werden können, können logische Operatoren zwei oder mehrere Ausdrücke miteinander verbinden und die Meldungen "richtig" oder "falsch" ausgeben, die danach in einer Entscheidung benutzt werden können. In Berechnungen gibt das logische OR das Bitergebnis 1, wenn das entsprechende Bit von einem oder beiden Operanden 1 ist. Hierdurch entsteht je nach den Operandenwerten als Ergebnis eine ganze Zahl. In Vergleichen wird der logische Operator OR auch benutzt, um zwei Ausdrücke zu einem Ausdruck zu verketten. Ist einer der Ausdrücke richtig, so ist der Wert des zusammengesetzten Ausdrucks richtig (−1). Ist in nachstehendem ersten Beispiel AA gleich BB oder ist XX gleich 20, dann ist der Ausdruck richtig.
Logische Operatoren wandeln ihre Operanden in 16-Bit ganzzahlige Zweierkomplemente mit Vorzeichen aus dem Bereich −32768 bis 32767 um. Liegen die Operanden nicht in diesem Bereich, so wird eine Fehlermeldung angezeigt. Jedes Bit des Ergebnisses wird durch die entsprechenden Bits in den beiden Operanden bestimmt.

BEISPIELE DES OR-OPERATORS:

```
100 IF (AA = BB) OR (XX = 20) THEN . . .
```

```
230 KK% = 64 OR 32: PRINT KK%
```

(Sie geben dies mit einem Bit-Wert von 1000000 für 64 und 100000 für 32 ein.)

(Der Computer antwortet mit dem Bit-Wert 1100000. 1100000=96.)

PEEK

TYP: Ganzzahl-Funktion
FORMAT: PEEK (<numerisch>)

Funktion: Gibt eine ganze Zahl im Bereich von 0 bis 255 wieder, die aus einem Speicherplatz gelesen wird. Der <numerische> Ausdruck ist ein Speicherplatz, der in dem Bereich von 0 bis 65535 liegen muß. Andernfalls wird die BASIC-Fehlermeldung **?ILLEGAL QUANTITY** angezeigt.

BEISPIELE DER PEEK-FUNKTION:

10 PRINT PEEK (53280) AND 15	(Gibt den Wert der Bildschirmrahmenfarbe wieder.)
5 A%=PEEK(45)+PEEK(46)*256	(Gibt die Adresse der BASIC-Variablentabelle wieder.)

POKE

TYP: Anweisung
FORMAT: POKE <Adresse>, <Wert>

Funktion: Die POKE-Anweisung wird benutzt, um einen 1-Byte-Binärwert (8 Bits) in einen gegebenen Speicherplatz oder ein Ein-/Ausgaberegister zu schreiben. Die <Adresse> ist ein arithmetischer Ausdruck, der im Bereich von 0 bis 65535 liegen muß. Der <Wert> ist ein Ausdruck, der einer ganzen Zahl von 0 bis 255 entsprechen muß. Liegt einer der Werte nicht im angegebenen Bereich, wird die BASIC-Fehlermeldung **?ILLEGAL QUANTITY** angezeigt.
Die Anweisungen POKE und PEEK sind nützlich für Datenspeicherung, Steuerung der Graphikanzeige oder Geräuscherzeugung, für das Laden von Assembler-Unterprogrammen und zum Übertragen von Argumenten und Ergebnissen zu bzw. von Assembler-Unterprogrammen. Darüber hinaus können Betriebssystemparameter mit den PEEK-Anweisungen überprüft oder mit den POKE-Anweisungen verändert werden. Anhang G gibt eine komplette Liste der nützlichen Adressen.

BEISPIELE DER POKE-ANWEISUNG:

```
POKE 1024, 1          (Setzt ein "A" in Bildschirmposition 1)
POKE 2040, PTR        (Aktualisiert Datenzeiger-Sprite #0)
10 POKE RED, 32
20 POKE 36897, 8
2050 POKE A, B
```

POS

TYP: Ganzzahlige Funktion
FORMAT: POS (<Hilfsargument>)

Funktion: Teilt Ihnen die derzeitige Cursorposition mit, die natürlich in dem Bereich von 0 (äußerst linkes Zeichen) bis 79 in einer logischen Bildschirmzeile von 80 Zeichen liegt. Da der COMMODORE 64 einen 40-Zeichen-Bildschirm hat, beziehen sich Positionen von 40 bis 79 auf die zweite Bildschirmzeile. Das Hilfsargument wird überlesen.

BEISPIEL DER POS-FUNKTION:

```
1000 IF POS(0) >38 THEN PRINT CHR$(13)
```

PRINT

TYP: Anweisung
FORMAT: PRINT [<Variable>] [<,/;><Variable>] . . .

Funktion: Die PRINT-Anweisung wird normalerweise benutzt, um Daten auf dem Bildschirm anzuzeigen. Um diese Ausgabe auf ein anderes Gerät des Systems umzuleiten, wird die CMD-Anweisung benutzt. Die <Variable/n> in der Ausgabeliste sind beliebige Ausdrücke. Ist keine Ausgabeliste vorhanden, so wird eine leere Zeile angezeigt. Die Position jedes angezeigten Zeichens wird durch die Interpunktionszeichen bestimmt, die zum Trennen der einzelnen Werte in der Ausgabeliste benutzt werden.

Die zur Verfügung stehenden Interpunktionszeichen sind Leerzeichen, Kommata oder Semikolon. Die logische Bildschirmzeile von 80 Zeichen wird in acht Druckzonen mit je 10 Zeichen unterteilt. In der Ausdrucksliste wird durch ein Komma der nächste Wert am Anfang der nächsten Zone angezeigt. Durch ein Semikolon wird der nächste Wert sofort nach dem vorherigen Wert angezeigt. Es gibt jedoch zwei Ausnahmen:

1) Numerische Ausdrücke werden von einem Leerzeichen gefolgt.
2) Vor positiven Zahlen steht ein Leerzeichen.

Werden zwischen Stringkonstanten oder Variablennamen Leerzeichen oder keine Interpunktionszeichen benutzt, so hat dies die gleiche Wirkung wie ein Semikolon. Leerzeichen zwischen einer Zeichenkette und einem numerischen Ausdruck oder zwischen zwei numerischen Ausdrücken stoppen jedoch die Ausgabe, ohne daß der zweite Wert angezeigt wird.
Steht am Ende der Ausgabeliste ein Komma oder ein Semikolon, so beginnt die nächste PRINT-Anweisung mit der Anzeige auf der gleichen Zeile und ist entsprechend abgetrennt. Steht am Ende der Liste kein Interpunktionszeichen, werden am Ende der Daten ein Wagenrücklauf und ein Zeilenvorschub angezeigt. Die PRINT-Anweisung beginnt in der nächsten Zeile. Wird ihre Ausgabe auf den Bildschirm geleitet und sind die angezeigten Daten länger als 40 Zeichen, so wird die Ausgabe in der nächsten Bildschirmzeile fortgesetzt.
Die PRINT-Anweisung ist die BASIC-Anweisung, die am vielseitigsten eingesetzt werden kann. Es gibt für diese Anweisung so viele Symbole, Funktionen und Parameter, daß man fast schon von einer eigenen, speziell zum Schreiben auf dem Bildschirm entworfenen Sprache innerhalb von BASIC sprechen kann.

BEISPIELE DER PRINT-ANWEISUNG:

1)

```
5 X = 5
10 PRINT −5*X, X−5, X+5, X↑5

−25          0          10          3125
```

2)

```
5 X=9
10 PRINT X;"SQUARED IS";X*X;"AND";
20 PRINT X "CUBED IS" X↑3

 9 SQUARED IS 81 AND 9 CUBED IS 729
```

3)

```
90 AA$="ALPHA":BB$="BAKER": CC$="CHARLIE":DD$="DOG":
   EE$="ECHO"
100 PRINT AA$BB$;CC$DD$,EE$

ALPHABAKERCHARLIEDOG        ECHO
```

ANFÜHRUNGSZEICHEN

Wenn ein Anführungszeichen (SHIFT 2) eingegeben ist, stoppt die Cursor-Steuerung, und die Steuerzeichen der Cursor-Steuertasten werden angezeigt. Auf diese Weise können Sie Cursorsteuerungen programmieren, da die Cursor-Funktionen beim Ausdruck des Textes mit ausgeführt werden. Die einzige Cursor-Steuertaste, die nicht durch diesen "Anführungsmodus" beeinflußt wird, ist die Taste INST/DEL

1. Cursorbewegung

Folgende Cursorsteuerungen können im Anführungszeichenmodus "programmiert" werden.

TASTE	ERSCHEINT ALS
CLR/HOME	S
SHIFT CLR/HOME	♥
⇑ CRSR ⇓	Q
SHIFT ⇑ CRSR ⇓	●
⇐ CRSR ⇒	]
SHIFT ⇐ CRSR ⇒	▌

Soll das Wort HELLO diagonal von der oberen linken Bildschirmecke aus angezeigt werden, geben Sie folgendes ein:

PRINT" CLR /HOME H ⇑ CRSR ⇓ E ⇑ CRSR ⇓ L ⇑ CRSR ⇓ L ⇑ CRSR ⇓ O"

Dies erscheint als:

PRINT" S H Q E Q L Q L Q O"

2. Unterlegte (negativ dargestellte) Zeichen

Durch gemeinsames Drücken der Tasten CTRL und 9 – nach Anführungszeichen – wird R angezeigt. Auf diese Weise werden jetzt alle Zeichen ähnlich einem Negativbild unterlegt angezeigt. Um dies zu beenden, sind die Tasten CTRL und Ø zu drücken (wodurch ▭ angezeigt wird), oder Sie geben RETURN ein (CHR$(13)) (beenden Sie hierzu einfach die PRINT-Anweisung ohne Semikolon oder Komma).

3. Farbsteuerungen

Durch gemeinsames Drücken der Taste CTRL oder C= mit einer der acht Farbtasten, erscheint ein besonderes unterlegtes Zeichen in den Anführungszeichen. Beim Ausdruck erscheint die Schrift dann in der ausgewählten Farbe.

TASTE	FARBE	ERSCHEINT ALS
CTRL 1	Schwarz	
CTRL 2	Weiß	
CTRL 3	Rot	
CTRL 4	Zyan	
CTRL 5	Purpur	
CTRL 6	Grün	
CTRL 7	Blau	
CTRL 8	Gelb	
C= 1	Orange	
C= 2	Braun	
C= 3	Hellrot	
C= 4	Grau 1	
C= 5	Grau 2	
C= 6	Hellgrün	
C= 7	Hellblau	
C= 8	Grau 3	

Soll das Wort HELLO in Zyan und THERE in Weiß angezeigt werden, geben Sie folgendes ein:

PRINT" CTRL 4 HELLO CTRL 2 THERE"

Dies erscheint als:

PRINT" HELLO E THERE"

4. Einfügemodus

Die über die Taste INST/DEL erzeugten Leerstellen haben die gleichen Eigenschaften wie der Anführungszeichen-Modus. Die Cursor- und Farbsteuerungen erscheinen als unterlegte Zeichen. Der einzige Unterschied besteht darin, daß SHIFT / INST ein T hervorruft.
Außerdem fügt die Taste INST , die normalerweise im Anführungszeichen-Modus ein Sonderzeichen erstellt, Leerzeichen ein.
Der "Einfüge-Modus" wird durch Anschlagen der Taste RETURN oder SHIFT RETURN beendet, oder wenn so viele Zeichen eingegeben wurden, wie Leerzeichen eingefügt sind.

5. Weitere Sonderzeichen

Es gibt einige andere Zeichen, die für spezielle Funktionen ausgegeben werden können, auch wenn sie nicht einfach über die Tastatur zur Verfügung stehen. Um diese in Anführungszeichen zu setzen, müssen Sie in der Zeile entsprechende Leerstellen lassen, RETURN oder SHIFT RETURN drücken und mit der Cursorsteuertaste zurück in die Leerstelle gehen. Nun drücken Sie die Taste CTRL und RVS/ON , um die umgekehrten Zeichen anzuzeigen, und schlagen folgende Tasten an:

FUNKTION	TASTENBETÄTIGUNG
SHIFT RETURN	SHIFT M
Umschalten zu Zeichen mit SHIFT	N
Umschalten zu Zeichen ohne SHIFT	SHIFT N
Umschalttasten nicht wirksam	H
Umschalttasten wirksam	I

SHIFT RETURN ist sowohl bei LIST als auch bei PRINT möglich, so daß bei Verwendung dieses Zeichens ein Editieren so gut wie unmöglich ist. Auch die Auflistung wird merkwürdig aussehen.

PRINT#

TYP: Ein-/Ausgabeanweisung
FORMAT: PRINT# <logische Filenummer> [<Variable>] [<,/;><Variable>] . . .

Funktion: Die Anweisung PRINT# wird benutzt, um Daten in eine Datei zu schreiben. Die Nummer muß die gleiche sein wie beim Öffnen der Datei. Die Ausgabe erfolgt auf die Gerätenummer, die in der OPEN-Anweisung benutzt wurde. Der <Variablen-Ausdruck> der Ausgabeliste kann beliebig gewählt werden. Die Interpunktionszeichen zwischen den einzelnen Werten sind die gleichen wie bei der PRINT-Anweisung und werden auf die gleiche Weise benutzt. Die Wirkung der Interpunktionszeichen ist jedoch aus zwei wesentlichen Gründen unterschiedlich. Wird PRINT# bei Kassettendateien benutzt, so hat das Komma die gleiche Wirkung wie ein Semikolon. Es ist daher stets gleich, ob Leerzeichen, Kommata, Semikolons oder keine Interpunktionszeichen zwischen Daten benutzt werden. Die Daten werden als kontinuierliche Zeichenkette geschrieben. Nach numerischen Daten folgt ein Leerzeichen, und wenn sie positiv sind, steht auch vor ihnen ein Leerzeichen.

Wird die Liste durch keine Interpunktionszeichen beendet, so wird am Ende der Daten ein Wagenrücklauf oder Zeilenvorschub geschrieben. Wird die Ausgabeliste durch ein Komma oder Semikolon beendet, werden Wagenrücklauf und Zeilenvorschub unterdrückt. Unabhängig von der Interpunktion beginnt die nächste Anweisung PRINT# die Ausgabe in der nächsten verfügbaren Zeichenposition. Der Zeilenvorschub wirkt als Stop, wenn die Anweisung INPUT# benutzt wird, und hinterläßt bei Ausführung der nächsten Anweisung INPUT# eine leere Variable. Der Zeilenvorschub kann entsprechend nachstehenden Beispielen unterdrückt oder ausgeglichen werden.

Die einfachste Art, um mehr als eine Variable in eine Datei auf Kassette oder Diskette zu schreiben, ist eine Stringvariable gleich CHR$(13) zu setzen, und dieses beim Schreiben der Datei zwischen alle anderen Variablen zu setzen.

BEISPIELE DER ANWEISUNG PRINT#:

1)

```
10 OPEN 1, 1, 1, "TAPE FILE"
20 R$ = CHR$(13)
30 PRINT# 1,1;R$;2;R$;3;R$;4;R$;5
40 PRINT# 1,6
50 PRINT# 1,7
```

(Durch Änderung von CHR$(13) in CHR$(44) wird zwischen jede Variable ein "," gesetzt. CHR$(59) setzt ein "Semikolon" zwischen jede Variable.)

2)

```
10 CO$=CHR$(44): CR$=CHR$(13)
20 PRINT#1, "AAA"CO$"BBB",
   "CCC";"DDD";"EEE"CR$"FFF"CR$;
30 INPUT#1, A$,BCDE$,F$
```

AAA,BBB CCCDDDEEE
(Wagenrücklauf)
(Wagenrücklauf)

3)

```
 5 CR$=CHR$(13)
10 PRINT#2, "AAA";CR$;"BBB"
20 PRINT#2, "CCC";

30 INPUT#2, A$,B$,DUMMY$,C$
```

(10 Leerzeichen)AAA
BBB

(10 Leerzeichen)CCC

READ

TYP: Anweisung
FORMAT: READ <Variable> [,<Variable>]

Funktion: Die READ-Anweisung wird benutzt, um den Variablennamen Konstanten aus den DATA-Anweisungen zuzuordnen. Die einzulesenden Daten müssen mit den angegebenen Variablentypen übereinstimmen. Andernfalls wird die BASIC-Fehlermeldung **?SYNTAX ERROR** angezeigt.* Variablen in DATA-Eingabelisten müssen durch Kommata getrennt werden.
Eine einzelne READ-Anweisung kann nacheinander auf eine oder mehrere DATA-Anweisungen zugreifen (siehe DATA). Oder es können mehrere READ-Anweisungen auf die gleiche DATA-Anweisung zugreifen. Werden mehr READ-Anweisungen ausgeführt, als die DATA-Anweisungen im Programm an Elementen enthalten, wird die BASIC-Fehlermeldung **?OUT OF DATA** angezeigt.

Ist die festgelegte Variablenzahl kleiner als die Zahl der Elemente in den DATA-Anweisungen, so setzen die folgenden READ-Anweisungen beim nächsten Datenelement ein. (Siehe RESTORE.)

***Anmerkung:** **?SYNTAX ERROR** erscheint mit der Zeilennummer der DATA-Anweisung und nicht der READ-Anweisung.

BEISPIELE DER READ-ANWEISUNG:

```
110 READ A,B,C$
120 DATA 1,2,HELLO

100 FOR X=1 TO 10: READ A(X):NEXT

200 DATA 3.08, 5.19, 3.12, 3.98, 4.24
210 DATA 5.08, 5.55, 4.00, 3.16, 3.37
```

(Füllt die Variablen (Zeile 1) in Reihenfolge der gezeigten Konstanten (Zeile 5))

```
1 READ CITY$,STATE$,ZIP

5 DATA DENVER,COLORADO, 80211
```

REM

TYP: Anweisung
FORMAT: REM [<Bemerkung>]

Funktion: Über die REM-Anweisung wird Ihr Programm bei der Auflistung verständlicher. Es erinnert Sie daran, welchen Zweck Sie mit den einzelnen Programmabschnitten verfolgten. So können Sie z. B. darauf hinweisen, wofür eine Variable benutzt wird usw. Diese Bemerkung kann ein beliebiger Text, ein Wort oder ein Zeichen einschließlich dem Doppelpunkt (:) oder BASIC-Schlüsselwörter sein. Die REM-Anweisung und alles Folgende in der gleichen Zeilennummer werden von BASIC überlesen. Die Bemerkungen werden jedoch bei der Programmauflistung genau wie eingegeben angezeigt. Auf eine REM-Anweisung kann durch eine GOTO- oder GOSUB-Anweisung Bezug genommen werden. Die Programmausführung setzt dann mit der nächsthöheren Programmzeile fort, die eine ausführbare Anweisung enthält.

BEISPIELE DER REM-ANWEISUNG:

```
10 CALCULATE AVERAGE VELOCITY
20 FOR X=1 TO 20 :REM LOOP FOR TWENTY VALUES
30 SUM=SUM + VEL(X): NEXT
40 AVG=SUM/20
```

RESTORE

TYP: Anweisung
FORMAT: RESTORE

Funktion: BASIC stellt den internen Zeiger (Pointer) auf die nächste zu lesende DATA-Konstante. Dieser Pointer kann in einem Programm über die RESTORE-Anweisung zur ersten DATA-Konstante zurückgestellt werden. Die RESTORE-Anweisung kann an einer beliebigen Programmstelle benutzt werden.

BEISPIELE DER RESTORE-ANWEISUNG:

```
100 FOR X=1 TO 10: READ A(X): NEXT
200 RESTORE
300 FOR Y=1 TO 10: READ B(Y): NEXT

4000 DATA 3.08, 5.19, 3.12, 3.98, 4.24
4100 DATA 5.08, 5.55, 4.00, 3.16, 3.37
```

(Füllt die beiden Felder mit identischen Daten)

```
10 DATA 1,2,3,4
20 DATA 5,6,7,8
30 FOR L=1 TO 8
40 READ A: PRINT A
50 NEXT
60 RESTORE
70 FOR L=1 TO 8
80 READ A: PRINT A
90 NEXT
```

RETURN

TYP: Anweisung
FORMAT: RETURN

Funktion: Die RETURN-Anweisung wird benutzt, um aus einem Unterprogramm, das durch eine GOSUB-Anweisung aufgerufen wurde, zurückzuspringen. Durch RETURN wird der Rest Ihres Programms ab der Anweisung nach dem entsprechenden GOSUB wieder gestartet. Bei der Verschachtelung von Unterprogrammen muß für jedes GOSUB mindestens eine RETURN-Anweisung vorhanden sein. Ein Unterprogramm kann beliebig viele RETURN-Anweisungen enthalten. Durch die zuerst gelesene RETURN-Anweisung wird das Unterprogramm jedoch beendet.

BEISPIEL DER RETURN-ANWEISUNG:

```
10 PRINT "THIS IS THE PROGRAM"
20 GOSUB 1000
30 PRINT "PROGRAM CONTINUES"
40 GOSUB 1000
50 PRINT "MORE PROGRAM"
60 END
1000 PRINT "THIS IS THE GOSUB":RETURN
```

RIGHT$

TYP: Stringfunktion
FORMAT: RIGHT$ (<String>, <Ganze Zahl>)

Funktion: Über die Funktion RIGHT$ wird ein Teilstring von der rechten Seite einer Zeichenkette wiedergegeben. Die Länge des Teilstrings wird durch die ganze Zahl des Argumentes bestimmt. Diese Zahl kann zwischen 0 und 255 liegen. Ist diese Zahl Null, dann wird ein Leerstring (" ") wiedergegeben. Ist die ganze Zahl größer als die Länge des Strings, dann wird der ganze String wiedergegeben.

BEISPIEL DER FUNKTION RIGHT$:

```
10 MSG$ = "COMMODORE COMPUTERS"
20 PRINT RIGHT$(MSG$,9)
RUN

COMPUTERS
```

RND

TYP: Gleitpunktfunktion
FORMAT: RND (<numerisch>)

Funktion: RND erstellt eine Gleitpunktzahl von 0.0 bis 1.0. Der Computer erstellt Zufallszahlen ausgehend von einer Zahl, die im Computer-Jargon "seed" (Samen) genannt wird. Die RND-Funktion wird bei Einschalten der Stromversorgung aktiviert. Das <numerische> Argument ist, außer in bezug auf das Vorzeichen (positiv, Null oder negativ), ein Hilfsargument. Ist das <numerische> Argument positiv, so wird die gleiche "Pseudozufall"-Folge von Zahlen, angefangen bei einem gegebenen Seed-Wert, wiedergegeben. Unterschiedliche Zahlenfolgen ergeben sich aus unterschiedlichen Seeds, jede Folge kann jedoch ab der gleichen Seed-Nummer wiederholt werden. Für das Überprüfen von Programmen ist es sinnvoll, eine bereits bekannte Folge von "Zufallszahlen" zu haben.
Wird das <numerische> Argument Null gewählt, erstellt RND eine Zahl direkt von einer freilaufenden Hardware-Uhr, der System-"Jiffy-Clock". Durch negative Argumente gilt für die RND-Funktion bei jedem Funktionsaufruf eine Seed-Rückstellung.

BEISPIEL DER RND-FUNKTION:

220 PRINT INT(RND(0)*50)	(Wiedergabe von ganzen Zufallszahlen 0–49)
100 X=INT(RND(1)*6)+INT(RND(1)*6)+2	(Simuliert 2 Würfel)
100 X=INT(RND(1)*1000)+1	(Ganze Zufallszahl von 1 bis 1000)
100 X=INT(RND(1)*150)+100	(Ganze Zufallszahl von 100 bis 249)
100 X=RND(1)*(U–L)+L	(Zufallszahlen zwischen oberer (U) und unterer (L) Grenze)

RUN

TYP: Befehl
FORMAT: RUN [<Zeilennummer>]

Funktion: Der Systembefehl RUN wird benutzt, um ein im Speicher befindliches Programm auszuführen. Über den Befehl RUN wird vor dem Programmstart ein CLR ausgeführt. Zur Vermeidung der Löschung kann das Programm durch CONT oder GOTO anstelle von RUN wieder gestartet werden. Wird eine <Zeilennummer> angegeben, so beginnt das Programm in dieser Zeile. Ansonsten beginnt der RUN-Befehl die Ausführung mit der ersten Programmzeile. Dieser Befehl kann auch innerhalb eines Programms benutzt werden. Wenn die angegebene Zeilenzahl nicht existiert, wird die BASIC-Fehlermeldung **UNDEF'D STATEMENT** angezeigt. Die Programmausführung stoppt, und BASIC kehrt zum Direktmodus zurück, wenn eine END- oder STOP-Anweisung erreicht wird, wenn die letzte Programmzeile beendet wird oder wenn während der Ausführung ein BASIC-Fehler auftritt.

BEISPIELE DES RUN-BEFEHLS:

RUN	(Beginnt bei der ersten Programmzeile)
RUN 500	(Beginnt bei Zeilennummer 500)
RUN X	(Beginnt bei Zeile X. Wenn keine Zeile X existiert, wird die Meldung UNDEF'D STATEMENT ERROR angezeigt.)

SAVE

TYP: Befehl
FORMAT: SAVE ["<Dateiname>"] [,<Gerätenummer>] [,<Adresse>]

Funktion: Über den SAVE-Befehl wird das im Speicher befindliche Programm auf Kassette oder Diskette gespeichert. Das Programm bleibt im derzeitigen Computerspeicher auch nach diesem Zeichenvorgang noch erhalten. Der Dateityp ist PRG (Programm). Wird die <Gerätenummer> ausgelassen, nimmt der C 64 automatisch an, daß das Programm auf Kassette, d. h. Gerätenummer 1, gespeichert werden soll. Ist die <Gerätenummer> eine <8>, wird das Programm auf Diskette geschrieben.
Die SAVE-Anweisung können Sie auch in Ihren Programmen benutzen. Die Ausführung fährt nach Beendigung dieser Speicherung mit der nächsten Anweisung fort.

Programme auf Kassette werden automatisch doppelt gespeichert, so daß der COMMODORE 64 beim erneuten Laden des Programms eine Fehlerüberprüfung durchführen kann. Werden Programme auf Kassette gespeichert, dann wird der <Dateiname> und die <Sekundär-Adresse> optional. Wird bei der Speicherung ein Programmname jedoch in Anführungszeichen (" ") oder durch eine Stringvariable (---$) angegeben, kann der COMMODORE 64 die einzelnen Programme leichter finden. Wird der Dateiname ausgelassen, so ist danach kein Laden mit Namen möglich.
Beim Speichern von Programmen auf Diskette muß der <Dateiname> angegeben sein.

BEISPIELE DES SAVE-BEFEHLS:

SAVE	(Schreiben auf Kassette ohne Namen)
SAVE "ALPHA", 1	(Speichern auf Kassette mit Dateiname "alpha")
SAVE "ALPHA", 1, 2	(Speichern von "alpha" mit Kassettenende-Kennzeichen)
SAVE "FUN.DISK",8	(Speichern auf Diskette (Geräte-Nr. 8))
SAVE A$	(Speichern auf Kassette mit der Bezeichnung A$)
10 SAVE "HI"	(Speichern des Programms und Übergang zur nächsten Programmzeile)
SAVE "ME",1,3	(Speichern im gleichen Speicherplatz und Setzen eines Kassettenende-Kennzeichens)

SGN

TYP: Ganzzahlige Funktion
FORMAT: SGN (<numerisch>)

Funktion: SGN gibt Ihnen abhängig vom Vorzeichen des <numerischen> Argu-ments einen ganzzahligen Wert. Ist das Argument positiv, dann ist das Ergebnis 1. Ist das Argument 0, dann ist auch das Ergebnis 0, und bei negativem Argument lautet das Ergebnis −1.

BEISPIEL DER SGN-FUNKTION:

```
90 ON SGN(DV)+2 GOTO 100, 200, 300
```

(Sprung zu 100, wenn DV=negativ, zu 200, wenn DV=0, zu 300, wenn DV=positiv)

SIN

TYP: Gleitpunktfunktion
FORMAT: SIN (<numerisch>)

Funktion: SIN gibt Ihnen den Sinus des <numerischen> Arguments, das in Bogenmaß anzugeben ist. Der Wert von COS(x) ist gleich SIN(x+3.14159265/2).

BEISPIEL DER SIN-FUNKTION:

```
235 AA = SIN(1.5): PRINT AA
.997494987
```

SPC

TYP: Stringfunktion
FORMAT: SPC (<numerisch>)

Funktion: Mit der SPC-Funktion wird die Datenformatierung entweder als Ausgabe auf dem Bildschirm oder in eine logische Datei überprüft. Die Anzahl der durch das <numerische> Argument angegebenen Leerstellen (SPC) wird beginnend bei der ersten verfügbaren Position angezeigt. Für Bildschirm- oder Kassettendateien liegt der Wert des Arguments zwischen 0 und 255 und für Diskettendateien bei max. 254. Bei Druckerdateien wird automatisch ein Wagenrücklauf und Zeilenvorschub ausgeführt, wenn die letzte Zeichenposition einer Zeile ein Leerzeichen ist.

BEISPIEL DER SPC-FUNKTION:

```
10 PRINT "RIGHT"; "HERE &";
20 PRINT SPC(5) "OVER" SPC(14) "THERE"
RUN

RIGHT HERE &     OVER                THERE
```

SQR

TYP: Gleitpunktfunktion
FORMAT: SQR (<numerisch>)

Funktion: SQR gibt Ihnen den Wert der Quadratwurzel des <numerischen> Arguments. Der Argumentenwert darf nicht negativ sein, da sonst die BASIC-Fehlermeldung **?ILLEGAL QUANTITY** angezeigt wird.

BEISPIEL DER SQR-FUNKTION:

```
FOR J = 2 TO 5: PRINT J*5, SQR(J * 5): NEXT

10        3.16227766
15        3.87298335
20        4.47213595
25        5
          READY
```

STATUS

TYP: Ganzzahlige Funktion
FORMAT: STATUS

Funktion: Sagt etwas über die letzte Ein-/Ausgabeoperation aus, die bei einer offenen Datei durchgeführt wurde. Der STATUS kann von einem beliebigen Peripheriegerät gelesen werden.
Das Schlüsselwort STATUS (oder einfach ST) ist ein systemdefinierter Variablenname, in den der KERNAL den STATUS der Ein-/Ausgabeoperationen gibt. Nachstehend sehen Sie eine Tabelle der STATUS-Codewerte für Kassette, Drucker, Diskette und RS-232:

ST-Bit-position	ST numerischer Wert	Lesen von Kassette	Serieller Bus Lesen/ Schreiben	Kassette Verify and Load
0	1		Time out Schreiben	
1	2		Time out Lesen	
2	4	Kurzer Block		Kurzer Block
3	8	Langer Block		Langer Block
4	16	Nicht erkennbarer Lesefehler		Beliebige fehlende Übereinstimmung
5	32	Prüfsummen-fehler		Prüfsummen-fehler
6	64	Dateiende	Ende der Eingabe	
7	−128	Kassetten-ende	Gerät nicht vorhanden	Kassettenende

BEISPIELE DER STATUS-FUNKTION:

```
10 OPEN 1, 4: OPEN 2, 8, 4, "MASTER FILE,SEQ,W"
20 GOSUB 100: REM CHECK STATUS
30 INPUT#2, A$, B, C
40 IF STATUS AND 64 THEN 80: REM HANDLE END-OF-FILE
50 GOSUB 100: REM CHECK STATUS
60 PRINT#1, A$, B; C
70 GOTO 20
80 CLOSE1: CLOSE2
90 GOSUB 100: END
100 IF ST >0 THEN 9000: REM HANDLE FILE I/= ERROR
111 RETURN
```

STEP

TYP: Anweisung
FORMAT: [STEP <Ausdruck>]

Funktion: Das nicht notwendige STEP-Schlüsselwort folgt nach der <Grenze> in einer FOR-Anweisung. Es bestimmt einen Inkrementwert für die Schleifenzähler-Variable. Als STEP-Inkrement kann ein beliebiger Wert außer Null benutzt werden. Wird das STEP-Schlüsselwort ausgelassen, so ist der Inkrementwert +1. Wird die NEXT-Anweisung für eine FOR-Schleife erreicht, wird das STEP-Inkrement wirksam. Der Zähler wird gegen den Endwert überprüft, um festzustellen, ob die Schleife beendet ist. (Bezüglich weiterer Einzelheiten siehe FOR-Anweisung.)

Anmerkung: Der STEP-Wert kann nicht innerhalb der Schleife geändert werden.

BEISPIELE DER STEP-ANWEISUNG:

25 FOR XX = 2 TO 20 STEP 2	(Zehnmalige Wiederholung der Schleife)
35 FOR ZZ = 0 TO −20 STEP −2	(Elfmalige Wiederholung der Schleife)

STOP

TYP: Anweisung
FORMAT: STOP

Funktion: Die STOP-Anweisung wird benutzt, um die Ausführung des derzeitigen Programms zu stoppen und zum Direktmodus zurückzukehren. Die STOP-Anweisung hat die gleiche Funktion wie die Betätigung der Taste RUN/STOP. Auf dem Bildschirm wird die BASIC-Fehlermeldung **?BREAK IN LINE** nnnnn und danach **READY** angezeigt. "nnnnn" gibt an, in welcher Zeilennummer die Programmausführung gestoppt wurde. Offene Dateien bleiben offen, und alle Variablen können überprüft werden. Das Fortsetzen des Programmes ist über die Anweisung CONT oder GOTO möglich.

BEISPIELE DER STOP-ANWEISUNG:

```
10 INPUT#1, AA, BB, CC
20 IF AA = BB AND BB = CC THEN STOP
30 STOP

(Ist die Variable AA gleich −1 und BB gleich CC, dann:)

BREAK IN LINE 20        (Für beliebige andere Datenwerte)
BREAK IN LINE 30
```

STR$

TYP: Stringfunktion
FORMAT: STR$ (<numerisch>)

Funktion: STR$ wandelt das numerische Argument in einen String um. Ist das Argument positiv oder 0, so beginnt der String mit einem Leerzeichen.

BEISPIEL DER FUNKTION STR$:

```
100 FLT = 1.5E4: ALPHA$ = STR$(FLT)
110 PRINT FLT, ALPHA$

15000     15000
```

SYS

TYP: Anweisung
FORMAT: SYS <Adresse>

Funktion: Dies ist die am weitesten verbreitete Art, ein BASIC-Programm mit einem Maschinensprache-Programm zu kombinieren. Das Programm in Maschinensprache beginnt an der durch die SYS-Anweisung angegebenen Adresse. Der Systembefehl SYS wird entweder im Direkt- oder Programmodus benutzt, um die Steuerung des Mikroprozessors zu einem im Speicher existierenden Maschinensprache-Programm zu übertragen. Die Adresse wird durch einen numerischen Ausdruck angegeben und kann an einem beliebigen RAM- oder ROM-Speicherplatz liegen.
Wenn Sie die SYS-Anweisung benutzen, muß dieser Abschnitt des Maschinensprache-Codes mit einer RTS-Anweisung (Rückkehr vom Unterprogramm) beendet werden, damit nach Ausführung des Maschinensprache-Programms die BASIC-Ausführung mit der Anweisung hinter dem SYS-Befehl fortgesetzt wird.

BEISPIELE DER SYS-ANWEISUNG:

SYS 64738	(Sprung zum System-Kaltstart im ROM)
10 POKE 4400,96: SYS 4400	(Geht zum Maschinencode-Platz 4400 und kehrt sofort zurück)

TAB

TYP: Stringfunktion
FORMAT: TAB (<numerisch>)

Funktion: Durch die TAB-Funktion wird der Cursor zu der durch das <numerische> Argument angegebenen Bildschirm-Position (gezählt ab der äußerst linken Position der derzeitigen Zeile) bewegt. Der Wert des Arguments kann zwischen 0 und 255 liegen. Die TAB-Funktion sollte nur mit der PRINT-Anweisung benutzt werden, da sie zusammen mit PRINT# für eine logische Datei unwirksam ist.

BEISPIELE DER TAB-FUNKTION:

```
100 PRINT "NAME" TAB(25) "AMOUNT": PRINT
110 INPUT#1, NAM$, AMT$
120 PRINT NAM$ TAB(25) AMT$
```

```
NAME                    AMOUNT

G.T.JONES               25.
```

TAN

TYP: Gleitpunktfunktion
FORMAT: TAN (<numerisch>)

Funktion: Gibt den Tangens des <numerischen> Ausdrucks, der einen Winkel in Bogenmaß darstellt, wieder. Beim Überlauf der TAN-Funktion wird die BASIC-Fehlermeldung **?DIVISION BY ZERO** angezeigt.

BEISPIEL DER TAN-FUNKTION:

```
10 XX = .785398163: YY = TAN(XX): PRINT YY
1
```

TIME

TYP: Numerische Funktion
FORMAT: TI

Funktion: Bei der TI-Funktion wird der Intervalltimer gelesen. Dieser Typ wird "Jiffy Clock" genannt. Der "Jiffy Clock"-Wert wird bei Einschalten der Stromversorgung auf Null gestellt (initialisiert). Dieser 1/60-Sekunden-Intervalltimer wird während der Kassettenein-/ausgabe abgeschaltet.

BEISPIEL DER TI-FUNKTION:

```
10 PRINT TI/60 "SECONDS SINCE POWER UP"
```

TIME$

TYP: Stringfunktion
FORMAT: TI$

Funktion: Der Timer arbeitet genau wie eine normale Uhr, solange das System eingeschaltet ist. Der Hardware-Intervalltimer (oder "Jiffy Clock") wird gelesen und zur Aktualisierung des Werts von TI$ benutzt. Hierdurch entsteht ein Zeit-String (TI$) von sechs Zeichen in Stunden, Minuten und Sekunden. Dem Timer kann, ähnlich wie bei Ihrer Armbanduhr, ein beliebiger Startpunkt zugeordnet werden. Der Wert von TI$ ist nach der Kassettenein-/ausgabe nicht mehr präzise.

BEISPIEL DER FUNKTION TI$:

```
1 TI$ = "000000": FOR J=1 TO 10000: NEXT: PRINT TI$

000011
```

USR

TYP: Gleitpunktfunktion
FORMAT: USR (<numerisch>)

Funktion: Über die USR-Funktion wird zu einem Maschinensprache-Unterprogramm gesprungen, das vom Benutzer aufgerufen werden kann und dessen Startadresse durch die Inhalte der Speicherplätze 785–786 angezeigt ist. Die Startadresse wird vor dem Aufruf der USR-Funktion durch POKE-Anweisung in die Adressen 785 und 786 eingegeben. Wurden die POKE-Anweisungen nicht durchgeführt, so erfolgt die Fehlermeldung **?ILLEGAL QUANTITY.**
Der Wert des <numerischen> Arguments wird im Gleitpunkt-Akkumulator (Startadresse 97) für die Verwendung im Assembler-Code gespeichert, und das Ergebnis der USR-Funktion wird an diesem Platz bei der Rückkehr vom Unterprogramm zu BASIC gespeichert.

BEISPIELE DER USR-FUNKTION:

```
10 B = T * SIN(Y)
20 C = USR (B/2)
30 D = USR (B/3)
```

VAL

TYP: Numerische Funktion
FORMAT: VAL (<String>)

Funktion: Gibt den numerischen Wert eines Strings wieder. Ist das erste nicht leere Zeichen des Strings nicht ein Pluszeichen (+), Minuszeichen (−) oder eine Zahl, ergibt sich der Wert Null. Die String-Umsetzung wird am Ende der Zeichenkette, oder wenn ein Nicht-Zahlenzeichen gefunden wird, beendet (mit Ausnahme des Dezimalpunkts oder des Exponenten e).

BEISPIEL DER VAL-FUNKTION:

```
10 INPUT#1, NAM$, ZIP$
20 IF VAL(ZIP$) <19400 OR VAL(ZIP$)> 96699 THEN PRINT NAM$ TAB(25)
   "GREATER PHILADELPHIA"
```

VERIFY

TYP: Befehl
FORMAT: VERIFY ["<Datenname>"] [,<Gerätenummer>]

Funktion: Der VERIFY-Befehl wird im Direkt- oder Programm-Modus benutzt, um die Inhalte von BASIC-Programmdateien auf Kassette oder Diskette mit dem derzeitigen im Speicher befindlichen Programm zu vergleichen. VERIFY wird normalerweise direkt nach der Speicherung (SAVE) benutzt, um sicherzustellen, daß das Programm korrekt gespeichert wurde.
Wird die Gerätenummer ausgelassen, gilt für das Programm die Gerätenummer 1 (Datasette). Wird für Kassettendateien der Dateiname ausgelassen, so wird das nächste auf der Kassette gefundene Programm verglichen. Für Diskettendateien (Geräte-Nr. 8) muß der Dateiname angegeben werden. Wird eine Abweichung vom Programmtext festgestellt, so erscheint die BASIC-Fehlermeldung **?VERIFY ERROR.**
Ein Programmname kann entweder in Anführungszeichen (" ") oder als Stringvariable angegeben werden. VERIFY wird auch benutzt, um ein Kassettenband hinter das letzte Programm zu spulen, so daß danach ein neues Programm abgespeichert werden kann. Auf diese Weise wird eine Programmüberschreibung vermieden.

BEISPIELE DES VERIFY-BEFEHLS:

```
VERIFY                          (Überprüft erstes Programm auf der
                                Kassette)

PRESS PLAY ON TAPE
OK
SEARCHING
FOUND <FILENAME>
VERIFYING

9000 SAVE "ME",8:               (Sucht das Programm bei Gerät Nr. 8)
9010 VERIFY "ME",8
```

WAIT

TYP: Anweisung
FORMAT: WAIT <Platz>, <Maske-1> [,<Maske-2>]

Funktion: Durch die WAIT-Anweisung wird die Programm-Ausführung so lange unterbrochen, bis eine gegebene Speicheradresse ein bestimmtes Bit-Muster erkennt. D. h., WAIT kann benutzt werden, um ein Programm so lange zu stoppen, bis eine externe Bedingung erfüllt ist. Dies erfolgt durch Überwachung des Status der Bits im Ein-/Ausgaberegister. Für die Daten von WAIT können beliebige numerische Ausdrücke gewählt werden. Diese werden jedoch in ganzzahlige Werte umgesetzt.

Die meisten Programmierer werden nie mit dieser Anweisung arbeiten. Hierdurch wird das Programm angehalten, bis die Bits eines bestimmten Speicherplatzes auf eine ganz bestimmte Weise verändert werden. Diese Anweisung wird fast ausschließlich für bestimmte Ein-/Ausgabevorgänge benutzt.

Die WAIT-Anweisung nimmt den Wert im Speicherplatz und führt eine logische UND-Verknüpfung mit dem Wert der Maske-1 durch. Enthält die Anweisung eine Maske-2, wird das Ergebnis des ersten Vorganges mit Maske-2 durch ein ausschließendes ODER verknüpft.

D. h., Maske-1 "filtert" beliebige Bits aus, die Sie nicht prüfen wollen. Ist das Bit in Maske-1 0, so ist das entsprechende Bit in Ihrem Ergebnis auch 0. Maske-2 dreht die Bits um, so daß Sie sowohl überprüfen können, ob eine Bedingung erfüllt ist oder nicht. Bits, für die ein 0-Test ausgeführt werden soll, müssen in der entsprechenden Position in Maske-2 eine 1 haben.

Sind die entsprechenden Bits der Operanden von Maske-1 und Maske-2 unterschiedlich, so ergibt sich durch die Verknüpfung mit ausschließendem ODER das Bit-Ergebnis 1. Ergibt sich bei den entsprechenden Bits das gleiche Ergebnis, so lautet das Bit 0. Über die WAIT-Anweisung kann eine unendliche Pause eingefügt werden. In diesem Fall ist eine Rückstellung mit den Tasten RUN/STOP und RESTORE möglich. Halten Sie die Taste RUN/STOP gedrückt und drücken Sie dann die Taste RESTORE . In nachstehendem ersten Beispiel wird so lange gewartet, bis an der Kassetteneinheit zur Fortsetzung des Programms eine Taste betätigt wird. Beim zweiten Beispiel wird gewartet, bis ein Sprite mit dem Bildschirmhintergrund kollidiert.

BEISPIELE DER WAIT-ANWEISUNG:

```
WAIT 1, 32, 32
WAIT 53273, 6, 6
WAIT 36868, 144, 16
```

(144 & 16 sind Masken. 144=10010000 binär und 16=10000 binär. Die WAIT-Anweisung stoppt das Programm, bis Bit 7 ein oder Bit 4 aus ist.)

TASTATUR UND MERKMALE DES COMMODORE 64

Das Betriebssystem hat einen Tastaturpuffer mit einer Kapazität von 10 Zeichen, der die über die Tastatur eingegebenen Befehle so lange speichert, bis sie ausgeführt werden können. Dieser Puffer speichert die Tastendrücke in der Reihenfolge, in der sie eingegeben wurden. D. h., die erste Eingabe wird auch zuerst ausgeführt. Folgt die zweite Tastenbetätigung z. B. ehe die erste ausgeführt werden kann, so wird die zweite im Puffer gespeichert, während die Ausführung des ersten Zeichens fortgeführt wird.

Nach Abarbeitung des ersten Zeichens wird überprüft, ob im Puffer weitere Daten sind. Dann wird die zweite Tastenbetätigung ausgeführt. Ohne diesen Puffer würden bei einer schnellen Eingabe über die Tastatur gelegentlich Zeichen verlorengehen.

Dies bedeutet also, daß der Puffer der Tastatur ein "Vorschreiben" ermöglicht und daß er Antworten auf INPUT-Rückfragen oder GET-Anweisungen unter Umständen vorwegnimmt. Bei Betätigung der Tastatur werden die entsprechenden Zeichen in einer Datei im Puffer aufgelistet und dann entsprechend der Eingangsreihenfolge ausgeführt. Dadurch kann es gelegentlich zu Störungen kommen, wenn durch eine versehentliche Tastenbetätigung das Programm aus dem Puffer ein falsches Zeichen empfängt.

Normalerweise stellen falsche Tastenbetätigungen keine Probleme dar, die durch die Taste ⇐CRSR oder die Löschtaste INST/DEL gelöscht und daher neu eingegeben werden können. Die Korrekturen werden vor dem nächsten "carriage-return" durchgeführt. Wird jedoch die Taste RETURN gedrückt, ist keine Korrektur möglich, da alle Zeichen im Puffer bis zu (und einschließlich) dem "carriage-return" vor einer weiteren Korrektur ausgeführt werden. Um dies zu vermeiden, kann eine Schleife benutzt werden, um den Tastaturpuffer vor dem Lesen zu löschen.

```
10 GET JUNK$: IF JUNK$ <>"" THEN 10: REM EMPTY THE KEYBOARD
BUFFER
```

Zusätzlich zu GET und INPUT kann die Tastatur auch über PEEK gelesen werden, indem aus dem Speicherplatz 197 ($00C5) der Wert der derzeitig gedrückten Taste gelesen wird. Wird bei der Ausführung von PEEK keine Taste gedrückt, so wird der Wert 64 wiedergegeben. Die numerischen Tastaturwerte, Tastensymbole und Zeichencodes (CHR$) werden in Anhang C gezeigt. Bei folgendem Beispiel wird solange eine Schleife durchgeführt, bis eine Taste gedrückt ist. Dann wird die ganze Zahl in ein Zeichen umgesetzt.

```
10 AA = PEEK(197): IF AA = 64 THEN 10
20 BB$ = CHR$(AA)
```

Die Tastatur ist eine Art Schaltersatz, der in eine Matrix von acht Spalten mal acht Reihen unterteilt ist. Die Tastaturmatrix wird über das CIA #1 Ein-/Ausgabechip (MOS 6526 Complex-Interface-Adapter) vom KERNAL hinsichtlich der Schalterstellungen abgetastet. Die Abtastung erfolgt über zwei CIA-Register. Register #0 bei Platz 56320 ($DC00) für die Spalten und Register #1 bei Platz 56321 ($DC01) für die Reihen.
Die Bits 0 bis 7 von Speicherplatz 56320 entsprechen den Spalten 0 bis 7. Die Bits 0 bis 7 von Speicherplatz 56321 entsprechen den Reihen 0 bis 7. Der KERNAL schreibt die Spaltenwerte nacheinander, liest dann die Reihenwerte und decodiert anschließend die Schalterstellung in den Wert CHR$ (N) der gedrückten Taste.
Aus acht Spalten mal acht Reihen ergeben sich 64 mögliche Werte. Wird jedoch zuerst die Taste RVS , CTRL , C= oder die Taste SHIFT gedrückt gehalten und ein weiterer Buchstabe eingegeben, so werden zusätzliche Werte erzeugt. Der KERNAL decodiert diese Tastaturen nämlich getrennt und "merkt sich", wenn eine Steuertaste gedrückt wurde. Das Ergebnis dieser Tastaturabfrage wird dann in Adresse 197 gespeichert.

Die Zeichen können auch direkt in den Tastaturpuffer über die POKE-Anweisung in die Speicherstellen 631–640 geschrieben werden. Diese Zeichen werden ausgeführt, wenn in Adresse 198 die Anzahl der Zeichen eingegeben wird. Auf diese Weise können Direkt-Modusbefehle automatisch durch Anzeige der Anweisungen auf dem Bildschirm, Eingabe von RETURNS in den Puffer und Einstellung des Zeichenzählers ausgeführt werden. Im nachstehenden Beispiel listet das Programm sich selbst auf dem Drucker auf und nimmt danach die Ausführung wieder auf.

```
10 PRINT CHR$(147)"PRINT#1: CLOSE 1: GOTO 50"
20 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3
30 OPEN 1,4: CMD1: LIST
40 END
50 REM PROGRAM RE-STARTS HERE
```

BILDSCHIRMEDITOR

Der Bildschirmeditor oder SCREEN EDITOR ist ein wirksames Hilfsmittel bei der Aufbereitung von Programmtexten. Nachdem ein Programmteil auf dem Bildschirm aufgelistet ist, kann man sich auf dem Bildschirm mit Hilfe der Cursorsteuertasten und weiterer Sondertasten frei bewegen, so daß die geeigneten Änderungen vorgenommen werden können. Wird nach Beendigung der Korrekturen einer bestimmten Textzeile die Taste RETURN an einer beliebigen Zeilenposition gedrückt, so liest der SCREEN EDITOR die gesamte logische Bildschirmzeile von 80 Zeichen ein.

Der Text wird dann zum Interpreter weitergegeben, gekennzeichnet und im Programm gespeichert. Hierbei wird eine alte Zeile im Speicher durch die aufbereitete ersetzt. Um eine zusätzliche Kopie von einer beliebigen Programmzeile zu erstellen, wird einfach die Zeilennummer geändert und die Taste RETURN gedrückt.

Überschreitet eine Programmzeile aufgrund der Verwendung von Schlüsselwortabkürzungen 80 Zeichen, so sind die überschüssigen Zeichen beim Aufbereiten der Zeile verloren, da der EDITOR nur zwei physische Bildschirmzeilen liest. Aus diesem Grund ist auch eine Eingabe von mehr als 80 Zeichen nicht möglich. Für praktische Anwendungen ist die Zeilenlänge eines BASIC-Textes daher entsprechend der Bildschirmanzeige auf 80 Zeichen begrenzt.

Unter bestimmten Bedingungen behandelt der SCREEN EDITOR die *Cursorsteuertasten* unterschiedlich zum normalen Modus. Steht der Cursor *rechts* neben einer ungeraden *Zahl* von Anführungszeichen ("), so arbeitet der Editor im QUOTE-MODUS (Anführungszeichen-Modus).

In diesem *Modus* werden Datenzeichen normal eingegeben, jedoch kann der Cursor nicht mehr über die Cursorsteuertasten bewegt werden. Durch Betätigung der Cursorsteuertasten werden statt dessen Zeichen in Negativdarstellung angezeigt. Das gleiche gilt für die Farbsteuertasten. Auf diese Weise können Sie Cursor und Farbsteuerung in Form von Strings in Ihr Programm aufnehmen. Sie werden noch feststellen, wie nützlich dies ist.
Wird nämlich ein Text, der zwischen Anführungszeichen steht, auf dem Bildschirm angezeigt, dann erfolgt automatisch die Cursorpositionierung und Farbsteuerung als Teil des Strings. Cursorsteuerung kann in Strings z. B. so benutzt werden:

```
Sie geben ein →          10 PRINT "A(R)(R)B(L)(L)(L)C(R)(R)D":
                         REM(R)=CRSR RIGHT, (L)=CRSR LEFT

Der Computer zeigt an →  AC BD
```

Die einzige Cursorsteuertaste, die NICHT vom Anführungszeichen-Modus beeinflußt wird, ist die Taste DEL . Erfolgt im Quote-Modus ein Fehler, kann nicht mit der Taste ⇐CRSR zurückgegangen und der Fehler überschrieben werden – selbst durch Anschlagen der Taste INST werden umgekehrte Bildschirmzeichen angezeigt.
Beenden Sie statt dessen die Eingabe der Zeile durch RETURN , dann können Sie diese normal abändern. Eine weitere Möglichkeit ist, die Tasten RUN/STOP und RESTORE zu drücken, wenn keine weiteren Cursorsteuerungen in der Zeichenkette benötigt werden. Hierdurch wird der Quote-Modus gelöscht. Die Cursorsteuertasten, die in Zeichenketten benutzt werden können, sind in Tabelle 2.2. gezeigt.

Tabelle 2.2. Cursorsteuertasten im QUOTE-MODUS

	Steuertaste	Bildschirmanzeige
Cursor nach oben	⇑ CRSR	●
Cursor nach unten	CRSR ⇓	Q
Cursor nach links	⇐CRSR	▌
Cursor nach rechts	CRSR⇒	]
Löschen	CLR/	♥
Ausgangsstellung	/HOME	S
Einfügen	INST/DEL	▌

Wenn Sie sich NICHT im Quote-Modus befinden, werden durch gleichzeitiges Drücken der Tasten SHIFT und INST die Daten rechts neben dem Cursor verschoben. So entsteht zwischen zwei Zeichen Platz für die Eingabe weiterer Zeichen. Der Editor arbeitet nun solange im *Einfügemodus,* bis alle geöffneten Leerstellen gefüllt sind.
Auch im Einfügemodus erscheinen nach Betätigung der Cursor- und Farbsteuertasten umgekehrte Zeichen. Der einzige Unterschied zeigt sich beim Drücken der Taste INST/DEL . Durch DEL wird nun ein umgekehrtes T angezeigt. Die Taste INST, die im Anführungszeichen-Modus Revers-Zeichen anzeigt, fügt Leerzeichen ein.
Dies bedeutet, daß in einer PRINT-Anweisung Löschungen (DEL) im Gegensatz zum Quote-Modus möglich sind. Der Einfügemodus wird durch Drücken der Tasten RETURN, SHIFT und RETURN oder RUN/STOP und RESTORE gelöscht. Außerdem wird dieser Modus gelöscht, wenn alle eingefügten Leerstellen gefüllt sind. DEL-Zeichen können in Zeichenketten z. B. so benutzt werden:

10 PRINT "HELLO" DEL INST INST DEL DEL P"
(Die obige Tastenfolge erscheint bei der Auflistung wie folgt:)
10 PRINT"HELP"

Wird nach diesem Beispiel RUN eingegeben, so wird das Wort HELP angezeigt. Die Buchstaben LO werden nämlich vor der Anzeige von P gelöscht. Die Löschzeichen in Zeichenketten gelten sowohl für die Anweisung LIST als auch für PRINT. Auf diese Weise können Sie alle oder einen Teil der Textzeilen "verstecken". Die Abänderung einer Zeile mit diesen Zeichen ist jedoch schwierig.
Es gibt noch weitere Zeichen, die für spezielle Funktionen angezeigt werden können, auch wenn diese nicht ganz einfach über die Tastatur zur Verfügung stehen. Um diese Zeichen in Anführungszeichen zu setzen, werden in der Zeile entsprechende Leerzeichen gelassen, die Taste RETURN gedrückt und dann zur Zeilenaufbereitung zurückgegangen.
Drücken Sie nun die Tasten CTRL und RVS/ON, um mit der Anzeige der unterlegten Zeichen zu beginnen. Schlagen Sie die Tasten wie folgt an:

Tastenfunktion	Tastenanschlag	Bildschirmanzeige
Großumschaltung RETURN	SHIFT M	
Umschaltung auf Klein-/Großschrift	N	N
Umschaltung auf Großbuchstaben/ Graphikzeichen	SHIFT N	

Das gleichzeitige Drücken der Tasten SHIFT und RETURN verursacht auf dem Bildschirm einen "Wagenrücklauf" und einen Zeilenvorschub, aber die Zeichenkette wird nicht beendet. Dies gilt sowohl für LIST als auch für PRINT, so daß bei Verwendung dieser Zeichen eine Abänderung schwierig ist. Wird für die Ausgabe über die CMD-Anweisung der Drucker gewählt, so wird durch das unterlegte Zeichen "N" der Zeichensatz für Klein-/Großschrift und durch SHIFT "N" der Zeichensatz für Großbuchstaben/Graphikzeichen eingeschaltet.
Durch gleichzeitiges Drücken der Tasten CTRL und RVS können unterlegte Bildschirmzeichen in Strings eingeschlossen werden. Auf dem Bildschirm erscheint dann in Anführungszeichen ein unterlegtes R. Auf diese Weise werden alle Zeichen auf dem Bildschirm unterlegt (d. h. wie ein Negativbild) angezeigt. Um diese Ausgabe zu beenden, drücken Sie gleichzeitig die Tasten CTRL und RVS/OFF. Nun wird ein unterlegtes Graphikzeichen angezeigt. Numerische Daten können unterlegt angezeigt werden, indem man zunächst ein CHR$(18) eingibt. Durch CHR$(146) oder ein "Carriage return" wird diese umgekehrte Bildschirmausgabe gelöscht.

KAPITEL 3

GRAPHIK-PROGRAMMIERUNG MIT DEM COMMODORE 64

- Graphikübersicht
- Lage der Graphikzeichen
- Standardzeichenmodus
- Programmierbare Zeichen
- Mehrfarbengraphiken
- Erweiterter Hintergrundfarbmodus
- "Bit-Mapped"-Graphiken
- Mehrfarben-"Bit-Map-Modus"
- Rollen der Bildschirmanzeige
- Sprites
- Weitere Graphikmöglichkeiten
- Programmieren von Sprites – ein anderer Aspekt

GRAPHIKÜBERSICHT

Sämtliche Graphikmöglichkeiten des COMMODORE 64 basieren auf dem Video-Interface-Chip 6567 (auch bekannt als VIC-II-Chip). Dieser Chip ermöglicht die verschiedensten Graphikarten, einschließlich einer Textdarstellung von 40 Zeichen mal 25 Zeilen, einem hoch auflösenden Display von 320 mal 200 Punkten sowie den SPRITES, kleinen beweglichen Objekten, die das Erstellen von Spielen wesentlich vereinfachen. Darüber hinaus können viele dieser Graphikarten auf dem gleichen Display gemischt werden. So ist es z. B. möglich, für die obere Bildschirmhälfte den Modus mit hoher Auflösung und für die untere Bildschirmhälfte den Textmodus zu wählen. SPRITES lassen sich mit allen Displayarten kombinieren. Wir werden später noch auf Sprites genauer eingehen. Zunächst beschäftigen wir uns mit den übrigen Graphikarten.
Mit dem VIC-II-Chip sind folgende Graphikarten möglich:

A) ZEICHENANZEIGE

1) Standardzeichen
- a) ROM-Zeichen
- b) RAM-programmierbare Zeichen

2) Mehrfarbige Zeichen
- a) ROM-Zeichen
- b) RAM-programmierbare Zeichen

3) Erweiterte Hintergrundfarbe
- a) ROM-Zeichen
- b) RAM-programmierbare Zeichen

B) BIT-MAP-MODUS

1) Standard-Bit-Map-Modus
2) Mehrfarben-Bit-Map-Modus

C) SPRITES

1) Standard-Sprites
2) Mehrfarben-Sprites

LAGE DER GRAPHIKZEICHEN

Zunächst einige allgemeine Informationen. Der Bildschirm des COMMODORE 64 verfügt über 1000 Positionen. Normalerweise beginnt der Bildschirm bei Adresse 1024 ($0400 in hexadezimaler Darstellung) und geht bis zu Adresse 2023. Jede dieser Adressen kann 8 Bits speichern, das entspricht einer beliebigen ganzen Zahl zwischen 0 und 255. Dem Bildschirmspeicher entspricht eine Gruppe von 1000 Adressen, die **FARBSPEICHER** oder **FARB-RAM** genannt wird. Diese beginnen bei Platz 55296 ($D800 in hexadezimaler Darstellung) und reichen bis zu 56295. Jede dieser Farb-RAM-Adressen speichert 4 Bits und kann daher eine beliebige ganze Zahl von 0 bis 15 aufnehmen. Da der COMMODORE 64 über 16 mögliche Farben verfügt, kann man hiermit sehr gut arbeiten.
Darüber hinaus können jederzeit 256 verschiedene Zeichen angezeigt werden. Bei der normalen Bildschirmanzeige enthält jede der 1000 Adressen des Bildschirmspeichers eine Code-Zahl, die dem VIC-II-Chip "sagt", welches Zeichen an diesem Bildschirmplatz anzuzeigen ist.
Die verschiedenen Graphikmodi werden über die 47 **Steuer**register im VIC-II-Chip gewählt. Viele Graphikfunktionen lassen sich steuern, indem der richtige Wert über die POKE-Anweisung in eines der Register geschrieben wird. Der VIC-II-Chip befindet sich an den Speicherplätzen 53248 ($D000 in Hexadezimaldarstellung) bis 53294 ($D02E).

WAHL DER VIDEO-BANK

Der VIC-II-Chip kann gleichzeitig auf einen Speicherbereich von 16K zugreifen. Da der COMMODORE 64 über einen 64K-Speicher verfügt, soll der VIC-II natürlich auch den ganzen Speicher "sehen" können. Dies ist möglich. Es gibt vier verschiedene **BANKS** (oder Abschnitte), die für jeweils 16K gelten. Nun muß lediglich noch geregelt werden, auf welche dieser Abschnitte der VIC-II-Chip zugreift. Auf diese Weise kann der Chip die gesamte Speicherkapazität von 64K "sehen". Die **Bankanwahl**-Bits, die Ihnen einen Zugriff auf die verschiedenen Speicherabschnitte ermöglichen, befinden sich im **COMPLEX-INTERFACE ADAPTER-CHIP #2 (CIA #2) 6526.** Über die BASIC-Anweisungen POKE und PEEK (oder die entsprechenden Versionen in der Maschinensprache) wird eine Bank durch Steuerung der Bits 0 und 1 von PORT A des CIA#2 (Platz 56576 (oder $DD00 in Hexadezimaldarstellung)) gewählt. Zur Änderung der Speicherabschnitte müssen diese zwei Bits auf *Ausgabe* gesetzt sein. Dies wird anhand nachstehenden Beispiels deutlich:

```
POKE 56578,PEEK(56578)OR 3 :REM BITS 0 UND 1 ALS AUSGANG
SETZEN
POKE 56576,(PEEK(56576)AND 252)OR A:REM VIDEO-BANK WECHSELN
```

"A" muß einen der folgenden Werte haben:

WERT VON A	BITS	BANK	START-PLATZ	BEREICH DES VIC-II-CHIP
0	00	3	49152	($C000–$FFFF)*
1	01	2	32768	($8000–$BFFF)
2	10	1	16384	($4000–$7FFF)*
3	11	0	0	($0000–$3FFF) (STANDARDWERT)

Dieses Konzept der 16K-Abschnitte spielt bei allen Anwendungen des VIC-II-Chip eine Rolle. Sie sollten stets wissen, auf welche Bank VIC-II zeigt, da dies beeinflußt, von wo die Zeichendatenmuster kommen, wo sich der Bildschirm befindet, von wo die Sprites kommen usw. Nach dem Einschalten des COMMODORE 64 gelten für die Bits 0 und 1 von Platz 56576 automatisch BANK 0 ($0000–$3FFF) für sämtliche Anzeigeninformationen.

***Anmerkung:** Der Zeichensatz des COMMODORE 64 ist in den Banks 1 und 3 für den VIC-II-Chip nicht verfügbar. (Siehe Abschnitt "Zeichenspeicher".)

BILDSCHIRMSPEICHER

Durch POKEn in das Kontrollregister 53272 ($D018 HEX) kann die Adresse des Bildschirmspeichers geändert werden. Dieses Register wird jedoch auch zur Steuerung des jeweils benutzten Zeichensatzes verwendet. Achten Sie daher besonders darauf, diesen Teil des Steuerregisters nicht zu stören. Die **oberen 4** Bits steuern den Platz des Bildschirmspeichers. Zur Bewegung des Bildschirms ist folgende Anweisung erforderlich:

```
POKE53272,(PEEK(53272)AND15)ORA
```

Hierbei hat A einen der folgenden Werte:

A	BITS	LAGE*	
		DEZIMAL	HEXADEZIMAL
0	0000XXXX	0	$0000
16	0001XXXX	1024	$0400 (STANDARD)
32	0010XXXX	2048	$0800
48	0011XXXX	3072	$0C00
64	0100XXXX	4096	$1000
80	0101XXXX	5120	$1400
96	0110XXXX	6144	$1800
112	0111XXXX	7168	$1C00
128	1000XXXX	8192	$2000
144	1001XXXX	9216	$2400
160	1010XXXX	10240	$2800
176	1011XXXX	11264	$2C00
192	1100XXXX	12288	$3000
208	1101XXXX	13312	$3400
224	1110XXXX	14336	$3800
240	1111XXXX	15360	$3C00

***Bitte denken** Sie daran, daß die Startadresse der jeweiligen Bank des VIC-II-Chip addiert werden muß.

FARBSPEICHER

Der Farbspeicher kann nicht verschoben werden. Er befindet sich stets an den Plätzen 55296 ($D800) bis 56295 ($DBE7). Bildschirmspeicher (1000 Plätze beginnend bei 1024) und Farbspeicher werden in den verschiedenen Graphikmodi unterschiedlich benutzt. Ein in einem Modus erstelltes Bild sieht in einem anderen Graphikmodus häufig völlig anders aus.

ZEICHENSPEICHER

Für die Programmierung von Graphiken ist es wesentlich, von wo genau der VIC-II die Zeicheninformation bekommt. Normalerweise erhält der Chip die Konturen der anzuzeigenden Zeichen vom **Character-Generator-ROM.** In diesem Chip werden die *Muster* gespeichert, die die verschiedenen Buchstaben, Zahlen, Interpunktionssymbole und alle anderen Zeichen der Tastatur bilden.

Eines der Merkmale des COMMODORE 64 ist seine Fähigkeit, im RAM-Speicher befindliche Muster zu benutzen. Diese RAM-Muster werden von Ihnen erstellt, so daß Ihnen ein nahezu unbegrenzter Satz an Symbolen für Spiele, Geschäftsanwendungen usw. zur Verfügung steht.
Ein normaler Zeichensatz enthält 256 Zeichen, bei dem jedes Zeichen durch 8 Bytes bestimmt wird. Da jedes Zeichen also 8 Bytes beansprucht, benötigt der komplette Zeichensatz 256*8=2K-Bytes. Da der VIC-II-Chip gleichzeitig auf 16K zugreift, gibt es acht verschiedene Speicherplatzmöglichkeiten für einen vollständigen Zeichensatz. Sie brauchen natürlich nicht immer einen ganzen Zeichensatz zu verwenden. Er muß jedoch stets an einem der acht möglichen Startplätze beginnen.
Die Lage des Zeichenspeichers wird durch 3 Bits vom VIC-II-Speicherregister an Platz 53272 ($D018 HEX) kontrolliert. Die Bits 3, 2 und 1 steuern, wo sich der Zeichensatz in 2K-Sätzen befindet. Bit 0 wird überlesen. Bitte denken Sie daran, daß dies das gleiche Register ist, das auch die Lage des Bildschirmspeichers bestimmt. Um die Lage vom Zeichenspeicher zu ändern, benutzen Sie folgende BASIC-Anweisung:

```
POKE 53272,(PEEK(53272)AND240)OR A
```

Hierbei hat A einen der folgenden Werte:

WERT VON A	BITS	LAGE DES ZEICHENSPEICHERS*		
		DEZIMAL	HEXADEZIMAL	
0	XXXX000X	0	$0000–$07FF	
2	XXXX001X	2048	$0800–$0FFF	
4	XXXX010X	4096	$1000–$17FF	ROM-IMAGE in BANK 0 & 2 (Standard)
6	XXXX011X	6144	$1800–$1FFF	ROM-IMAGE in BANK 0 & 2
8	XXXX100X	8192	$2000–$27FF	
10	XXXX101X	10240	$2800–$2FFF	
12	XXXX110X	12288	$3000–$37FF	
14	XXXX111X	14336	$3800–$3FFF	

***Bitte denken Sie daran, die Startadresse der Bank zu addieren.**

Das **ROM-IMAGE** in obiger Tabelle bezieht sich auf das Character-Generator-ROM. Es erscheint im RAM bei obigen Plätzen in Bank 0. Außerdem erscheint es im entsprechenden RAM an den Plätzen 36864–40959 ($9000–$9FFF) in Bank 2. Da der VIC-II-Chip gleichzeitig nur auf 16K zugreifen kann, erscheinen die ROM-Zeichenmuster in dem Satz, auf den gerade zugegriffen wird. Aus diesem Grund ist das System so entwickelt, daß VIC-II davon ausgeht, daß sich die ROM-Zeichen bei 4096–8191 ($1000–$1FFF) befinden, wenn Ihre Daten in Bank 0 sind, und bei 36864–40959 ($9000–$9FFF) im Fall von Bank 2. Die ROM-Zeichen befinden sich jedoch tatsächlich an den Plätzen 53248–57343 ($D000–$DFFF).
Diese "Spiegelung" bezieht sich nur auf Zeichendaten, wie sie vom VIC-II-Chip "gesehen" werden. RAM an diesen Adressen kann wie jeder andere RAM-Speicher für Programme, andere Daten usw. benutzt werden.

Anmerkung: Wenn diese ROM-Spiegelungen Ihre eigenen Graphiken behindern, wählen Sie mit den **BANKANWAHL-BITS** eine der Banks ohne Belegung (Bank 1 oder 3). Die ROM-Muster tauchen dort nicht auf.

ADRESSE			VIC-II-SPIEGE-LUNG	INHALT
BLOCK	DEZIMAL	HEX		
0	53248	D000–D1FF	1000–11FF	Großbuchstaben
	53760	D200–D3FF	1200–13FF	Graphikzeichen
	54272	D400–D5FF	1400–15FF	Großbuchstaben in Reversdarstellung
	54784	D600–D7FF	1600–17FF	Graphikzeichen in Reversdarstellung
1	55296	D800–D9FF	1800–19FF	Kleinbuchstaben
	55808	DA00–DBFF	1A00–1BFF	Großbuchstaben und Graphikzeichen
	56320	DC00–DDFF	1C00–1DFF	Kleinbuchstaben in Reversdarstellung
	56832	DE00–DFFF	1E00–1FFF	Großbuchstaben in Reversdarstellung

Dem aufmerksamen Leser wird jetzt aufgefallen sein, daß die vom Zeichen-ROM beanspruchten Plätze die gleichen sind wie die der VIC-II-Chip-Steuerregister. Dies ist möglich, da die Plätze nicht gleichzeitig beansprucht werden.

Benötigt der VIC-II-Chip den Zugriff auf die Zeichendaten, so wird das ROM eingeschaltet. In der 16K-Speicherbank, auf die der VIC-II-Chip zugreift, entsteht die entsprechende "Spiegelung". Ansonsten wird dieser Bereich von den Ein-/Ausgaberegistern beansprucht und das Zeichen-ROM kann nur vom VIC-II erreicht werden.
Es kann jedoch passieren, daß Sie das Zeichen-ROM benötigen, und zwar dann, wenn Sie programmierbare Zeichen benutzen wollen und eine Kopie eines Teils vom Zeichen-ROM für die Zeichendefinition benötigen. In diesem Fall müssen Sie das Ein-/Ausgaberegister aus- und das Zeichen-ROM einschalten. Dann können Sie kopieren. Danach muß das Ein-/Ausgaberegister erneut eingeschaltet werden. Während des Kopierens (bei ausgeschalteter Ein-/Ausgabe) sind keine Unterbrechungen erlaubt. Für Unterbrechungen werden nämlich die Ein-/Ausgaberegister benötigt. Wenn Sie dies vergessen und eine Unterbrechung vornehmen, passiert Unvorhersehbares. Die Tasteneingabe darf während des Kopierens nicht gelesen werden. Um die Tastatur und weitere normale Unterbrechungen abzuschalten, die mit dem COMMODORE 64 möglich sind, benutzen Sie folgende POKE-Anweisung:

```
POKE 56334,PEEK(56334)AND254    (Interrupt AUS)
```

Wenn Sie den Zugriff auf den Zeichen-ROM beendet haben und bereit für die Programmfortsetzung sind, wird die Tastatur durch folgende POKE-Anweisung wieder eingeschaltet:

```
POKE 56334,PEEK(56334)OR1        (Interrupt EIN)
```

Durch folgende POKE-Anweisung wird die Ein-/Ausgabe ausgeschaltet und der Zeichen-ROM eingeschaltet:

```
POKE 1,PEEK(1)AND251
```

Der Zeichen-ROM befindet sich nun an den Plätzen 53248 bis 57343 ($D000–$DFFF). Um die Ein-/Ausgabe für den normalen Betrieb zurück in $D000 zu schalten, benutzen Sie folgende POKE-Anweisung:

```
POKE 1,PEEK(1)OR 4
```

STANDARDZEICHENMODUS

Beim Einschalten des COMMODORE 64 befindet sich dieser im Standardzeichenmodus. Dies ist der Modus, in dem Sie normalerweise Programmierungen vornehmen.
Zeichen können aus dem ROM oder dem RAM gelesen werden. Normalerweise wird jedoch auf die Zeichen im ROM zugegriffen. Benötigen Sie für ein Programm spezielle Graphikzeichen, so brauchen Sie lediglich die neuen Zeichenmuster im RAM zu definieren und den VIC-II-Chip anzuweisen, die Zeicheninformationen von da und nicht aus dem Zeichen-ROM zu nehmen. Dies wird im nachstehenden Abschnitt noch genauer beschrieben.
Um Zeichen auf dem Bildschirm in Farbe anzuzeigen, greift der VIC-II-Chip auf den Bildschirmspeicher zu, um den Zeichen-Code für diesen Bildschirmplatz zu bestimmen. Gleichzeitig greift er auf den Farbspeicher zu, um die Farbe für die Zeichenanzeige festzulegen. Der Zeichen-Code wird vom VIC-II in die Startadresse des 8-Byte-Satzes mit Ihrem Zeichenmuster umgesetzt. Dieser Satz befindet sich im Zeichenspeicher.
Die Umsetzung ist nicht zu kompliziert, zur Erstellung der gewünschten Adresse werden jedoch verschiedene Punkte kombiniert. Zunächst wird der von Ihnen bei der POKE-Anweisung für den Bildschirmspeicher benutzte Zeichencode mit 8 multipliziert. Danach wird der Anfang vom Zeichenspeicher addiert (siehe Abschnitt "Zeichenspeicher"). Nun werden die Bankanwahl-Bits berücksichtigt. Hierzu wird die Basisadresse (siehe Abschnitt "Video-Bankwahl") addiert. Anhand der folgenden einfachen Gleichung können Sie sehen, wie dies gemeint ist:

CHARACTER ADDRESS = SCREEN CODE*8+(CHARACTER SET*2048)+(BANK*16384)

ZEICHENDEFINITIONEN

Jedes Zeichen wird aus einer Matrix von 8 mal 8 Punkten gebildet. Hierbei können die einzelnen Punkte entweder ein- oder ausgeschaltet sein. Beim COMMODORE 64 sind die Zeichenbilder im Zeichengenerator-ROM abgelegt. Jedes Zeichen ist hierbei als Satz von 8 Bytes gespeichert. Jedes Byte steht für das Punktmuster einer Reihe im Zeichen und jedes Bit für einen Punkt. Ein 0-Bit zeigt an, daß der Punkt ausgeschaltet, und ein 1-Bit, daß er eingeschaltet ist.
Der Zeichenspeicher im ROM beginnt bei Platz 53248 (bei ausgeschalteter Ein-/Ausgabe). Die ersten 8 Bytes von Platz 53248 ($D000) bis 53255 ($D007) enthalten das Muster für das Zeichen @, dessen Zeichencodewert im Bildschirmspeicher 0 ist.

Die nächsten 8 Bytes von Platz 53256 ($D008) bis 53263 ($D00F) enthalten die Information zur Bildung des Buchstabens A.

BELEGUNG	BINÄR	PEEK
**	00011000	24
****	00111100	60
** **	01100110	102
******	01111110	126
** **	01100110	102
** **	01100110	102
** **	01100110	102
	00000000	0

Jeder vollständige Zeichensatz beansprucht eine Speicherkapazität von 2K (2048 Bits). Insgesamt sind 256 Zeichen enthalten, wobei jedes Zeichen 8 Bytes umfaßt. Da es insgesamt zwei Zeichensätze gibt, und zwar einen für die Großbuchstaben und Graphikzeichen und den anderen für Groß- und Kleinbuchstaben, enthält der ROM-Zeichenspeicher insgesamt 4K Speicherplätze.

PROGRAMMIERBARE ZEICHEN

Da die Zeichen im ROM gespeichert sind, sieht es so aus, als ob sie für frei programmierbare Zeichen nicht geändert werden könnten. Der Speicherplatz, der dem VIC-II-Chip mitteilt, wo die Zeichen zu finden sind, ist jedoch ein programmierbares Register. Dieses kann so geändert werden, daß es auf viele Speicherbereiche zeigt. Indem der Zeichenspeicherzeiger so geändert wird, daß er auf den RAM zeigt, kann der Zeichensatz beliebig programmiert werden.
Soll sich Ihr Zeichensatz im RAM befinden, so gibt es einige **SEHR WICHTIGE** Dinge, die Sie dabei berücksichtigen müssen. Darüber hinaus sind zwei weitere wichtige Aspekte bei der Erstellung Ihrer eigenen Sonderzeichen zu beachten:

1) Dies ist ein Alles-oder-Nichts-Vorgang. Im allgemeinen, wenn Sie den VIC-II-Chip angewiesen haben, die Zeicheninformationen aus dem vorbereiteten RAM-Bereich zu nehmen, sind die Standardzeichen vom COMMODORE 64 für Sie nicht verfügbar. Um dieses Problem zu lösen, müssen Sie alle Buchstaben, Zahlen oder Standardgraphikzeichen vom Commodore 64 in den RAM-Speicher kopieren, den Sie dann in Ihrem Programm benutzen wollen. Hierbei können Sie beliebige Zeichen auswählen und brauchen auch nicht auf die Reihenfolge zu achten!

2) Ihr Zeichensatz benutzt denselben Speicher wie das BASIC-Programm. Da für das BASIC-Programm jedoch 38K zur Verfügung stehen, ist dies meist problemlos.

Achtung: Achten Sie darauf, daß Ihr Zeichensatz nicht vom BASIC-Programm, das auch den RAM benutzt, überschrieben wird.

Zwei Adressen im COMMODORE 64 **dürfen nicht als Beginn des Zeichensatzes gewählt werden: Adresse 0 und Adresse 2048.** Der erste darf nicht benutzt werden, da das System auf Seite 0 (0-Page) wichtige Daten speichert. Adresse 2048 ist der Beginn Ihres BASIC-Programms!
Für Ihren Zeichensatz stehen jedoch noch sechs weitere Anfangspositionen zur Verfügung.
Am besten wählen Sie hierzu am Anfang Platz 12288 ($3000 in Hexadezimaldarstellung). Dies erfolgt, indem die unteren 4 Bits von Platz 53272 mit 12 gePOKEt werden. Probieren Sie nun folgende POKE-Anweisung aus:

```
POKE 53272,(PEEK(53272)AND240)+12
```

Sofort sind alle Buchstaben vom Bildschirm verschwunden. Der Grund hierfür liegt darin, daß bis jetzt noch kein Zeichensatz ab Adresse 12288 steht . . . nur zufällige Bytes. Kehren Sie mit dem COMMODORE 64 durch Betätigen der Tasten RUN/STOP und RESTORE wieder zurück in den Normalmodus.
Nun wollen wir Graphikzeichen erstellen. Um Ihren Zeichensatz zu schützen, sollten Sie die Speicherkapazität für BASIC reduzieren. Der Speicher in Ihrem Computer bleibt unverändert . . . Sie haben lediglich BASIC die Anweisung gegeben, einen bestimmten Teil nicht zu benutzen. Tippen Sie folgendes ein:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Die angezeigte Zahl gibt die unbenutzte Speicherkapazität an. Geben Sie nun folgendes ein:

```
POKE 52,48:POKE56,48:CLR
```

Und nun:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Sehen Sie die Änderung? BASIC nimmt nun an, daß weniger Speicherkapazität zur Verfügung steht. In diesen gewonnenen Speicherplatz können Sie nun Ihren Zeichensatz eingeben.
Als nächstes müssen nun Ihre Zeichen in den RAM eingegeben werden. Zu Beginn stehen ab 12288 ($3000 in HEX) zufällige Daten. Sie müssen das Zeichenmuster in den RAM eingeben (auf die gleiche Art wie sie im ROM gespeichert sind), damit der VIC-II-Chip sie benutzen kann. Durch folgendes Programm werden 64 Zeichen vom ROM in den RAM-Zeichensatz übertragen:

```
5 PRINTCHR$(142)                    :REM SWITCH TO
UPPER CASE
10 POKE52,48:POKE56,48:CLR          :REM RESERVE MEMORY
FOR CHARACTERS
20 POKE56334,PEEK(56334)AND254      :REM TURN OFF
KEYSCAN INTERRUPT TIMER
30 POKE1,PEEK(1)AND251              :REM SWITCH IN
CHARACTER
40 FORI=0TO511:POKEI+12288,PEEK(I+53248):NEXT
50 POKE1,PEEK(1)OR4                 :REM SWITCH IN I/O
60 POKE56334,PEEK(56334)OR1         :REM RESTART
KEYSCAN INTERRUPT TIMER
70 END
```

POKEn Sie in die Adresse 53272 den Wert (PEEK(53272)AND240)+12. Nichts passiert, stimmt's? Fast nichts! Der COMMODORE 64 bekommt die Zeicheninformationen nun vom RAM und nicht vom ROM. Da wir jedoch die Zeichen genau vom ROM kopiert haben, ist kein Unterschied zu sehen . . . noch nicht.
Die Zeichen können nun leicht geändert werden. Löschen Sie den Bildschirm, und drücken Sie die Taste @. Bewegen Sie den Cursor um einige Zeilen nach unten, und geben Sie dann folgendes ein:

```
FOR I = 12288 TO 12288+7:POKE I, 255 - PEEK(I) : NEXT
```

Sie haben soeben ein @ in Reversdarstellung erstellt!

Hinweis: Negativ dargestellte Zeichen gehen aus den Zeichen durch Umkehrung der Bit-Muster im Zeichenspeicher hervor.

Bewegen Sie nun den Cursor wieder zum Programmanfang, und drücken Sie die Taste RETURN erneut, um das Zeichen noch einmal umzukehren (d. h., es wird wieder normal). Die Tabelle der Bildschirm-Codes zeigt Ihnen, wo die einzelnen Zeichen im RAM sind. Denken Sie daran, daß zur Speicherung jedes Zeichens 8 Speicherplätze benötigt werden. Hier ein paar Beispiele:

ZEICHEN	BILDSCHIRM-CODE	DERZEITIGE STARTADRESSE IM RAM
@	0	12288
A	1	12296
!	33	12552
>	62	12784

Denken Sie daran, daß wir nur die ersten 64 Zeichen genommen haben. Wird eines der anderen Zeichen gewünscht, so ist vorher noch etwas zu berücksichtigen.
Was ist zu tun, wenn Sie nun Zeichennummer 154, ein umgekehrtes Z wünschen? Sie können das erreichen, indem Sie ein Z umkehren, oder Sie können den Satz der umgekehrten Zeichen vom ROM kopieren oder einfach das eine Zeichen aus dem ROM holen und ein nicht benötigtes Zeichen im RAM dadurch ersetzen.
Nehmen wir an, Sie benötigen das Zeichen > nicht mehr. Dieses Zeichen soll also gegen das negativ dargestellte Z ausgetauscht werden. Geben Sie folgendes ein:

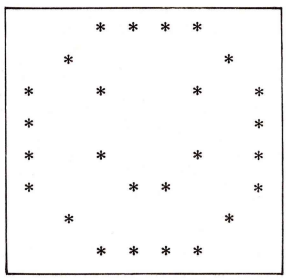
```
FOR I=0 TO 7: POKE 12784 + I, 255-PEEK(I+12496): NEXT
```

Geben Sie nun ein > ein. Es erscheint als umgekehrtes Z. So oft Sie nun dieses auch eingeben, erscheint es immer als umgekehrtes Z. (Diese Änderung betrifft jedoch nur die Darstellung auf dem Bildschirm. Auch wenn das Zeichen wie ein umgekehrtes Z aussieht, wirkt es in einem Programm doch immer noch als >.)
Probieren Sie das an einem Beispiel aus, bei dem dieses Zeichen benötigt wird.
Fassen wir zusammen: Sie können nun Zeichen vom ROM in das RAM kopieren. Sie können hierbei selbst die Zeichen auswählen. Hinsichtlich der programmierbaren Zeichen fehlt Ihnen also nur noch ein Punkt (und zwar der beste!) . . . das Erstellen Ihrer eigenen Zeichen.
Wissen Sie noch, wie Zeichen im ROM gespeichert sind? Jedes Zeichen wird als Gruppe von 8 Bytes gespeichert. Die Bit-Muster der Bytes geben direkt das Zeichen wieder. Werden 8 Bytes übereinander angeordnet und jedes Byte als achtstellige Binärzahl geschrieben, so entsteht eine 8-mal-8-Matrix, die wie die Zeichen aussieht. Ist ein Bit eine 1, so ist an diesem Platz ein Punkt. Ist ein Bit eine 0, ist an diesem Platz eine Leerstelle.
Zum Erstellen Ihrer eigenen Zeichen geben Sie in den Speicher die gleiche Bitanordnung ein. Geben Sie NEW und danach dieses Programm ein:

```
10 FOR I = 12448 TO 12455 : READ A: POKE I,A: NEXT
20 DATA 60, 66, 165, 129, 165, 153, 66, 60
```

Geben Sie nun RUN ein. Das Programm ersetzt den Buchstaben T durch ein "Gesicht". Um das Gesicht zu sehen, geben Sie mehrere T's ein. Jede Zahl in der DATA-Anweisung in Zeile 20 ist eine Reihe in diesem Gesicht. Es gilt folgende Matrix:

	7 6 5 4 3 2 1 0	BINÄR	DEZIMAL
Reihe 0		00111100	60
1		01000010	66
2		10100101	165
3		10000001	129
4		10100101	165
5		10011001	153
6		01000010	66
Reihe 7		00111100	60



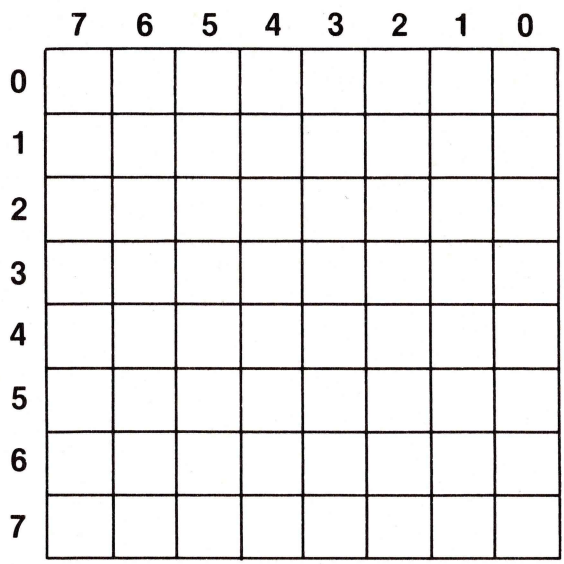


Abb. 3.1. Arbeitsblatt für programmierbare Zeichen

Das Arbeitsblatt für programmierbare Zeichen (Abb. 3.1.) hilft Ihnen beim Entwurf Ihrer eigenen Zeichen. Das Blatt enthält eine Matrix von 8 mal 8 mit Reihennummern sowie Nummern oben über jeder Spalte. (Wird jede Reihe als Binärwort gesehen, so sind die Spaltennummern die jeweiligen Werte der Bit-Position. Der Wert läßt sich einfach als Zweierpotenz errechnen. Das linke äußerste Bit entspricht 128 oder 2^7, das nächste 64 oder 2^6 usw., bis das äußerste rechte Bit (Bit 0) erreicht ist. Bit 0 entspricht 1 oder 2^0.)

Geben Sie in die Matrix überall da ein X ein, wo in Ihrem Zeichen ein Punkt erscheinen soll. Ist das Zeichen fertig, dann können Sie die DATA-Anweisung dafür erstellen.

Beginnen Sie mit der ersten Reihe. Überall da, wo ein X eingesetzt ist, lesen Sie die Nummer oben an der Spalte ab (die Zweierpotenz) und notieren Sie sie. Dann werden die Zweierpotenzen der ersten Reihe addiert. Notieren Sie diese Summe neben der Reihe. Sie wird später in der DATA-Anweisung benutzt, um diese Reihe als Bitmuster wiederzugeben.

Das gleiche gilt für die übrigen Reihen (1–7). Sie müssen dann insgesamt 8 Zahlen zwischen 0 und 255 haben. Liegt eine dieser Zahlen nicht innerhalb dieses Bereiches, überprüfen Sie die Addition. Bei richtiger Addition müssen die Zahlen auf jeden Fall in diesem Bereich liegen. Haben Sie weniger als 8 Zahlen, dann haben Sie wahrscheinlich eine Reihe vergessen. Es ist durchaus korrekt, wenn auch Nullen dabei sind. Diese 0-Reihen sind genauso wichtig wie die anderen Zahlen.

Ersetzen Sie die Zahlen in der DATA-Anweisung in Zeile 20 durch die soeben berechneten Zahlen und geben Sie danach RUN ein. Drücken Sie nun die Taste T. Bei jedem Betätigen dieser Taste sehen Sie Ihr eigenes Zeichen!

Wenn Ihnen dieses Zeichen noch nicht gefällt, ändern Sie einfach die Zahlen in der DATA-Anweisung, bis die Zeichendarstellung zufriedenstellend ist. Das ist alles!

Hinweis: Die vertikalen Linien in Ihren Zeichen sollten stets mindestens zwei Punkte (Bits) breit sein. Hierdurch werden bei der Anzeige auf dem Bildschirm Farbfehler in den Zeichen vermieden.

Nachfolgend sehen Sie ein Programmbeispiel mit den programmierbaren Standardzeichen:

```
10 REM * EXAMPLE 1 *
20 REM CREATING PROGRAMMABLE CHARACTERS
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251:
REM TURN OFF KB AND I/O
35 FORI=0TO63:REM CHARACTER RANGE TO BE COPIED
FROM ROM
36 FORJ=0TO7:REM COPY ALL 8 BYTES PER CHARACTER
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPY A
BYTE
38 NEXTJ:NEXTI:REM GOTO NEXT BYTE OR CHARACTER
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM
TURN ON I/O AND KB
40 POKE53272,(PEEK(53272)AND240)+12:REM SET CHAR
POINTER TO MEM. 12288
60 FORCHAR=60TO63:REM PROGRAM CHARACTERS 60 THRU 63
80 FORBYTE=0TO7:REM DO ALL 8 BYTES OF A CHARACTER
100 READ NUMBER:REM READ IN 1/8TH OF CHARACTER DATA
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM STORE THE
DATA IN MEMORY
140 NEXTBYTE:NEXTCHAR:REM ALSO COULD BE NEXT BYTE,
CHAR
150 PRINTCHR$(147)TAB(255)CHR$(60);
155 PRINTCHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LINE 150 PUTS THE NEWLY DEFINED CHARACTERS
ON THE SCREEN
170 GETA$:REM WAIT FOR USER TO PRESS A KEY
180 IFA$=""THENGOTO170:REM IF NO KEYS WERE PRESSED,
 TRY AGAIN!
190 POKE53272,21:REM RETURN TO NORMAL CHARACTERS
200 DATA4,6,7,5,7,7,3,3:REM DATA FOR CHARACTER 60
210 DATA 32,96,224,160,224,224,192,192:REM DATA
FOR CHARACTER 61
220 DATA7,7,7,31,31,95,143,127:REM DATA FOR
CHARACTER 62
230 DATA 224,224,224,248,248,248,240,224:REM DATA
FOR CHARACTER 63
240 END
```

MEHRFARBIGE GRAPHIKEN

Durch die standardmäßige hochauflösende Graphik können Sie selbst Einzelpunkte auf dem Bildschirm ansteuern. Für jeden Punkt im Zeichenspeicher stehen zwei Werte zur Verfügung: 1 für EIN und 0 für AUS. Hat ein Punkt den Wert 1, so wird er in der von Ihnen für die jeweilige Bildschirmposition gewählten Farbe angezeigt. Bei der hochauflösenden Graphik können alle Punkte innerhalb der 8-mal-8-Matrix entweder in der Hinter- oder Vordergrundfarbe angezeigt werden. Hierdurch wird die Farbauflösung innerhalb dieses Bereiches eingeschränkt. So können z. B. Schwierigkeiten entstehen, wenn sich zwei Linien mit verschiedenen Farben kreuzen.
Dieses Problem wird durch den Mehrfarbenmodus gelöst. Hierbei kann jeder Punkt eine von vier Farben haben: Bildschirmfarbe (Hintergrundfarbregister #0), die Farbe im Hintergrundregister #1, die Farbe im Hintergrundfarbregister #2 oder die Zeichenfarbe. Die einzige Einschränkung liegt in der horizontalen Auflösung, da im Mehrfarbenmodus jeder Punkt doppelt so breit ist wie bei Hochauflösung. Es überwiegen jedoch bei weitem die vielen Vorteile des Mehrfarbenmodus.

DAS MEHRFARBEN-MODUS-BIT

Zum Einschalten des Modus für mehrfarbige Zeichen wird Bit 4 des Steuerregisters des VIC-II durch folgende POKE-Anweisung bei 53270 ($D016) auf 1 gesetzt:

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

Zum Abschalten dieser Betriebsart wird Bit 4 an Speicherplatz 53270 durch nachstehende POKE-Anweisung auf 0 gesetzt:

```
POKE 53270,PEEK(53270)AND 239
```

Der Mehrfarben-Modus wird für jede Bildschirmstelle ein- oder ausgeschaltet, so daß Mehrfarbengraphiken und Graphiken mit hoher Auflösung (hi-res) kombiniert werden können. Dies wird über Bit 3 im Farbspeicher gesteuert. Der Farbspeicher beginnt bei 55296 ($D800 HEX). Ist die Zahl im Farbspeicher kleiner als 8 (0–7), so gilt für die entsprechende Stelle auf dem Bildschirm Hochauflösung in der gewählten Farbe (0–7). Ist die Zahl im Farbspeicher größer oder gleich 8 (von 8 bis 15), dann wird die entsprechende Stelle im Mehrfarbenmodus angezeigt.

Die Zeichenfarbe einer Bildschirm-Position kann durch eine POKE-Anweisung im Farbspeicher geändert werden. Durch das POKEn einer Zahl von 0 bis 7 werden die Zeichen in normaler Farbdarstellung angezeigt. Durch das POKEn einer Zahl zwischen 8 und 15 gilt für die entsprechende Bildschirmstelle der Mehrfarbenmodus. D. h., durch das Einschalten von Bit 3 im Farbspeicher wird der Mehrfarbenmodus und durch Ausschalten der normale Hochauflösungsmodus gewählt.
Gilt für eine Bildschirmstelle der Mehrfarbenbetrieb, so wird durch die Zeichen-Bits bestimmt, welche Farben für die Punkte angezeigt werden. Nachstehend sehen Sie z. B. die Darstellung des Buchstabens A und das entsprechende Bit-Muster:

DARSTELLUNG	BIT-MUSTER
**	00011000
*****	00111100
** **	01100110
******	01111110
** **	01100110
** **	01100110
** **	01100110
	00000000

Im normalen oder "hi-res" (d. h. hochauflösenden Graphik)-Modus wird die Bildschirmfarbe bei jedem 0-Bit und die Zeichenfarbe stets da angezeigt, wo das Bit 1 ist. Beim Mehrfarbenmodus werden die Bits paarweise benutzt:

DARSTELLUNG	BIT-MUSTER
AABB	00 01 10 00
CCCC	00 11 11 00
AABBAABB	01 10 01 10
AACCCCBB	01 11 11 10
AABBAABB	01 10 01 10
AABBAABB	01 10 01 10
AABBAABB	01 10 01 10
	00 00 00 00

Im obigen Bildbereich werden die durch AA gekennzeichneten Stellen in der Hintergrundfarbe #1, die durch BB gekennzeichneten Stellen in der Hintergrundfarbe #2 und die durch CC gekennzeichneten Stellen in der Zeichenfarbe dargestellt. Dies wird entsprechend nachstehender Tabelle durch die Bit-Paare bestimmt:

BIT-PAAR	FARBREGISTER	SPEICHERPLATZ
00	Hintergrundfarbe #0 (Bildschirmfarbe)	53281 ($D021)
01	Hintergrundfarbe #1	53282 ($D022)
10	Hintergrundfarbe #2	53283 ($D023)
11	Durch die unteren 3 Bits im Farbspeicher bestimmte Farbe	Farbspeicher

Geben Sie NEW und nachstehendes Programm ein:

```
100 POKE53281,1:REM SET BACKGROUND COLOR #0 TO
WHITE
110 POKE53282,3:REM SET BACKGROUND COLOR #1 TO CYAN
120 POKE53283,8:REM SET BACKGROUND COLOR #2 TO
ORANGE
130 POKE53270,PEEK(53270)OR16:REM TURN ON
MULTICOLOR MODE
140 C=13*4096+8*256:REM SET C TO POINT TO COLOR
MEMORY
150 PRINTCHR$(147)"AAAAAAAAAA"
160 FORL=0TO9
170 POKEC+L,8:REM USE MULTI BLACK
180 NEXT
```

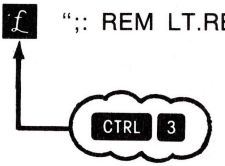
Die Bildschirmfarbe ist weiß, die Zeichenfarbe schwarz, ein Farbregister zyan (grünblau) und das andere orange.
Sie geben nicht tatsächlich Farb-Codes in die Stellen für die Zeichenfarbe ein, sondern benutzen eigentlich Hinweise auf die jeweiligen Farbregister. Hierdurch wird Speicherplatz gespart, da zwei Bits benutzt werden, um zwischen 16 bzw. 8 Farben (Hintergrund bzw. Zeichen) zu wählen. Hierdurch werden einige raffinierte Tricks möglich. Durch einfaches Ändern eines der indirekten Register wird jeder Punkt, der in dieser Farbe gezeichnet ist, geändert.

Alles, was in Bildschirm- und Hintergrundfarben angezeigt ist, kann daher sofort auf dem gesamten Bildschirm geändert werden. Nachstehend sehen Sie ein Beispiel zur Änderung des Hintergrundfarbregisters #1:

```
100 POKE53270,PEEK(53270)OR16:REM TURN ON
MULTICOLOR MODE
110 PRINTCHR$(147)CHR$(18);
120 PRINT"■■";:REM TYPE C= & 1 FOR ORANGE OR
MULTICOLOR BLACK BACKGROUND
130 FORL=1TO22:PRINTCHR$(65);:NEXT
135 FORT=1TO500:NEXT
140 PRINT"■";:REM TYPE CTRL & 7 FOR BLUE COLOR
CHANGE
145 FORT=1TO500:NEXT
150 PRINT"■HIT A KEY"
160 GETA$:IFA$=""THEN160
170 X=INT(RND(1)*16)
180 POKE 53282,X
190 GOTO 160
```

Über die Taste C= und die Farbtasten können allen Zeichen, einschließlich den Mehrfarbenzeichen, beliebige Farben gegeben werden. Geben Sie z. B. folgenden Befehl ein:

POKE 53270,PEEK(53270)OR 16:PRINT " £ ";: REM LT.RED/MULTI-COLOR RED



Das Wort READY und alles übrige, was Sie über die Tastatur eingeben, wird im Mehrfarbenmodus angezeigt. Durch eine andere Farbsteuerung können Sie wieder den Normal-Modus wählen.

Nachstehend sehen Sie ein Programmbeispiel mit programmierbaren Mehrfarbenzeichen:

```
10 REM * EXAMPLE 2 *
20 REM CREATING MULTI COLOR PROGRAMMABLE CHARACTERS
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM CHARACTER RANGE TO BE COPIED
FROM ROM
36 FORJ=0TO7:REM COPY ALL 8 BYTES PER CHARACTER
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPY A
BYTE
38 NEXTJ,I:REM GOTO NEXT BYTE OR CHARACTER
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM
TURN ON I/O AND KB
40 POKE53272,(PEEK(53272)AND240)+12:REM SET CHAR
POINTER TO MEM. 12288
50 POKE53270,PEEK(53270)OR16
51 POKE53281,0:REM SET BACKGROUND COLOR #0 TO BLACK
52 POKE53282,2:REM SET BACKGROUND COLOR #1 TO RED
53 POKE53283,7:REM SET BACKGROUND COLOR #2 TO
YELLOW
60 FORCHAR=60TO63:REM PROGRAM CHARACTERS 60 THRU 63
80 FORBYTE=0TO7:REM DO ALL 8 BYTES OF A CHARACTER
100 READNUMBER:REM READ 1/8TH OF CHARACTER DATA
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM STORE THE
DATA IN MEMORY
140 NEXTBYTE,CHAR
150
PRINT"□"TAB(255)CHR$(60)CHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LINE 150 PUTS THE NEWLY DEFINED CHARACTERS
ON THE SCREEN
170 GETA$:REM WAIT FOR USER TO PRESS A KEY
180 IFA$=""THEN170:REM IF NO KEYS WERE PRESSED,
TRY AGAIN
190 POKE53272,21:POKE53270,PEEK(53270)AND239:REM
RETURN TO NORMAL CHARACTERS
200 DATA129,37,21,29,93,85,85,85:REM DATA FOR
CHARACTER 60
210 DATA66,72,84,116,117,85,85,85:REM DATA FOR
CHARACTER 61
220 DATA87,87,85,21,8,8,40,0:REM DATA FOR
CHARACTER 62
230 DATA213,213,85,84,32,32,40,0:REM DATA FOR
CHARACTER 63
240 END
```

SHIFT CLR/HOME

ERWEITERTER HINTERGRUNDFARBMODUS

In diesem Modus können Sie für jedes einzelne Zeichen die Farbe sowohl im Hintergrund als auch im Vordergrund steuern. So ist es z. B. möglich, auf einem weißen Bildschirm ein blaues Zeichen mit gelbem Hintergrund anzuzeigen.
Für den erweiterten Hintergrundfarbmodus stehen vier Register zur Verfügung. Für jedes Register kann eine der 16 Farben gewählt werden.
In diesem Modus wird über den Farbspeicher die Vordergrundfarbe festgelegt. Die Anwendung ist die gleiche wie beim Standard-Zeichenmodus.
Beim erweiterten Modus ist die Anzahl der verschiedenen anzeigbaren Zeichen jedoch eingeschränkt. Ist der erweiterte Farbmodus eingeschaltet, können nur die ersten 64 Zeichen des Zeichen-ROM (oder die ersten 64 in Ihrem programmierbaren Zeichensatz) benutzt werden. Zwei Bits des Zeichen-Codes werden nämlich für die Wahl der Hintergrundfarbe benutzt.
Der Zeichen-Code (die auf dem Bildschirm gePOKEte Zahl) vom Buchstaben A ist eine 1. Im erweiterten Farbmodus erscheint nach dem POKEn einer 1 ein A. Normalerweise muß nach dem POKEn von 65 das Zeichen mit dem Zeichen-Code (CHR$) 129, also ein umgekehrtes "A", erscheinen. Dies passiert nicht im erweiterten Farbmodus. Es erscheint genau das gleiche "A" wie vorher, jedoch eine andere Hintergrundfarbe. Entnehmen Sie die Codes der nachstehenden Tabelle:

ZEICHENCODE			HINTERGRUNDFARBREGISTER	
BEREICH	BIT 7	BIT 6	NUMMER	ADRESSE
0– 63	0	0	0	53281 ($D021)
64–127	0	1	1	53282 ($D022)
128–191	1	0	2	53283 ($D023)
192–255	1	1	3	53284 ($D024)

Zum Einschalten des erweiterten Farbmodus wird Bit 6 des VIC-II-Registers mit der Adresse 53265 ($D011 in HEX) auf 1 gesetzt. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53265, PEEK(53265)OR 64
```

Zum Ausschalten des erweiterten Farbmodus wird Bit 6 des VIC-II-Registers mit der Adresse 53265 ($D011) auf 0 gesetzt. Hierzu dient folgende Anweisung:

```
POKE 53265, PEEK(53265)AND 191
```

GRAPHIKEN DURCH BIT-MAPPING

Beim Schreiben von Spielen, Zeichnen von Tabellen für Geschäftsanwendungen oder Schreiben von sonstigen Programmen werden Sie früher oder später Bildschirmdarstellungen mit hoher Auflösung benötigen.
Der COMMODORE 64 wurde genau hierfür konstruiert: Hohe Auflösung wird durch "Bit-Mapping" des Bildschirms möglich. "Bit-Mapping" ist die Methode, bei der jedem darstellbaren Punkt (Pixel) auf dem Bildschirm sein eigenes Bit (Platz) im Speicher zugeordnet wird. Ist dieses Speicherbit eine 1, so ist der entsprechende Punkt eingeschaltet. Ist das Bit 0, so ist der Punkt ausgeschaltet.
Das Arbeiten mit Graphiken mit hoher Auflösung hat jedoch einige Nachteile und wird daher nicht immer benutzt. Zunächst wird durch das Bit-Mapping des gesamten Bildschirms erhebliche Speicherkapazität in Anspruch genommen. Jeder Pixel benötigt nämlich ein Speicherbit, d. h., Sie brauchen 1 Byte für 8 Pixel. Da jedes Zeichen eine 8-mal-8-Matrix ist und 40 Zeilen mit 25 Zeichen vorhanden sind, beträgt die Auflösung 320 Pixel (Punkte) mal 200 Pixel für den gesamten Bildschirm. Hieraus ergeben sich 64000 Punkte, von denen jeder ein Speicherbit benötigt. Für ein Bit-Mapping des gesamten Bildschirms brauchen Sie also 8000 Byte.
Im allgemeinen bestehen Operationen zur Erstellung von Graphiken hoher Auflösung aus mehreren kurzen, einfachen Wiederhol-Routinen. Dies ist für diese Zwecke meist zu BASIC ziemlich langsam. Die Maschinensprache ist jedoch am besten für solche kurzen, einfachen Wiederhol-Routinen geeignet. Sie sollten die Programme daher ganz in Maschinensprache schreiben oder in maschinensprachegeschriebene hi-res-Routinen vom BASIC-Programm über den Befehl SYS aufrufen. Auf diese Weise können Sie sowohl die Einfachheit von BASIC als auch die Geschwindigkeit der Maschinensprache bei Graphiken nutzen.
Alle in diesem Kapitel gegebenen Beispiele sind in BASIC. Nun zu den technischen Details.
BIT-MAPPING ist die am weitesten verbreitete Graphiktechnik in der Computerwelt. Dieses Verfahren wird benutzt, um Bilder mit großem Detailreichtum zu erstellen. Grundsätzlich zeigt der COMMODORE 64 direkt einen 8K-Speicherbereich auf dem Bildschirm an, wenn er sich im Bit-Map-Modus befindet.
Im Bit-Map-Modus können Sie *direkt* steuern, ob ein einzelner Punkt auf dem Bildschirm an oder aus ist.
Mit dem COMMODORE 64 stehen zwei verschiedene Arten von Bit-Mapping zur Verfügung:

1) Standard-Bit-Map-Modus (hi-res) (320 mal 200 Punktauflösung)
2) Mehrfarben-Bit-Map-Modus (160 mal 200 Punktauflösung)

Beim Standard-Bit-Mapping ist zwar die Auflösung größer, es stehen jedoch weniger Farbmöglichkeiten zur Verfügung. Beim Mehrfarben-Bit-Mapping wird eine geringere horizontale Auflösung durch die Möglichkeit wettgemacht, eine größere Anzahl von Farben in einem 8-mal-8-Punktefeld unterzubringen.

STANDARD-BIT-MAPPING MIT HOHER AUFLÖSUNG

Beim Standard-Bit-Mapping haben Sie eine Auflösung von 320 mal 200 Punkten und können in jedem 8-mal-8-Punktebereich zwischen zwei Farben wählen. Zum Einschalten des Bit-Mapping-Betriebs wird Bit 5 des VIC-II-Kontrollregisters in Adresse 53265 ($D011 in HEX) auf 1 gesetzt. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53265,PEEK(53265)OR 32
```

Zum Abschalten dieser Betriebsart wird Bit 5 des VIC-II-Kontrollregisters in Adresse 53265 ($D011) auf 0 gesetzt. Hierzu dient folgende Anweisung:

```
POKE 53265,PEEK(53265)AND 223
```

Bevor wir uns nun im einzelnen mit dem Bit-Map-Modus beschäftigen, müssen wir zuvor ein weiteres Problem lösen: Die Plazierung des Bit-Mapping-Bereichs.

FUNKTIONSWEISE

Wenn Sie noch den Abschnitt über PROGRAMMIERBARE ZEICHEN in Erinnerung haben, werden Sie sich erinnern, daß Sie das Bitmuster eines im RAM gespeicherten Zeichens beliebig wählen können. Genauso wie Sie ein auf dem Bildschirm angezeigtes Zeichen ändern können, können Sie auch einen einzelnen Punkt ändern. Dies ist das Grundmerkmal von Bit-Mapping.
Der gesamte Bildschirm ist mit programmierbaren Zeichen belegt. Ihre Änderungen erfolgen direkt in dem Speicher, von dem die programmierbaren Zeichen ihre Muster erhalten.
Jeder Platz im Bildschirmspeicher, der im Normalmodus für die Steuerung der Zeichenwiedergabe benutzt wurde, wird nun für die Farbinformation herangezogen. So wird nun durch das POKEn einer 1 in Speicherplatz 1024 nicht mehr ein "A" links oben auf dem Bildschirm angezeigt, sondern durch Speicherplatz 1024 werden nun die Farben der Bits in der linken oberen Ecke des Bildschirms gesteuert.

Beim Bit-Mapping-Betrieb kommen die Farben der 1000 Bildschirmfelder nicht vom Farbspeicher wie im Normalmodus. Die Farben werden vielmehr aus dem Bildschirmspeicher genommen. Die oberen vier Bits des Bildschirmspeichers legen die Farben der Bits fest, die im über diesen Bildschirmspeicherplatz gesteuerten 8-mal-8-Bereich auf 1 gesetzt sind. Die unteren vier Bits enthalten die Farben für jedes Bit, das auf 0 gesetzt ist.

BEISPIEL: Geben Sie folgendes ein:

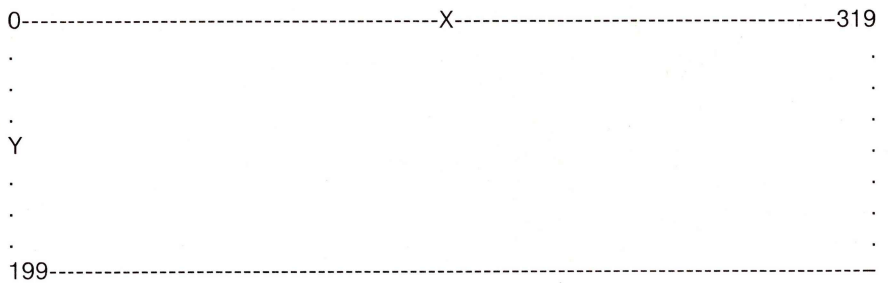
```
5 BASE=2*4096:POKE53272,PEEK(53272)OR8:REM PUT BIT
MAP AT 8192
10 POKE53265,PEEK(53265)OR32:REM ENTER BIT MAP MODE
```

Geben Sie nun zum Ausführen des Programms RUN ein.
Auf dem Bildschirm erscheint nichts Brauchbares, stimmt's? Wie der "normale" Bildschirm, muß auch der HI-RES-Bildschirm zuvor gelöscht werden. In diesem Fall funktioniert dies leider nicht durch CLR. Sie müssen vielmehr den Speicherbereich löschen, den Sie für Ihre programmierbaren Zeichen benutzen. Drücken Sie die Tasten RUN/STOP und RESTORE und fügen Sie dann zum Löschen des HI-RES-Bildschirms folgende Zeilen in Ihr Programm ein:

```
20 FORI=BASETOBASE+7999:POKEI,0:NEXT:REM CLEAR BIT
MAP
30 FORI=1024TO2023:POKEI,3:NEXT:REM SET COLOR TO
CYAN AND BLACK
```

Geben Sie nun erneut RUN ein. Der Bildschirm wird nun gelöscht und die grünblaue Farbe (zyan) auf dem ganzen Bildschirm angezeigt werden. Nun wollen wir einzelne Punkte auf dem HI-RES-Bildschirm ein- und ausschalten.
Um einen Punkt zu setzen (einzuschalten) oder zu löschen (auszuschalten), müssen Sie wissen, wie Sie das richtige Bit im Zeichenspeicher finden, das auf 1 gesetzt werden soll. D. h., Sie müssen das zu ändernde Zeichen, die Zeichenreihe sowie das entsprechende Bit dieser Reihe finden. Für diese Berechnung benötigen Sie eine Gleichung.

Wir benutzen X und Y für die horizontale bzw. vertikale Punktposition. Der Punkt, an dem X gleich 0 und Y gleich 0 ist, befindet sich oben links in der Anzeige. Rechts liegende Punkte haben höhere X-Werte und alle Punkte, die darunter liegen, höhere Y-Werte. Bit-Mapping läßt sich am sinnvollsten nutzen, wenn die Anzeige folgendermaßen angeordnet ist:



Jeder Punkt hat eine X- und eine Y-Koordinate. In diesem Koordinatensystem läßt sich die Lage jedes Punktes auf dem Bildschirm leicht beschreiben.
Anschaulich gesehen ist die Reihenfolge der Bytes auf dem Bildschirm wie folgt:

OBERSTE ZEILE (REIHE 0)

------	BYTE 0	BYTE 8	BYTE 16 BYTE 24	BYTE 312
	BYTE 1	BYTE 9	. .	BYTE 313
	BYTE 2	BYTE 10	. .	BYTE 314
	BYTE 3	BYTE 11	. .	BYTE 315
	BYTE 4	BYTE 12	. .	BYTE 316
	BYTE 5	BYTE 13	. .	BYTE 317
	BYTE 6	BYTE 14	. .	BYTE 318
------	BYTE 7	BYTE 15	. .	BYTE 319

ZWEITE ZEILE (REIHE 1)

------	BYTE 320	BYTE 328	BYTE 336 BYTE 344	BYTE 632
	BYTE 321	BYTE 329	. .	BYTE 633
	BYTE 322	BYTE 330	. .	BYTE 634
	BYTE 323	BYTE 331	. .	BYTE 635
	BYTE 324	BYTE 332	. .	BYTE 636
	BYTE 325	BYTE 333	. .	BYTE 637
	BYTE 326	BYTE 334	. .	BYTE 638
------	BYTE 327	BYTE 335	. .	BYTE 639

Die programmierbaren Zeichen der Bit-MAP sind in 25 Reihen mit je 40 Spalten angeordnet. Dies ist zwar für den Textaufbau eine gute Methode, erschwert jedoch das Bit-Mapping. (Für diese Methode gibt es einen guten Grund. Siehe Abschnitt KOMBINIERTE BETRIEBSARTEN.)
Durch nachstehende Gleichung läßt sich ein Punkt in der Bit-Map-Anzeige leichter steuern:
Der Anfang des Bildschirm-Speicherbereichs wird als BASIS bezeichnet. Die Reihenzahl (von 0 bis 24) Ihres Punkts ist:

```
ROW = INT(Y/8) (Es gibt 320 Bytes pro Zeile)
```

Die Zeichenposition dieser Zeile (von 0 bis 39) lautet:

```
CHAR = INT(X/8) (Es gibt 8 Bytes pro Zeichen)
```

Die Zeile dieser Zeichenposition (von 0 bis 7) lautet:

```
LINE = Y AND 7
```

Das Bit dieses Bytes ist:

```
BIT = 7-(X AND 7)
```

Nun setzen wir diese Gleichungen zusammen. Das Byte, in dem der Zeichenspeicherpunkt (X, Y) liegt, wird wie folgt berechnet:

```
BYTE = BASE + ROW*320+ CHAR*8 + LINE
```

Um ein beliebiges Bit im Gitter mit Koordinaten (X, Y) einzuschalten, verwenden Sie diese Zeile:

```
POKE BYTE, PEEK(BYTE) OR 2↑BIT
```

Wir fügen diese Berechnungen in das Programm ein. In folgendem Beispiel zeichnet der COMMODORE 64 eine Sinuskurve:

```
50 FORX=0TO319STEP.5:REM WAVE WILL FILL THE SCREEN
60 Y=INT(90+80*SIN(X/10))
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LN=YAND7
90 BY=BASE+RO*320+8*CH+LN
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2↑BI)
120 NEXTX
125 POKE1024,16
130 GOTO130
```

Durch die Gleichung in Zeile 60 werden die Werte für die Sinusfunktion von +1 bis −1 in 10 bis 170 umgeändert. In den Zeilen 70 bis 100 werden das Zeichen, die Reihe, das Byte und zugehörige Bit berechnet. Zeile 125 signalisiert, daß das Programm beendet ist, indem sich in der oberen linken Bildschirmecke die Farbe ändert. Durch Zeile 130 wird das Programm in eine unendliche Schleife geführt. Nach dem Betrachten der Graphik drücken Sie einfach gleichzeitig die Tasten RUN/STOP und RESTORE.

Als weiteres Beispiel wird das Sinuskurvenprogramm so geändert, daß ein Halbkreis angezeigt wird. Hierzu ist das Programm wie folgt abzuändern:

```
50 FORX=0TO160:REM DO HALF THE SCREEN
55 Y1=100+SQR(160*X-X*X)
56 Y2=100-SQR(160*X-X*X)
60 FORY=Y1TOY2STEPY1-Y2
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LN=YAND7
90 BY=BASE+RO*320+8*CH+LN
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2↑BI)
114 NEXT
```

Hierdurch wird im HI-RES-Bereich des Bildschirms ein Halbkreis erstellt.

Achtung: BASIC-Variablen können Ihren HI-RES-Bildschirm überlagern. Wenn Sie mehr Speicherplatz benötigen, müssen Sie den Anfang von BASIC über den HI-RES-Bildschirmbereich legen, oder aber Sie verschieben Ihren HI-RES-Bildschirmbereich. Dieses Problem ergibt sich nicht bei der Maschinensprache. Es tritt lediglich dann auf, wenn die Programme in BASIC geschrieben wurden.

MEHRFARBEN-BIT-MAPPING

Wie beim Mehrfarben-Modus der Zeichen können auch beim Mehrfarben-Bit-Mapping in jedem 8-mal-8-Bereich der Bit-Map bis zu vier verschiedene Farben angezeigt werden.
Auch hier wird die horizontale Auflösung reduziert (von 320 auf 160 Punkte).
Beim Mehrfarben-Bit-Mapping wird für die Bit-Map ein 8K-Speicherbereich benutzt. Sie wählen die Farben für das Mehrfarben-Bit-Mapping (1) vom Hintergrund-Farbregister 0 (Bildschirm-Hintergrundfarbe), (2) von der Video-Matrix (die oberen 4 Bits geben eine der möglichen Farben, die unteren 4 Bits eine weitere) und (3) vom Farbspeicher.
Zum Einschalten dieses Modus wird Bit 5 von 53265 ($D011) und Bit 4 in Adresse 53270 ($D016) auf 1 gesetzt. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53265,PEEK(53625)OR 32: POKE 53270,PEEK(53270)OR 16
```

Zum Ausschalten des Mehrfarben-Bit-Map-Modus wird Bit 5 in 53265 ($D011) und Bit 4 in 53270 ($D016) auf 0 gesetzt. Hierzu dient folgende POKE-Anweisung:

```
POKE 53265,PEEK(53265)AND 223: POKE 53270,PEEK(53270)AND 239
```

Wie beim Standard-Bit-Map-Modus (HI-RES) besteht eine 1:1-Entsprechung zwischen dem für die Anzeige benutzten 8K-Speicherbereich und der Bildschirm-Darstellung. Die horizontalen Punkte sind jedoch immer zwei Bits breit. Jeweils zwei Bits im Anzeigenspeicher bilden einen Punkt, der eine von vier Farben haben kann.

BITS	FARBINFORMATION KOMMT VON
00	Hintergrundfarbe #0 (Bildschirmfarbe)
01	Oberen 4 Bits des Bildschirmspeichers
10	Unteren 4 Bits des Bildschirmspeichers
11	Farbnybble (Nybble = ½ Byte = 4 Bit)

KONTINUIERLICHES VERSCHIEBEN

Über den VIC-II-Chip ist ein einfaches "Verschieben" (Bildschirm-Rollen) sowohl in horizontalen als auch vertikalen Richtungen möglich. Das Verschieben ist eine Ein-Pixel-Bewegung des gesamten Bildschirms in eine Richtung. Die Bewegung kann entweder nach oben, unten, links oder rechts erfolgen. Hierdurch werden neue Informationen angezeigt, und gleichzeitig verschwinden andere Zeichen auf der gegenüberliegenden Seite.
Auch wenn der VIC-II-Chip Ihnen viele Aufgaben abnimmt, muß dieses Verschieben doch über ein Maschinensprache-Programm erfolgen. Über den VIC-II-Chip kann der Video-Bildschirm in eine beliebige von 8 horizontalen und 8 vertikalen Positionen gebracht werden. Die Positionierung wird über die VIC-II-Register zum Bildschirmrollen (genannt SCROLL-Register) gesteuert. Der VIC-II-Chip hat auch einen 38-Spalten- und 24-Reihen-Modus. Die kleineren Bildschirmgrößen geben Ihnen einen Platz für die neuen Daten, die beim Verschieben gebraucht werden.
Gehen Sie für das Verschieben wie folgt vor:

1) Den Bildschirm verkleinern (der Rahmen wird breiter).
2) Das SCROLL-Register auf den Maximalwert stellen (oder auf den Minimalwert je nach Richtung des Rollens).
3) Die neuen Daten in den geeigneten Bildschirmbereich eingeben.
4) Das SCROLL-Register vergrößern (oder verkleinern), bis es den Maximalwert (oder Minimalwert) erreicht.
5) Zu diesem Zeitpunkt den gesamten Bildschirm um ein Zeichen in Verschieberichtung rollen. Benutzen Sie hierzu Ihre Maschinensprache-Routine.
6) Nun zu Schritt 2 zurückgehen.

Um in den 38-Spalten-Modus zu gehen, wird Bit 3 von Adresse 53270 ($D016) auf 0 gesetzt. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53270,PEEK(53270)AND 247
```

Zur Rückkehr in den 40-Spalten-Modus wird Bit 3 von Adresse 53270 ($D016) auf 1 gesetzt. Hierzu dient folgende POKE-Anweisung:

```
POKE 53270,PEEK(53270)OR 8
```

Für den 24-Reihen-Modus wird Bit 3 von Adresse 53265 ($D011) auf 0 gesetzt. Hierzu dient folgende POKE-Anweisung:

```
POKE 53265,PEEK(53265)AND 247
```

Zur Rückkehr in den 25-Reihen-Modus wird Bit 3 von Adresse 53265 ($D011) durch folgende POKE-Anweisung auf 1 gesetzt:

```
POKE 53265,PEEK(53265)OR 8
```

Beim Verschieben in X-Richtung muß der VIC-II-Chip in den 38-Spalten-Modus versetzt werden. Hierdurch wird für die neuen Daten Platz geschaffen. Beim Verschieben nach links werden die neuen Daten rechts eingegeben. Beim Schieben nach rechts erscheinen die neuen Daten entsprechend auf der linken Seite. Bitte beachten Sie, daß der Bildschirmspeicher noch 40 Spalten hat. Lediglich 38 sind jedoch sichtbar.
Beim Verschieben in Y-Richtung muß der VIC-II-Chip in 24-Reihen-Modus versetzt werden. Beim Rollen nach oben werden die neuen Daten in die letzte Reihe eingegeben. Beim Rollen nach unten erscheinen die neuen Daten entsprechend in der ersten Reihe. Beim X-Verschieben sind unsichtbare Bereiche auf beiden Bildschirmseiten. Beim Y-Verschieben gibt es jedoch nur einen unsichtbaren Bereich.
Ist das Y-SCROLL-Register auf 0 gesetzt, dann ist die erste Zeile unsichtbar und bereit für neue Daten. Ist das Y-SCROLL-Register auf 7 gesetzt, so ist die letzte Reihe unsichtbar.
Zum Rollen in X-Richtung befindet sich das SCROLL-Register in den Bits 2 bis 0 des VIC-II-Steuerregisters in Adresse 53270 ($D016 HEX).
Auch hier dürfen auf jeden Fall nur diese Bits verändert werden. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53270, (PEEK(53270)AND 248)+X
```

wobei X die X-Bildschirmposition 0 bis 7 ist.
Zum Rollen in Y-Richtung befindet sich das SCROLL-Register in den Bits 2 bis 0 des VIC-II-Steuerregisters in Adresse 53265 ($D011 HEX). Auch hierbei dürfen wieder nur diese Bits verändert werden. Hierzu dient folgende POKE-Anweisung:

```
POKE 53265, (PEEK(53265)AND 248)+Y
```

wobei Y die Y-Bildschirmposition 0 bis 7 angibt.
Um den Text von unten auf den Bildschirm zu rollen, müssen die unteren 3 Bits von Adresse 53265 von 0–7 gesetzt, weitere Daten in die abgedeckte Zeile unten auf den Bildschirm eingegeben und danach der Vorgang wiederholt werden.
Ändert man die Verschiebebits mit der Schrittweite von –1, so wird der Text in entgegengesetzter Richtung bewegt.

BEISPIEL: Textrollen vom unteren Bildschirmrand her:

```
10 POKE53265,PEEK(53265)AND247              :REM GO
INTO 24 ROW MODE
20 PRINTCHR$(147)                           :REM
CLEAR THE SCREEN
30 FORX=1TO24:PRINTCHR$(17);:NEXT           :REM MOVE
THE CURSOR TO THE BOTTOM
40 POKE53265,(PEEK(53265)AND248)+7:PRINT :REM
POSITION FOR 1ST SCROLL
50 PRINT"      HELLO";
60 FORP=6TO0STEP-1
70 POKE53265,(PEEK(53265)AND248)+P
80 FORX=1TO50:NEXT                          :REM
DELAY LOOP
90 NEXT:GOTO40
```

SPRITES

Ein SPRITE ist ein besonderer Typ von freidefinierbaren Zeichen, die an beliebiger Stelle auf dem Bildschirm angezeigt werden können. Sprites werden direkt vom VIC-II-Chip verwaltet. Sie brauchen lediglich für jedes Sprite festzulegen, "wie es aussehen soll", "welche Farbe es haben soll" und "wo es auf dem Bildschirm plaziert werden soll". Der VIC-II-Chip erledigt für Sie den Rest! Sprites können eine der 16 möglichen Farben haben.
Sprites können zusammen mit jedem beliebigen Graphik-Modus, Bit-Mapping, Zeichen, Mehrfarben-Modus usw. benutzt werden. Eine Spritedefinition enthält die Farbe, den Modus (HI-RES oder Mehrfarben) und die Form.
Vom VIC-II-Chip können automatisch gleichzeitig jeweils 8 Sprites verwaltet werden. Durch RASTER-INTERRUPT-Techniken können weitere Sprites angezeigt werden.
Sprites haben folgende Merkmale:

1) Punktgröße 24 mal 21 (horizontal x vertikal)
2) Farbsteuerung für jedes Sprite
3) Sprites im Mehrfarbenmodus
4) Vergrößerung (2x) in horizontaler und/oder vertikaler Richtung
5) Wahlmöglichkeit: Sprites vor oder hinter dem Hintergrund
6) Wahl der Reihenfolge, in der die Sprites "hintereinander" angeordnet sind
7) Sprite-Kollisionserkennung
8) Kollisionserkennung zwischen Sprite und Hintergrund.

Auf diese Weise lassen sich zahlreiche Tele-Spiele einfach programmieren. Da die Sprites durch das Betriebssystem unterstützt werden, kann ein gutes Spiel sogar in BASIC geschrieben werden!
Vom VIC-II-Chip werden 8 Sprites unterstützt. Sie sind von 0 bis 7 numeriert. Jedes Sprite hat seinen eigenen Speicherbereich für das Bitmuster, seine Positions- und Farbregister sowie seine eigenen Bits zur Erkennung von Kollisionen und zum Ein- und Ausschalten.

SPRITEDEFINITION

Sprites werden genau wie programmierbare Zeichen definiert. Da ein Sprite jedoch größer ist, werden mehr Bytes benötigt. Jedes Sprite besteht aus 24 mal 21 oder 504 Punkten. Für die Definition eines Sprites werden also 63 Bytes (504/8) benötigt.

SPALTEN-NR.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
BIT	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BITWERTE (EIN = 1)	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
REIHE 0																								
REIHE 1																								
REIHE 2																								
REIHE 3																								
REIHE 4																								
REIHE 5																								
REIHE 6																								
REIHE 7																								
REIHE 8																								
REIHE 9																								
REIHE 10																								
REIHE 11																								
REIHE 12																								
REIHE 13																								
REIHE 14																								
REIHE 15																								
REIHE 16																								
REIHE 17																								
REIHE 18																								
REIHE 19																								

Abb. 3.2. Spritedefinition

Die 63 Bytes sind in 21 Reihen zu je 3 Bytes angeordnet. Eine Spritedefinition sieht folgendermaßen aus:

BYTE 0	BYTE 1	BYTE 2
BYTE 3	BYTE 4	BYTE 5
BYTE 6	BYTE 7	BYTE 8
..	..	..
..	..	..
..	..	..
BYTE 60	BYTE 61	BYTE 62

Auch bei Betrachtung der Spritedefinition auf Bit-Ebene läßt sich erkennen, wie Sprites erstellt werden. (Siehe Abb. 3.2.)
Bei einem Standardsprite (HI-RES) wird jedes auf 1 gesetzte Bit in der entsprechenden Sprite-Vordergrundfarbe angezeigt. Jedes auf 0 gesetzte Bit ist transparent, so daß der dahinter liegende Wert sichtbar wird. Dies entspricht der Situation beim Standardzeichen.
Mehrfarbige Sprites entsprechen mehrfarbigen Zeichen. Horizontal wird die Ortsauflösung zugunsten der Farbauflösung verschlechtert. Die Auflösung beträgt 12 mal 21 Punkte (horizontal x vertikal). Jeder Punkt im Sprite wird doppelt so breit, die Anzahl der anzeigbaren Farben im Sprite wird jedoch auf 4 erhöht.

SPRITE-POINTER

Auch wenn jedes Sprite für die Definition nur 63 Bytes benötigt, wird doch ein weiteres Byte am Ende jedes Sprites als Platzhalter benötigt, d. h., daß jedes Sprite 64 Bytes beansprucht. Auf diese Weise können Sie leicht berechnen, wo sich Ihre Spritedefinition im Speicher befindet, da 64 Bytes eine gerade Zahl darstellen und im Binärsystem eine gerade Potenz.
Jedes der acht Sprites hat ein Byte, das **Sprite-Pointer** genannt wird. Dieser Sprite-Pointer gibt an, wo sich die Spritedefinition im Speicher befindet. Diese acht Bytes sind stets die letzten acht Bytes vom 1K-Bereich des Bildschirmspeichers. Normalerweise bedeutet dies beim COMMODORE 64, daß Sie bei Adresse 2040 ($07F8 in HEX) beginnen. Bei Bewegung des Bildschirms verändert sich jedoch auch die Lage des Sprite-Pointers.
Jeder Sprite-Pointer kann eine Zahl zwischen 0 und 255 aufnehmen. Diese Zahl zeigt auf die Definition für das Sprite. Da jede Spritedefinition 64 Bytes benötigt, bedeutet dies, daß der Pointer auf jeden Platz im 16K-Speicherbereich zeigen kann, der für den VIC-II-Chip zugänglich ist (da 256*64=16K).

Wenn der Sprite-Pointer #0 an Adresse 2040 z. B. die Zahl 14 enthält, bedeutet dies, daß Sprite 0 mit den 64 Bytes beginnend bei Adresse 14*64 = 896 beginnt, d. h., im Kassettenpuffer. Dies wird anhand folgender Gleichung deutlich:

LOCATION = (BANK * 16384) + (SPRITE POINTER VALUE * 64)

wobei BANK einen der 16K-Speicherbereiche bezeichnet, auf die der VIC-II-Chip zugreifen kann und die von 0 bis 3 durchnumeriert sind.
Obige Gleichung gibt den Anfang der 64 Bytes des Spritedefinitionssatzes an.
Wenn der VIC-II-Chip auf BANK 0 oder BANK 2 zugreift, ist in einigen Speicherplätzen ein ROM-Image des Zeichensatzes (wie bereits erwähnt) vorhanden. Hier können keine Spritedefinitionen stehen. Werden aus irgendwelchen Gründen mehr als 128 verschiedene Spritedefinitionen benötigt, müssen Sie eine der Banks ohne ROM-Spiegelung benutzen (1 oder 3).

EINSCHALTEN DER SPRITES

Das VIC-II-Steuerregister in Adresse 53269 ($D015 HEX) ist das Sprite-Aktivierungsregister. Jedes Sprite hat in diesem Register ein Bit, das steuert, ob das Sprite EIN oder AUS ist. Das Register sieht folgendermaßen aus:

$D015 7 6 5 4 3 2 1 0

Um z. B. Sprite 1 einzuschalten, muß das entsprechende Bit gesetzt werden. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53269,PEEK(53269)OR 2
```

Folgendes ist eine mehr allgemeine Anweisung:

```
POKE 53269,PEEK(53269)OR (2↑SN)
```

wobei SN die Spritezahl von 0 bis 7 ist.

Anmerkung: Ein Sprite wird erst sichtbar, wenn es eingeschaltet wird.

AUSSCHALTEN DER SPRITES

Ein Sprite wird ausgeschaltet, indem sein Bit im VIC-II-Steuerregister bei 53269 ($D015 HEX) auf 0 gesetzt gelöscht wird. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53269, PEEK(53269)AND (255−2↑SN)
```

wobei SN die Spritezahl von 0 bis 7 angibt.

FARBEN

Ein Sprite kann eine der 16 Farben haben, die vom VIC-II-Chip erzeugt werden. Jedes Sprite hat sein eigenes Sprite-Farbregister. Die Farbregister haben folgende Adressen:

ADRESSE		BESCHREIBUNG
53287	($D027)	FARBREGISTER VON SPRITE 0
53288	($D028)	FARBREGISTER VON SPRITE 1
53289	($D029)	FARBREGISTER VON SPRITE 2
53290	($D02A)	FARBREGISTER VON SPRITE 3
53291	($D02B)	FARBREGISTER VON SPRITE 4
53292	($D02C)	FARBREGISTER VON SPRITE 5
53293	($D02D)	FARBREGISTER VON SPRITE 6
53294	($D02E)	FARBREGISTER VON SPRITE 7

Alle Punkte des Sprites werden in der Farbe angezeigt, die im Sprite-Farbregister enthalten ist. Der Rest des Sprites ist transparent, so daß die hinter diesem Sprite liegenden Werte (normalerweise der Hintergrund) angezeigt werden.

MEHRFARBENMODUS

Im Mehrfarbenmodus kann jedes Sprite max. vier verschiedene Farben haben. So wie bei den anderen Mehrfarbenmodi ist jedoch auch hier die horizontale Auflösung auf die Hälfte reduziert, d. h., beim Arbeiten im Mehrfarbenmodus (wie bei Zeichen im Mehrfarbenmodus) wird ein Sprite horizontal nicht mehr 24 Punkte, sondern in 12 Punkten ausgeführt. Jedes Punktepaar wird BITPAAR genannt. Stellen Sie sich jedes Bitpaar (Punktepaar) als einen einzelnen Punkt in Ihrem Gesamtsprite vor, wenn Sie die Farben für die Punkte in Ihren Sprites wählen.

In nachstehender Tabelle finden Sie die Bitpaar-Kombinationen, die Sie zum Einschalten der vier Farben für die Sprites benötigen:

BITPAAR	BESCHREIBUNG
00	TRANSPARENT, BILDSCHIRMFARBE
01	SPRITE-MEHRFARBENREGISTER #0 (53285) ($D025)
10	SPRITE-FARBENREGISTER
11	SPRITE-MEHRFARBENREGISTER #1 (53286) ($D026)

WÄHLEN DES MEHRFARBENMODUS FÜR EIN SPRITE

Um den Mehrfarbenmodus für ein Sprite zu wählen, müssen Sie das entsprechende VIC-II-Steuerregister in Adresse 53276 ($D01C) einschalten. Dies geschieht durch folgende POKE-Anweisung:

```
POKE 53276,PEEK(53276) OR (2↑SN)
```

wobei SN die Sprite-Nummer angibt (0 bis 7).

VERGRÖSSERTE SPRITES

Der VIC-II-Chip hat die Fähigkeit, ein Sprite in vertikaler und/oder horizontaler Richtung zu vergrößern. Bei der Ausdehnung wird jeder Punkt im Sprite zweimal so breit oder zweimal so hoch. Die Auflösung nimmt nicht zu; das Sprite wird lediglich größer.
Um ein Sprite in horizontaler Richtung zu strecken, muß das entsprechende Bit im VIC-II-Steuerregister in Adresse 53277 ($D01D HEX) eingeschaltet (auf 1 gesetzt) werden. Durch folgende POKE-Anweisung wird ein Sprite in X-Richtung vergrößert:

```
POKE 53277,PEEK(53277)OR (2↑SN)
```

wobei SN die Sprite-Nummer (0 bis 7) angibt.
Um ein Sprite in horizontaler Richtung wieder zu verkleinern, muß das entsprechende Bit im VIC-II-Steuerregister in Adresse 53277 ($D01D HEX) ausgeschaltet (auf 0 gesetzt) werden. Durch folgende POKE-Anweisung wird ein Sprite in X-Richtung wieder verkleinert:

```
POKE 53277,PEEK(53277)AND (255−2↑SN)
```

wobei SN die Sprite-Nummer von 0 bis 7 angibt.

Um ein Sprite in vertikaler Richtung zu vergrößern, muß das entsprechende Bit im VIC-II-Steuerregister in Adresse 53271 ($D017 HEX) eingeschaltet (auf 1 gesetzt) werden. Durch folgende POKE-Anweisung wird ein Sprite in Y-Richtung gestreckt:

```
POKE 53271,PEEK(53271)OR (2↑SN)
```

wobei SN die Sprite-Nummer von 0 bis 7 angibt.
Um ein Sprite in vertikaler Richtung wieder zu verkleinern, muß das entsprechende Bit im VIC-II-Steuerregister in Adresse 53271 ($D017 HEX) ausgeschaltet (auf 0 gesetzt) werden. Durch folgende POKE-Anweisung wird ein Sprite in Y-Richtung wieder verkleinert:

```
POKE 53271,PEEK(53271)AND (255−2↑SN)
```

wobei SN die Sprite-Nummer von 0 bis 7 angibt.

SPRITEPOSITIONIERUNG

Nachdem Sie ein Sprite konstruiert haben, können Sie es auf dem Bildschirm bewegen. Hierzu benutzt der COMMODORE 64 drei Positionsregister:

1) SPRITE X-POSITIONSREGISTER
2) SPRITE Y-POSITIONSREGISTER
3) HÖCHSTES BIT DES X-POSITIONSREGISTERS (engl. MSB = Most significant Bit)

Jedes Sprite hat ein X-Positionsregister, ein Y-Positionsregister und ein Bit im MSB-X-Register. Auf diese Weise können Sie die Sprites sehr genau positionieren. Hierzu stehen 512 mögliche X- und 256 mögliche Y-Positionen zur Verfügung.
Die X- und Y-Positionsregister "arbeiten" paarweise zusammen. Die Adressen von X- und Y-Register erscheinen wie folgt im Speicher: Zunächst das X-Register für Sprite 0, dann das Y-Register für das gleiche Sprite.
Danach folgt das X-Register und dann das Y-Register für Sprite 1 usw.
Nach allen 16 X- und Y-Registern kommt das höchste Bit der X-Position (X MSB) in seinem eigenen Register.

Nachstehende Tabelle gibt die Adressen der einzelnen Sprite-Positionsregister an. Sie können auf diese Adressen durch POKE-Anweisungen zugreifen:

PLATZ		BESCHREIBUNG
DEZIMAL	HEX.	
53248	($D000)	X-POSITIONSREGISTER VON SPRITE 0
53249	($D001)	Y-POSITIONSREGISTER VON SPRITE 0
53250	($D002)	X-POSITIONSREGISTER VON SPRITE 1
53251	($D003)	Y-POSITIONSREGISTER VON SPRITE 1
53252	($D004)	X-POSITIONSREGISTER VON SPRITE 2
53253	($D005)	Y-POSITIONSREGISTER VON SPRITE 2
53254	($D006)	X-POSITIONSREGISTER VON SPRITE 3
53255	($D007)	Y-POSITIONSREGISTER VON SPRITE 3
53256	($D008)	X-POSITIONSREGISTER VON SPRITE 4
53257	($D009)	Y-POSITIONSREGISTER VON SPRITE 4
53258	($D00A)	X-POSITIONSREGISTER VON SPRITE 5
53259	($D00B)	Y-POSITIONSREGISTER VON SPRITE 5
53260	($D00C)	X-POSITIONSREGISTER VON SPRITE 6
53261	($D00D)	Y-POSITIONSREGISTER VON SPRITE 6
53262	($D00E)	X-POSITIONSREGISTER VON SPRITE 7
53263	($D00F)	Y-POSITIONSREGISTER VON SPRITE 7
53264	($D010)	X MSB REGISTER

Die Position eines Sprites wird von der OBEREN LINKEN ECKE des 24-mal-21-Punktebereichs berechnet, der für ein Sprite zur Verfügung steht. Es spielt hierbei keine Rolle, wie viele bzw. wenige Punkte Sie für ein Sprite benutzt haben. Auch wenn nur ein Punkt für das Sprite benutzt wurde und dieses in der Mitte des Bildschirms stehen soll, müssen Sie für die Positionierung die obere linke Ecke als Bezugspunkt verwenden.

VERTIKALE POSITIONIERUNG

Die Positionierung in horizontaler Richtung ist etwas schwieriger als die vertikale Positionierung. Daher werden wir uns zunächst mit der vertikalen Positionierung (Y) beschäftigen.

Es gibt 200 verschiedene Punktpositionen, die auf dem Bildschirm in Y-Richtung programmiert werden können. Das Y-Positionsregister der Sprites kann Zahlen bis zu 255 fassen; d. h., Sie haben ausreichend Registerplätze, um ein Sprite nach oben und unten zu bewegen.

Ein Sprite soll jedoch auch auf dem Bildschirm erscheinen und verschwinden. Hierzu benötigen Sie mehr als 200 Werte.

Der erste Wert, bei dem ein Sprite von oben auf dem Bildschirm auftaucht und der für ein in Y-Richtung unvergrößertes Sprite gilt, ist 30. Für ein in Y-Richtung gestrecktes Sprite lautet dieser Wert 9 (da jeder Punkt zweimal so hoch ist und die Ausgangsposition auch hier von der obersten linken Ecke des Sprites berechnet wird, kann hier der entsprechende Wert kleiner sein).
Der erste Y-Wert, bei dem ein Sprite (vergrößert oder nicht) ganz auf dem Bildschirm erscheint (alle 21 möglichen Zeilen werden angezeigt), lautet 50.
Der letzte Y-Wert, bei dem ein unvergrößertes Sprite noch ganz auf dem Bildschirm vorhanden ist, ist 229. Der letzte Y-Wert, bei dem ein vergrößertes Sprite noch ganz auf dem Bildschirm erscheint, lautet 208.
Der erste Y-Wert, bei dem ein Sprite vollständig vom Bildschirm verschwunden ist, ist 250.

BEISPIEL:

SHIFT CLR/HOME

```
10 PRINT"□"                          :REM CLEAR SCREEN
20 POKE2040,13                       :REM GET SPRITE 0
DATA FROM BLOCK 13
30 FORI=0TO62:POKE832+I,129:NEXT:REM POKE SPRITE
DATA INTO BLOCK 13 (13*64=832)
40 V=53248                           :REM SET BEGINNING
OF VIDEO CHIP
50 POKEV+21,1                        :REM ENABLE SPRITE
1
60 POKEV+39,1                        :REM SET SPRITE 0
COLOR
70 POKEV+1,100                       :REM SET SPRITE 0
Y POSITION
80 POKEV+16,0:POKEV,100              :REM SET SPRITE 0
X POSITION
```

HORIZONTALE POSITIONIERUNG

Positionierung in horizontaler Richtung ist komplizierter, da hier mehr als 256 Positionen zur Verfügung stehen. D. h., ein Extrabit oder neuntes Bit zur Steuerung der X-Position wird benötigt. Durch Hinzunahme des Extrabits hat ein Sprite nun 512 mögliche Positionen in der X-Richtung (links/rechts). Hierdurch stehen mehr Positionen zur Verfügung, als auf dem Bildschirm angezeigt werden können. Jedes Sprite kann eine Position von 0 bis 511 haben. Es sind jedoch lediglich die Werte zwischen 24 und 343 auf dem Bildschirm sichtbar. Wenn die X-Position eines Sprites größer als 255 (auf der rechten Bildschirmseite) ist, muß das entsprechende Bit im MSB-Register auf 1 gesetzt (eingeschaltet) sein. Wenn die X-Position eines Sprites kleiner als 256 (z. B. auf der linken Bildschirmseite) ist, dann muß das X MSB-Register dieses Sprites auf 0 gesetzt (ausgeschaltet) sein.

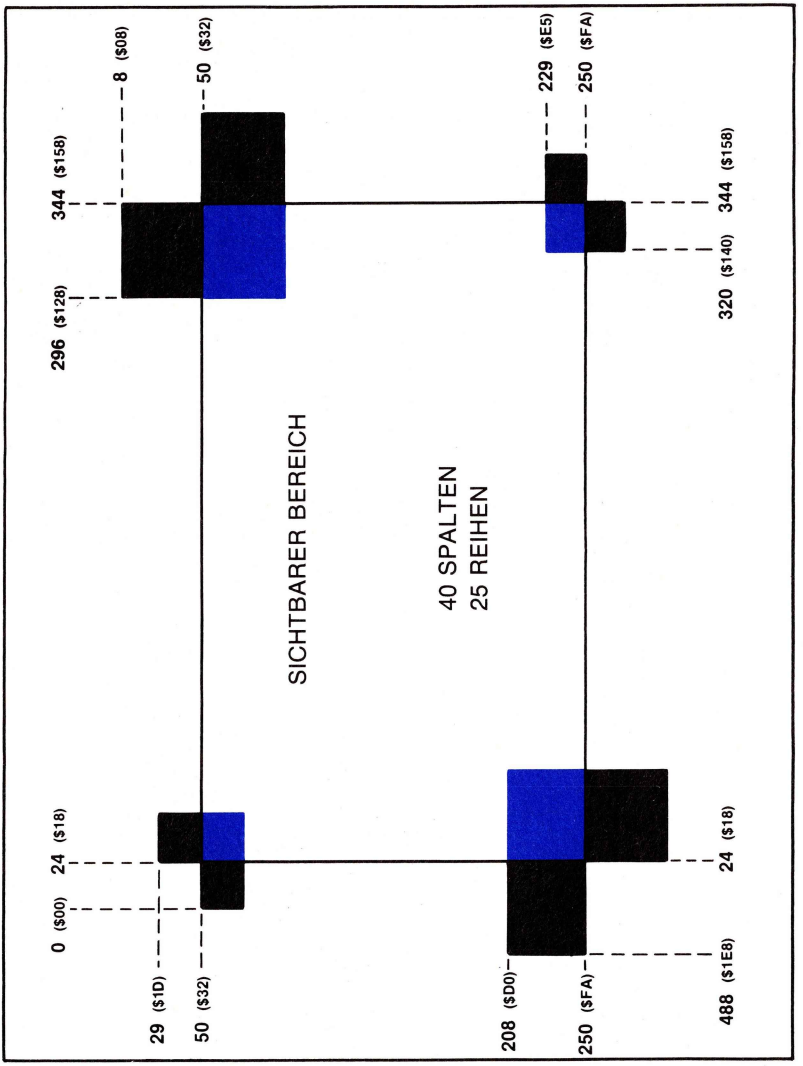
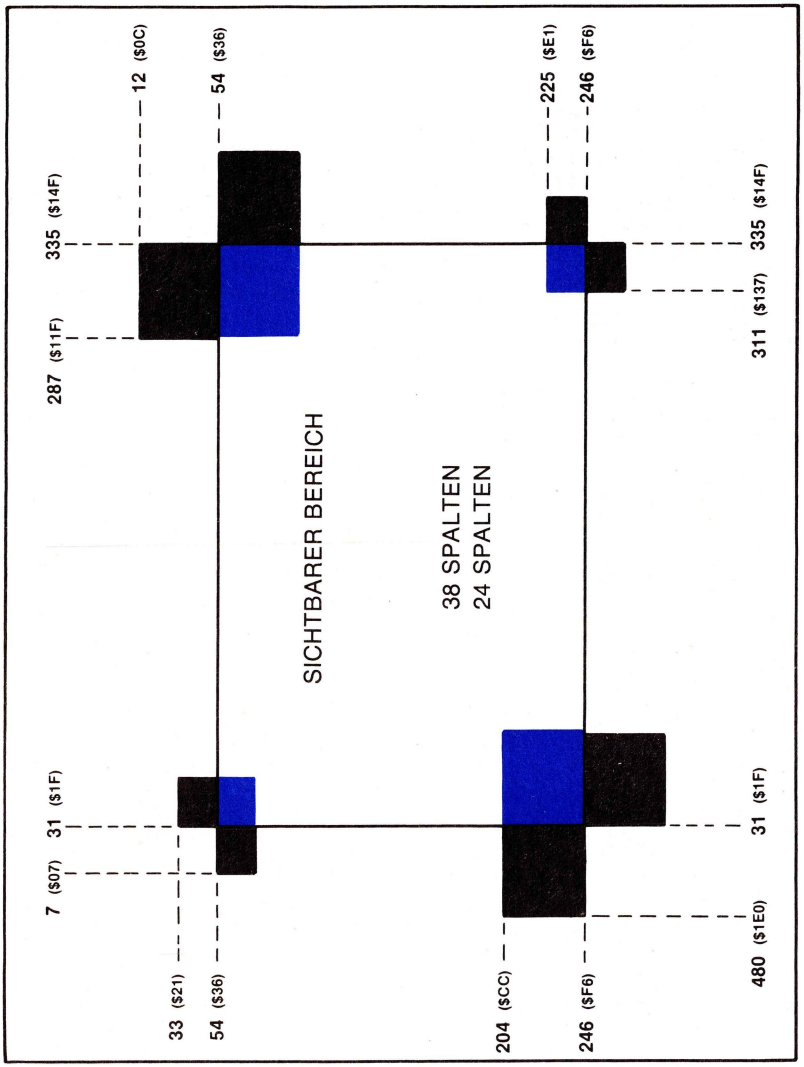


Abb. 3.3. Sprite



*North American television transmission standards for your home TV.

Positioniertabelle

Die Bits 0 bis 7 vom X MSB-Register entsprechen den Sprites 0 bis 7. Durch folgendes Programm wird ein Sprite über den Bildschirm bewegt:

BEISPIEL:

```
10 PRINT"SHIFT CLR/HOME"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV+1,100
80 FORJ=0TO347
90 HX=INT(J/256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX:NEXT
```

Beim Bewegen von vergrößerten Sprites auf die linke Bildschirmseite in X-Richtung soll zu Beginn der Bewegung das Sprite auf der rechten Seite nicht sichtbar sein. Ein erweitertes Sprite ist nämlich größer als der verfügbare Platz auf der linken Bildschirmseite.

BEISPIEL:

```
10 PRINT"SHIFT CLR/HOME"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1:POKEV+23,1:POKEV+29,1
70 POKEV+1,100
80 J=488
90 HX=INT(J/256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX
110 J=J+1:IFJ>511THENJ=0
120 IFJ>488ORJ<348GOTO90
```

Die Tabellen in Abbildung 3.3. erklären die Spritepositionierung. Damit können Sie jedes Sprite beliebig positionieren. Durch Bewegung eines Sprites um jeweils eine einzelne Punktposition wird eine runde freie Bewegung möglich.

ZUSAMMENFASSUNG ÜBER DIE SPRITEPOSITIONIERUNG

Unvergrößerte Sprites sind im 40-Spalten-mal-25-Reihen-Modus innerhalb folgender Parameter zumindest teilweise sichtbar:

$$1 <= X <= 343$$

$$30 <= Y <= 249$$

Im 38-Spalten-Modus ändern sich die X-Parameter wie folgt:

$$8 <= X <= 334$$

Im 24-Reihen-Modus ändern sich die Y-Parameter wie folgt:

$$34 <= Y <= 245$$

Vergrößerte Sprites sind innerhalb folgender Parameter im 40-Spalten-mal-25-Reihen-Modus sichtbar:

$$489 >= X <= 343$$

$$9 >= Y <= 249$$

Im 38-Spalten-Modus ändern sich die X-Parameter wie folgt:

$$496 >= X <= 334$$

Im 24-Reihen-Modus ändern sich die Y-Parameter wie folgt:

$$13 <= Y <= 245$$

SPRITE-ANZEIGEPRIORITÄTEN

Die Wege der verschiedenen Sprites können sich kreuzen. Darüber hinaus können sich Sprites vor oder hinter anderen Objekten auf dem Bildschirm bewegen. Durch diese räumliche Darstellung können Sie bei Spielen einen dreidimensionalen Effekt erzeugen. Die Priorität zwischen den einzelnen Sprites ist festgelegt. Sprite 0 hat dabei die oberste, Sprite 1 die nächste Priorität usw., so daß Sprite 7 entsprechend die niedrigste Priorität hat. D. h., wenn Sprite 1 und Sprite 6 einander kreuzen, so erscheint Sprite 1 vor Sprite 6.
Soll ein Sprite also im Bildvordergrund erscheinen, so muß es eine niedrigere Zahl erhalten als das, das im Hintergrund erscheinen soll.

Anmerkung: Ein "Fenstereffekt" ist möglich. Hat ein Sprite mit höherer Priorität "Löcher" (Bereiche, in denen die Punkte nicht auf 1 eingeschaltet sind), so scheinen Sprites mit niedrigerer Priorität durch. Das gleiche gilt für Sprites und Hintergrunddaten.

Das Prioritätsverhältnis zwischen Sprites und Bildschirmhintergrund wird durch das entsprechende Register in Adresse 53275 ($D01B) gesteuert. In diesem Register hat jedes Sprite ein Bit. Ist das Bit 0, so hat das entsprechende Sprite eine höhere Priorität als der Bildschirmhintergrund; d. h., das Sprite erscheint vor den Hintergrunddaten. Ist das Bit 1, so hat der Hintergrund Priorität gegenüber dem Sprite. Dieses erscheint dann hinter den Hintergrunddaten.

KOLLISIONSERKENNUNG

Einer der interessanteren Aspekte des VIC-II-Chips ist die Möglichkeit der Kollisionserkennung. Kollisionen können zwischen den verschiedenen Sprites oder Sprite und einem bestimmten Hintergrund erkannt werden. Zu einer Kollision kommt es, wenn ein "nicht-0"-Teil eines Sprites einen "nicht-0"-Teil eines weiteren Sprites oder eines Bildschirmzeichens überlappt.

KOLLISION ZWISCHEN EINZELNEN SPRITES

Eine Kollision zwischen einzelnen Sprites wird vom Computer erkannt oder im entsprechenden Register an Adresse 53278 ($D01E HEX) im VIC-II-Chip-Steuerregister gekennzeichnet. In diesem Register hat jedes Sprite ein Bit. Ist dieses Bit 1, dann ist das Sprite an einer Kollision beteiligt. Die Bits in diesem Register bleiben bis zum Lesen (PEEK-Anweisung) gesetzt. Nach dem Lesen wird das Register automatisch gelöscht. Der Wert sollte daher besser in einer Variablen gespeichert werden, bis er verarbeitet wird.

Anmerkung: Kollisionen können auch dann auftreten, wenn Sprites ausgeschaltet sind.

KOLLISION ZWISCHEN SPRITES UND DATEN

Eine Kollision zwischen einem Sprite und Daten wird im entsprechenden Register in Adresse 53279 ($D01F HEX) des VIC-II-Chip-Steuerregisters festgestellt. In diesem Register hat jedes Sprite ein Bit. Ist dieses Bit eine 1, dann ist dieses Sprite an einer Kollision beteiligt. Die Bits in diesem Register bleiben bis zum Lesen (PEEK-Anweisung) gesetzt. Nach dem Lesen wird das Register automatisch gelöscht. Der Wert sollte daher in einer Variablen gespeichert werden, bis er verarbeitet wird.

Anmerkung: Der MULTI-COLOR-Wert 01 wird bei Kollisionen als transparent angesehen, auch wenn er auf dem Bildschirm sichtbar ist. Beim Erstellen eines Hintergrunds sollte daher all das, was nicht zu einer Kollision führen darf, im Mehrfarbenmodus auf 01 gesetzt werden.

```
10 REM SPRITE EXAMPLE 1...
20 REM THE HOT AIR BALLOON
30 VIC=13*4096:REM THIS IS WHERE THE VIC REGISTERS
BEGIN
35 POKEVIC+21,1:REM ENABLE SPRITE 0
36 POKEVIC+33,14:REM SET BACKGROUND COLOR TO LIGHT
BLUE
37 POKEVIC+23,1:REM EXPAND SPRITE 0 IN Y
38 POKEVIC+29,1:REM EXPAND SPRITE 0 IN X
40 POKE2040,192:REM SET SPRITE 0'S POINTER
180 POKEVIC+0,100:REM SET SPRITE 0'S X POSITION
190 POKEVIC+1,100:REM SET SPRITE 0'S Y POSITION
220 POKEVIC+39,1:REM SET SPRITE 0'S COLOR
250 FORY=0TO63:REM BYTE COUNTER WITH SPRITE LOOP
300 READA:REM READ IN A BYTE
310 POKE192*64+Y,A:REM STORE THE DATA IN SPRITE
AREA
320 NEXTY:REM CLOSE LOOP
330 DX=1:DY=1
340 X=PEEK(VIC):REM LOOK AT SPRITE 0'S X POSITION
350 Y=PEEK(VIC+1):REM LOOK AT SPRITE 0'S Y POSITION
360 IFY=50ORY=208THENDY=-DY:REM IF Y IS ON THE
EDGE OF THE....
370 REM SCREEN, THEN REVERSE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM IF
SPRITE IS....
390 REM TOUCHING THE LEFT EDGE (X=24 AND THE MSB
FOR SPRITE 0 IS 0), REVERSE IT
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM IF
SPRITE IS....
410 REM TOUCHING THE RIGHT EDGE (X=40 AND THE MSB
FOR SPRITE 0 IS 1), REVERSE IT
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM SWITCH TO OTHER SIDE OF THE SCREEN
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SWITCH TO OTHER SIDE OF THE SCREEN
460 X=X+DX:REM ADD DELTA X TO X
470 X=XAND255:REM MAKE SURE X IS IN ALLOWED RANGE
480 Y=Y+DY:REM ADD DELTA Y TO Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM PUT NEW X VALUE INTO SPRITE 0'S
X POSITION
510 POKEVIC+1,Y:REM PUT NEW Y VALUE INTO SPRITE
0'S Y POSITION
530 GOTO340
600 REM ***** SPRITE DATA *****
610 DATA0,127,0,1,255,192,3,255,224,3,231,224
620 DATA7,217,240,7,223,240,7,217,240,3,231,224
630 DATA3,255,224,3,255,224,2,255,160,1,127,64
640 DATA1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
650 DATA0,62,0,0,62,0,0,62,0,0,28,0,0
```

```
10 REM SPRITE EXAMPLE 2...
20 REM THE HOT AIR BALLOON AGAIN
30 VIC=13*4096:REM THIS IS WHERE THE VIC REGISTERS
BEGIN
35 POKEVIC+21,63:REM ENABLE SPRITES 0 THRU 5
36 POKEVIC+33,14:REM SET BACKGROUND COLOR TO LIGHT
BLUE
37 POKEVIC+23,3:REM EXPAND SPRITES 0 AND 1 IN Y
38 POKEVIC+29,3:REM EXPAND SPRITES 0 AND 1 IN X
40 POKE2040,192:REM SET SPRITE 0'S POINTER
50 POKE2041,193:REM SET SPRITE 1'S POINTER
60 POKE2042,192:REM SET SPRITE 2'S POINTER
70 POKE2043,193:REM SET SPRITE 3'S POINTER
80 POKE2044,192:REM SET SPRITE 4'S POINTER
90 POKE2045,193:REM SET SPRITE 5'S POINTER
100 POKEVIC+4,30:REM SET SPRITE 2'S X POSITION
110 POKEVIC+5,58:REM SET SPRITE 2'S Y POSITION
120 POKEVIC+6,65:REM SET SPRITE 3'S X POSITION
130 POKEVIC+7,58:REM SET SPRITE 3'S Y POSITION
140 POKEVIC+8,100:REM SET SPRITE 4'S X POSITION
150 POKEVIC+9,58:REM SET SPRITE 4'S Y POSITION
160 POKEVIC+10,100:REM SET SPRITE 5'S X POSITION
170 POKEVIC+11,58:REM SET SPRITE 5'S Y POSITION
175 PRINT"■"TAB(15)"THIS IS TWO HIRES SPRITES";
176 PRINTTAB(55)"ON TOP OF EACH OTHER"
180 POKEVIC+0,100:REM SET SPRITE 0'S X POSITION
190 POKEVIC+1,100:REM SET SPRITE 0'S Y POSITION
200 POKEVIC+2,100:REM SET SPRITE 1'S X POSITION
210 POKEVIC+3,100:REM SET SPRITE 1'S Y POSITION
220 POKEVIC+39,1:REM SET SPRITE 0'S COLOR
230 POKEVIC+41,1:REM SET SPRITE 2'S COLOR
240 POKEVIC+43,1:REM SET SPRITE 4'S COLOR
250 POKEVIC+40,6:REM SET SPRITE 1'S COLOR
260 POKEVIC+42,6:REM SET SPRITE 3'S COLOR
270 POKEVIC+44,6:REM SET SPRITE 5'S COLOR
280 FORX=192TO193:REM THE START OF THE LOOP THAT
DEFINES THE SPRITES
290 FORY=0TO63:REM BYTE COUNTER WITH SPRITE LOOP
300 READA:REM READ IN A BYTE
310 POKEX*64+Y,A:REM STORE THE DATA IN SPRITE AREA
320 NEXTY,X:REM CLOSE LOOPS
330 DX=1:DY=1
340 X=PEEK(VIC):REM LOOK AT SPRITE 0'S X POSITION
350 Y=PEEK(VIC+1):REM LOOK AT SPRITE 0'S Y POSITION
360 IFY=50ORY=208THENDY=-DY:REM IF Y IS ON THE
EDGE OF THE...
370 REM SCREEN, THEN REVERSE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM IF
SPRITE IS...
390 REM TOUCHING THE LEFT EDGE, THEN REVERSE IT
```

CTRL 2

SHIFT CLR/HOME

```
400 IFX=40ANDCPEEK(VIC+16)AND1)=1THENDX=-DX:REM IF
SPRITE IS...
410 REM TOUCHING THE RIGHT EDGE, THEN REVERSE IT
420 IFX=255ANDDX=1THENX=-1:SIDE=3
430 REM SWITCH TO OTHER SIDE OF THE SCREEN
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SWITCH TO OTHER SIDE OF THE SCREEN
460 X=X+DX:REM ADD DELTA X TO X
470 X=XAND255:REM MAKE SURE X IS IN ALLOWED RANGE
480 Y=Y+DY:REM ADD DELTA Y TO Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM PUT NEW X VALUE INTO SPRITE 0'S
X POSITION
500 POKEVIC+2,X:REM PUT NEW X VALUE INTO SPRITE
1'S X POSITION
510 POKEVIC+1,Y:REM PUT NEW Y VALUE INTO SPRITE
0'S Y POSITION
520 POKEVIC+3,Y:REM PUT NEW Y VALUE INTO SPRITE
1'S Y POSITION
530 GOTO340
600 REM ***** SPRITE DATA *****
610 DATA0,255,0,3,153,192,7,24,224,7,56,224,14,126,
112,14,126,112,14,126,112
620 DATA6,126,96,7,56,224,7,56,224,1,56,128,0,153,
0,0,90,0,0,56,0
630 DATA0,56,0,0,0,0,0,0,0,0,126,0,0,42,0,0,84,0,0,
40,0,0
640 DATA0,0,0,0,102,0,0,231,0,0,195,0,1,129,128,1,
129,128,1,129,128
650 DATA1,129,128,0,195,0,0,195,0,4,195,32,2,102,
64,2,36,64,1,0,128
660 DATA1,0,128,0,153,0,0,153,0,0,0,0,0,84,0,0,42,
0,0,20,0,0
```

```
10 REM SPRITE EXAMPLE 3...
20 REM THE HOT AIR GORF
30 VIC=53248:REM THIS IS WHERE THE VIC REGISTERS
BEGIN
35 POKEVIC+21,1:REM ENABLE SPRITE 0
```

```
36 POKEVIC+33,14:REM SET BACKGROUND COLOR TO LIGHT
BLUE
37 POKEVIC+23,1:REM EXPAND SPRITE 0 IN Y
38 POKEVIC+29,1:REM EXPAND SPRITE 0 IN X
40 POKE2040,192:REM SET SPRITE 0'S POINTER
50 POKEVIC+28,1:REM TURN ON MULTICOLOR
60 POKEVIC+37,7:REM SET MULTICOLOR 0
70 POKEVIC+38,4:REM SET MULTICOLOR 1
180 POKEVIC+0,100:REM SET SPRITE 0'S X POSITION
190 POKEVIC+1,100:REM SET SPRITE 0'S Y POSITION
220 POKEVIC+39,2:REM SET SPRITE 0'S COLOR
290 FORY=0TO63:REM BYTE COUNTER WITH SPRITE LOOP
300 READA:REM READ IN A BYTE
310 POKE12288+Y,A:REM STORE THE DATA IN SPRITE AREA
320 NEXT Y:REM CLOSE LOOP
330 DX=1:DY=1
340 X=PEEK(VIC):REM LOOK AT SPRITE 0'S X POSITION
350 Y=PEEK(VIC+1):REM LOOK AT SPRITE 0'S Y POSITION
360 IFY=50ORY=208THENDY=-DY:REM IF Y IS ON THE
EDGE OF THE...
370 REM SCREEN, THEN REVERSE DELTA Y
380 IF X=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM
IF SPRITE IS...
390 REM TOUCHING THE LEFT EDGE, THEN REVERSE IT
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM IF
SPRITE IS...
410 REM TOUCHING THE RIGHT EDGE, THEN REVERSE IT
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM SWITCH TO OTHER SIDE OF THE SCREEN
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SWITCH TO OTHER SIDE OF THE SCREEN
460 X=X+DX:REM ADD DELTA X TO X
470 X=XAND255:REM MAKE SURE X IS IN ALLOWED RANGE
480 Y=Y+DY:REM ADD DELTA Y TO Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM PUT NEW X VALUE INTO SPRITE 0'S
X POSITION
510 POKEVIC+1,Y:REM PUT NEW Y VALUE INTO SPRITE
0'S Y POSITION
520 GETA$:REM GET A KEY FROM THE KEYBOARD
521 IFA$="M"THENPOKEVIC+28,1:REM USER SELECTED
MULTICOLOR
522 IFA$="H"THENPOKEVIC+28,0:REM USER SELECTED
HIGH RESOLUTION
530 GOTO340
600 REM ***** SPRITE DATA *****
610 DATA64,0,1,16,170,4,6,170,144,10,170,160,42,
170,168,41,105,104,169,235,106
620 DATA169,235,106,169,235,106,170,170,170,170,
170,170,170,170,170,170,170,170
630 DATA166,170,154,169,85,106,170,85,170,42,170,
168,10,170,160,1,0,64,1,0,64
640 DATA5,0,80,0
```

WEITERE GRAPHIKMÖGLICHKEITEN

WEGBLENDEN DES BILDSCHIRMS

Über Bit 4 des VIC-II-Steuerregisters wird das Wegblenden des Bildschirms gesteuert. Es befindet sich im Steuerregister an Adresse 53265 ($D011). Ist dieses Bit eingeschaltet (d. h. auf 1 gesetzt), dann ist der Bildschirm normal. Ist Bit 4 auf 0 gesetzt (AUS), dann nimmt der gesamte Bildschirm die Rahmenfarbe an.
Durch folgende POKE-Anweisung wird der Bildschirm weggeblendet. Die Daten gehen nicht verloren, sie werden lediglich nicht mehr angezeigt.

```
POKE 53265,PEEK(53265)AND 239
```

Zur Rücksetzung des Bildschirms dient folgende POKE-Anweisung:

```
POKE 53265,PEEK(53265)OR 16
```

Anmerkung: Durch Ausschalten des Bildschirms wird der Prozessor etwas beschleunigt, d. h., auch die Programmausführung erfolgt etwas schneller.

RASTERREGISTER

Das Rasterregister befindet sich im VIC-II-Chip an Adresse 53266 ($D012). Das Rasterregister hat einen doppelten Zweck. Beim Lesen des Registers werden die unteren 8 Bits der derzeitigen Rasterposition wiedergegeben. Die Rasterposition des signifikantesten Bits ist im Registerplatz 53265 ($D011). Sie können das Rasterregister benutzen, um das Bildschirmflackern zu reduzieren. Änderungen der Bildschirmanzeige sollen vorgenommen werden, wenn das Raster nicht im sichtbaren Anzeigebereich liegt, d. h., wenn die Punktpositionen zwischen 51 und 251 liegen.
Nach dem Zuordnen des Rasterregisters (einschl. MSB) wird die zugeordnete Zahl für den Rastervergleich gespeichert. Ist der tatsächliche Rasterwert gleich der Zahl des Rasterregisters, so wird ein Bit im VIC-II-Chip-Interrupt-Register 53273 ($D019) auf 1 gesetzt (EIN).

Anmerkung: Wird das richtige Interrupt-Bit wirksam auf 1 gesetzt, so kommt es zu einem Interrupt (IRQ).

INTERRUPT-STATUSREGISTER

Das Interrupt-Statusregister zeigt den derzeitigen Status einer beliebigen Interruptmöglichkeit. Der derzeitige Status von Bit 2 des Interrupt-Registers ist eine 1, wenn

es zu einer Kollision zwischen zwei Sprites kommt. Das gleiche gilt in 1 : 1-Entsprechung für die in nachstehender Tabelle aufgeführten Bits 0 bis 3. Auch Bit 7 wird bei einem Interrupt auf 1 gesetzt.
Das Interrupt-Statusregister befindet sich an Speicherplatz 53273 ($D019) und sieht wie folgt aus:

SCHALTER	BIT#	BESCHREIBUNG
IRST	0	Gesetzt, wenn derzeitiger Rasterwert gleich gespeichertem Rasterwert.
IMDC	1	Gesetzt durch eine Kollision zwischen Sprite und einem Zeichen auf dem Bildschirm, Zurückstellung durch RESET.
IMMC	2	Gesetzt durch eine Kollision zwischen zwei Sprites, Zurückstellung durch RESET.
ILP	3	Gesetzt bei negativer Flanke am Lightpen-Eingang.
IRQ	7	Wird gesetzt, wenn eines der Bits #0 bis 3 gesetzt ist.

Nach dem Setzen eines Interrupt-Bits ist dieses "latched" und muß durch Schreiben einer 1 für dieses Bit im Interrupt-Register gelöscht werden (= RESET).
Hierdurch kann der Interrupt selektiv ohne die Speicherung der anderen Interrupt-Bits gehandhabt werden.
Das **INTERRUPT-AKTIVIERUNGSREGISTER** befindet sich in Adresse 53274 ($D01A). Dieses Register hat das gleiche Format wie das Interrupt-Statusregister. Wenn das entsprechende Bit im Interrupt-Aktivierungsregister nicht auf 1 gesetzt ist, wird von dieser Quelle kein Interrupt angefordert. Das Interrupt-Statusregister kann noch immer abgerufen werden, es werden jedoch keine Interrupts erzeugt.
Um eine Interrupt-Anforderung wirksam zu machen, muß das entsprechende Interrupt-Aktivierungsbit (wie in obiger Tabelle gezeigt) auf 1 gesetzt sein.
Über diese Interrupt-Struktur können Betriebsarten mit geteiltem Bildschirm benutzt werden. So kann z. B. für die eine Hälfte des Bildschirms Bit-Mapping, für eine Hälfte Text, mehr als 8 Sprites gleichzeitig usw. benutzt werden. Die Interrupts müssen nur richtig gehandhabt werden. Soll die obere Bildschirmhälfte z. B. im Bit-Mapping und die untere mit Text dargestellt werden, muß lediglich das Raster-Vergleichsregister (wie bereits erklärt) für die untere Bildschirmhälfte gesetzt sein. Bei einem Interrupt muß der VIC-II-Chip die Zeichen aus dem ROM nehmen; dann wird das Raster-Vergleichsregister für einen Interrupt am oberen Bildschirmrand eingestellt. Wenn es dort zu einem Interrupt kommt, muß der VIC-II-Chip die Zeichen aus dem RAM (Bit-Mapping) nehmen.
Auf die gleiche Weise können auch mehr als 8 Sprites angezeigt werden. Hierzu ist BASIC jedoch leider nicht schnell genug. Beim Arbeiten mit Anzeigeinterrupts sollten Sie also die Maschinensprache wählen.

VORSCHLÄGE FÜR BILDSCHIRM-ZEICHENFARBE-KOMBINATIONEN

Bei Farbfernsehgeräten gibt es Einschränkungen hinsichtlich der Fähigkeit, bestimmte Farben nebeneinander anzuzeigen. Bestimmte Kombinationen von Bildschirm und Zeichenfarben führen zu unscharfen Bildern. Die nachstehende Tabelle zeigt Ihnen, welche Farbkombinationen Sie besser vermeiden sollten und welche Farben gut miteinander kombiniert werden können.

ZEICHENFARBE

BILDSCHIRMFARBE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	●	X	●	●	●	X	●	●	X	●	●	●	●	●	●
1	●	X	●	X	●	●	●	X	●	●	●	●	●	X	●	●
2	X	●	X	X	●	X	X	●	●	X	●	X	X	X	X	●
3	●	X	X	X	X	●	●	X	X	X	X	●	X	X	●	X
4	●	●	X	X	X	X	X	X	X	X	X	X	X	X	X	●
5	●	●	X	●	X	X	X	X	X	X	X	●	X	●	X	●
6	●	●	X	●	X	X	X	X	X	X	X	X	X	●	●	●
7	●	X	●	X	X	X	●	X	●	●	●	●	●	X	X	X
8	●	●	●	X	X	X	X	●	X	●	X	X	X	X	X	●
9	X	●	X	X	X	X	X	●	●	X	●	X	X	X	X	●
10	●	●	●	X	X	X	X	●	X	●	X	X	X	X	X	●
11	●	●	X	●	X	X	X	●	X	X	X	X	●	●	●	●
12	●	●	●	X	X	X	●	X	X	●	X	●	X	X	X	●
13	●	X	X	X	X	●	●	X	X	X	X	●	X	X	X	X
14	●	●	X	●	X	X	●	X	X	X	X	●	X	X	X	●
15	●	●	●	X	●	●	●	X	X	●	●	●	●	X	●	X

● = GUT
● = ANNEHMBAR
X = SCHLECHT

PROGRAMMIEREN VON SPRITES – EIN ANDERER ASPEKT

Allen, die Schwierigkeiten mit Graphiken haben, werden Sprites in diesem Kapitel auf etwas einfachere Weise erklärt.

PROGRAMMIERUNG DER SPRITES IN BASIC – EIN KURZES PROGRAMM

Es gibt mindestens drei verschiedene BASIC-Programmiertechniken zum Erstellen von Graphikbildern und Zeichentrickfilmen mit dem COMMODORE 64. Sie können den computereigenen Graphikzeichensatz (siehe Seite 376) benutzen. Sie können in Ihren eigenen Zeichen (siehe Seite 108) programmieren oder, die beste Möglichkeit . . . die in den Computer eingebauten "Sprite-Graphiken" benutzen.
Damit Sie sehen, wie einfach dies ist, zeigen wir Ihnen hier ein kurzes Programm für die Erstellung von Sprites in BASIC:




```
10 PRINT"□"
20 POKE2040,13
30 FORS=832TO832+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV,24
80 POKEV+1,100
```

Dieses Programm enthält die Hauptbestandteile, die Sie beim Einstellen von Sprites benötigen. Die POKE-Zahlen stammen aus der Spritetabelle auf Seite 176. Dieses Programm definiert das erste Sprite – Sprite 0 – als weißes Quadrat auf dem Bildschirm. Wir wollen das Programm nun Zeile für Zeile erklären:

ZEILE 10 löscht den Bildschirm.

ZEILE 20 setzt den "Sprite-Zeiger" auf die Speicherstelle, aus der der COMMODORE 64 die Spritedaten lesen soll. Sprite 0 wird auf 2040, Sprite 1 auf 2041, Sprite 2 auf 2042 usw. und Sprite 7 auf 2047 gesetzt. Durch Verwendung der nachfolgenden Zeile anstelle von Zeile 20 können alle 8 Sprite-Zeiger auf 13 gesetzt werden:

```
FOR SP=2040TO2047:POKE SP,13:NEXT SP
```

ZEILE 30 schreibt das erste Sprite (Sprite 0) in 63 Bytes des RAM-Speichers des COMMODORE 64 beginnend bei Adresse 832 (jedes Sprite benötigt 63 Bytes des

Speichers). Das erste Sprite (Sprite 0) wird in den Speicherplätzen 832 bis 894 abgespeichert.

ZEILE 40 setzt die Variable "V" gleich 53248, der Startadresse des VIDEO-CHIPS. Durch diese Eingabe können wir die Formel (V + Zahl) für Spriteeingaben benutzen. Wir benutzen diese Formel beim POKEn von Spriteeingaben, da sie Speicherkapazität einspart und das Arbeiten mit kleineren Zahlen ermöglicht. So haben wir z. B. in Zeile 50 POKE V + 21 eingegeben. Dies entspricht der Eingabe von POKE 53248 + 21 oder 53269. V + 21 benötigt jedoch weniger Platz als 53269 und läßt sich leichter merken.

ZEILE 50 aktiviert Sprite 0. Es gibt 8 Sprites mit der Zahl 0 bis 7. Zum Einschalten der einzelnen Sprites oder einer Kombination von Sprites müssen Sie lediglich POKE V + 21 gefolgt von einer Zahl zwischen 0 (Ausschalten aller Sprites) und 255 (Einschalten aller 8 Sprites) eingeben. Durch das POKEn folgender Zahlen können ein oder mehrere Sprites eingeschaltet werden:

ALL ON	SPRITE0	SPRITE1	SPRITE2	SPRITE3	SPRITE4	SPRITE5	SPRITE6	SPRITE7	ALL OFF
V+21,255	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128	V+21,0

Durch POKE V + 21,1 wird Sprite 0 eingeschaltet. POKE V + 21,128 schaltet Sprite 7 ein. Es kann auch eine Spritekombination eingeschaltet werden. So wird z. B. durch POKE V + 21,129 sowohl Sprite 0 als auch Sprite 7 durch Addition der beiden Einschaltzahlen (1 + 128) eingeschaltet. (Siehe Spritetabelle, Seite 176.)

ZEILE 60 legt die Farbe von Sprite 0 fest. Es gibt 16 mögliche Spritefarben, die von 0 (Schwarz) bis 15 (Grau) numeriert sind. Jedes Sprite benötigt für die Farbe eine unterschiedliche POKE-Anweisung von V + 39 bis V + 46. POKE V + 39,1 gibt Sprite 0 die Farbe Weiß. Durch POKE V + 46,15 erhält Sprite 7 die Farbe Grau (bezüglich weiterer Einzelheiten siehe Spritetabelle).
Beim Erstellen eines Sprites bleibt dieses so lange im Speicher erhalten, bis es neu definiert oder der Computer abgeschaltet wird. Auf diese Weise kann Farbe, Position und Form des Sprites im Direktmodus geändert werden. Dies ist besonders sinnvoll beim Editieren.
Führen Sie z. B. obiges Programm aus und geben Sie danach diese Zeile im Direktmodus (ohne Zeilennummer) ein. Danach drücken Sie die Taste RETURN :

POKE V+39,8

Das Sprite auf dem Bildschirm ist nun ORANGE. Versuchen Sie das POKEn einer anderen Zahl zwischen 0 und 15. Es wird eine andere Spritefarbe erscheinen. Da

dies im Direktmodus erfolgte, wird das Sprite beim Ausführen des Programms wieder die ursprüngliche Farbe (Weiß) haben.

ZEILE 70 bestimmt die horizontale oder "X"-Position des Sprites auf dem Bildschirm. Diese Zahl legt die Position der OBEREN LINKEN ECKE des Sprites fest. Die externe linke Position, die Sie auf dem Bildschirm sehen können, ist die Position Nr. 24, auch wenn Sie das Sprite über den Bildschirmrand hinaus auf Position Nr. 0 bewegen können.

ZEILE 80 bestimmt die vertikale oder "Y"-Position des Sprites. In diesem Programm plazierten wir das Sprite an die X-Position 24 und Y-Position 100. Probieren Sie eine andere Position aus. Geben Sie dazu folgende POKE-Anweisung im Direktmodus ein und drücken Sie danach RETURN :

POKE V,24:POKE V+1,50

Hierdurch wird das Sprite in die obere linke Bildschirmecke gesetzt. Um das Sprite in die untere linke Ecke zu bewegen, geben Sie folgendes ein:

POKE V,24:POKE V+1,229

Jede Zahl von 832 bis 895 im Speicherbereich von Sprite 0 repräsentiert einen Satz von 8 Pixel, wobei drei 8-Pixel-Sätze eine horizontale Reihe des Sprites darstellen. Die Schleife in Zeile 30 gibt dem Computer die Anweisung POKE 832,255, wodurch die ersten 8 Pixel "ausgefüllt" werden, und danach werden durch POKE 833,255 die nächsten 8 Pixel ebenfalls "ausgefüllt" usw. bis zu Adresse 894, der die letzte Gruppe von 8 Pixel in der unteren rechten Spriteecke angibt.
Damit Sie besser sehen, wie dies funktioniert, versuchen Sie folgendes im Direktbetrieb und beachten Sie, daß die zweite Gruppe der 8 Pixel gelöscht wird:

POKE 833,0 (zum Zurücksetzen über die Tastatur POKE 833,255 oder RUN eingeben)

Durch folgende Zeile, die Sie in Ihr Programm aufnehmen können, wird die Mitte des erstellten Sprites gelöscht:

```
90 FOR A=836 TO 891 STEP 3:POKE A,0:NEXT A
```

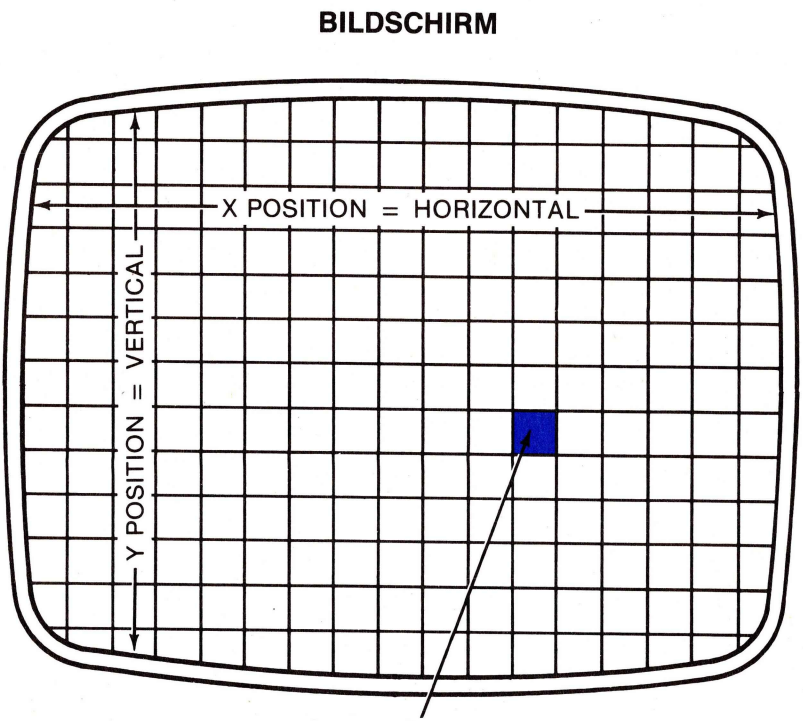
Denken Sie daran, daß die Pixel, aus denen die Sprites aufgebaut sind, in Sätzen von acht gruppiert sind. Diese Zeile löscht die 5. Gruppe von 8 Pixel (Satz 836) und jeden dritten Satz bis zu Satz 890. Versuchen Sie, andere Zahlen in die Adressen von 832 bis 894 zu POKEn. Mit 255 erzeugen Sie Blöcke, die durch 0 gelöscht werden können.

KOMPRIMIEREN IHRER SPRITE-PROGRAMME

Nun noch einen nützlichen Tip zum Komprimieren: Das oben beschriebene Programm ist zwar bereits ziemlich kurz, kann jedoch durch "Komprimieren" noch kürzer gestaltet werden. In unserem Beispiel zeigten wir die Spriteeingaben in verschiedenen Programmzeilen, so daß Sie sehen können, was im Programm passiert. Bei der tatsächlichen Anwendung wird ein guter Programmierer dieses Programm als ZWEIZEILEN-PROGRAMM schreiben, indem er es wie folgt komprimiert:

```
10PRINTCHR$(147):V=53248:POKEV+21,1:POKE2040,13:POKEV+39,1
20FORS=832TO894:POKES,255:NEXT:POKEV,24:POKEV+1,100
```

Bezüglich weiterer Einzelheiten über das Komprimieren von Programmen und somit das Einsparen von Speicherkapazität siehe Seite 24.



Für dieses Sprite hier muß sowohl die X-Position (horizontal) als auch die Y-Position (vertikal) angegeben werden, damit es auf dem Bildschirm angezeigt wird.

Abb. 3.4. Der Bildschirm ist in ein Gitter aus X- und Y-Koordinaten unterteilt.

POSITIONIERUNG DER SPRITES AUF DEM BILDSCHIRM

Der gesamte Bildschirm ist wie ein Koordinatensystem in X- und Y-Koordinaten unterteilt. Die X-Koordinate ist die horizontale Position und die Y-Koordinate die vertikale Position auf dem Bildschirm (siehe Abb. 3.4.).

Um die Sprites auf dem Bildschirm zu positionieren, müssen zwei Eingaben – X- und Y-Position – gePOKEt werden. Auf diese Weise erfährt der Computer, wo sich die obere linke Ecke des Sprites auf dem Bildschirm befinden soll. Bitte denken Sie daran, daß ein Sprite aus 504 einzelnen Pixels (24 horizontal mal 21 vertikal) besteht. Beim POKEn eines Sprites in die obere linke Bildschirmecke wird dieses als graphische Darstellung mit 24 horizontalen und 21 vertikalen Pixels angezeigt. Die Anzeige beginnt hierbei in der von Ihnen definierten X-Y-Position. Die Anzeige des Sprites basiert stets auf der oberen linken Ecke, auch wenn Sie für das gesamte Sprite lediglich einen kleinen Teil des 24-mal-21-Pixel-Spritebereichs benutzen.

Die Funktionsweise der X-Y-Positionierung können Sie dem nachstehenden Diagramm (Abb. 3.5.) entnehmen. Dieses zeigt die X- und Y-Zahlen in Zusammenhang mit der Bildschirmanzeige. Bitte beachten Sie, daß der graue Bereich im Diagramm den sichtbaren Bildschirmteil und der weiße Bereich den Teil außerhalb des Bildschirms angibt.

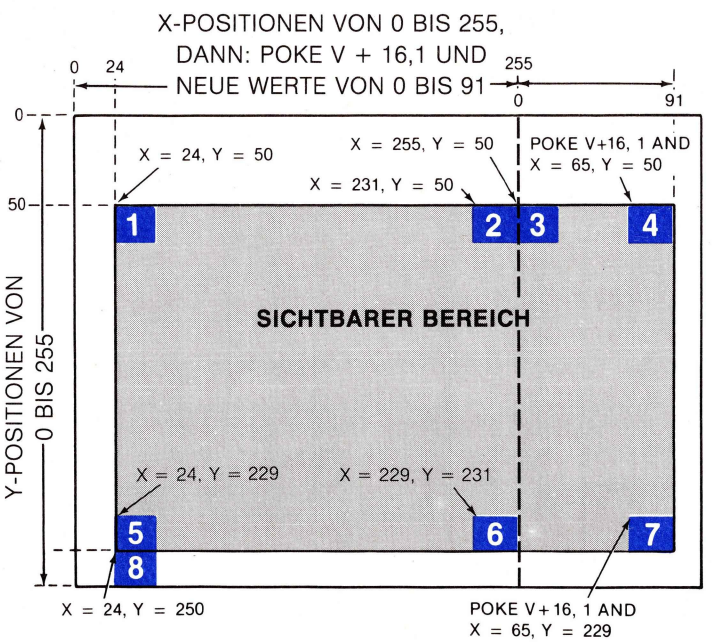


Abb. 3.5. Bestimmung der X-Y-Spritepositionen

Um ein Sprite an der gewählten Stelle anzuzeigen, müssen Sie die X- und Y-Eingaben für jedes Sprite POKEn . . . denken Sie daran, daß jedes Sprite seine eigene X- und Y-POKE-Anweisung hat. Nachstehend sehen Sie die X- und Y-Eingaben für alle 8 Sprites:

ZUM SETZEN DER X-Y-SPRITEPOSITIONEN POKEN SIE DIESE WERTE

	SPRITE0	SPRITE1	SPRITE2	SPRITE3	SPRITE4	SPRITE5	SPRITE6	SPRITE7
SET X	V,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
SET Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
RIGHTX	V+16,1	V+16,2	V+16,4	V+16,8	V+16,16	V+16,32	V+16,64	V+16,128

POKEN EINER X-POSITION: Die möglichen X-Werte sind, gezählt von links nach rechts, 0 bis 255. Die Werte 0 bis 23 plazieren alles oder einen Teil des Sprites außerhalb des sichtbaren Bereichs auf der linken Bildschirmseite – die Werte 24 bis 255 zeigen das Sprite im sichtbaren Bereich bis zur 255. Position an (bezüglich Einzelheiten über die Eingabe außerhalb der 255. X-Position siehe nachstehenden Abschnitt). Um ein Sprite in eine dieser Positionen zu plazieren, geben Sie lediglich die X-Positions-POKE-Anweisung für das benutzte Sprite ein. Um z. B. Sprite 1 an die äußerst linke X-Position im sichtbaren Bereich zu POKEn, geben Sie folgendes ein: POKE V + 2,24.

X-Werte außerhalb der 255. Position: Um über die 255. Position des Bildschirms hinaus zu gelangen, benötigen Sie eine zweite POKE-Anweisung. Normalerweise geht die horizontale Numerierung (X) über die 255. Position bis zu 256, 257 usw. hinaus. Da die Register jedoch nur 8 Bit enthalten, müssen wir ein "zweites Register" erstellen, um auf die rechte Bildschirmseite zu gelangen. Die X-Numerierung beginnt hier wieder mit 0. Um also über die X-Position 255 hinaus zu gelangen, ist POKE 5 + 16 sowie eine Zahl (abhängig vom Sprite) erforderlich. Hierdurch erhalten Sie 64 zusätzliche X-Positionen (numeriert von 0 bis 65) im sichtbaren Bereich auf der rechten Bildschirmseite. (Sie können den rechten X-Wert tatsächlich bis auf 255 POKEn.)

POKEN EINER Y-POSITION: Die möglichen Y-Werte sind 0 bis 255 und werden von oben nach unten gezählt. Durch die Werte 0 bis 49 wird das Sprite ganz oder teilweise außerhalb des sichtbaren Bereichs oben am Bildschirm angezeigt. Mit den Werten 50 bis 259 befindet sich das Sprite im sichtbaren Bereich. Durch die Werte 230 bis 255 wird das Sprite ganz oder teilweise aus dem sichtbaren Bereich hinausbewegt.

Wir wollen uns nun mit X-Y-Positionierung beschäftigen und nehmen hierzu Sprite 1 als Beispiel. Geben Sie folgendes Programm ein:

SHIFT CLR/HOME

```
10 PRINT"□":V=53248:POKEV+21,2:POKE2041,13:
FORS=832TO895:POKES,255:NEXT
20 POKEV+40,7
30 POKEV+2,24
40 POKEV+3,50
```

Dieses einfache Programm zeigt Sprite 1 als Kästchen an und setzt es in die obere linke Bildschirmecke. Ändern Sie die Zeile 40 nun wie folgt:

```
40 POKE V+3,229
```

Hierdurch wird das Sprite in die untere linke Bildschirmecke bewegt. Nun wollen wir die rechte X-Grenze des Sprites überprüfen: Ändern Sie Zeile 30 wie folgt:

```
30 POKE V+2,255
```

Hierdurch wird das Sprite nach rechts bewegt. Es erreicht jedoch die rechte X-Grenze, die durch 255 festgelegt ist. An diesem Punkt muß das höchste Bit in Register 16 gesetzt sein. D. h., Sie müssen POKE V + 16 sowie eine Zahl eingeben, die in der rechten "X"-Spalte in der X-Y-POKE-Tabelle angezeigt wird. Auf diese Weise wird der X-Positionszähler bei der 256. Pixel-Position auf dem Bildschirm neu gestartet. Ändern Sie Zeile 30 wie folgt:

```
30 POKE V+16, PEEK(V+16)OR 2:POKE V+2,0
```

Durch **POKE V + 16,2** wird das höchste Bit der X-Position für Sprite 1 gesetzt und bei der 256. Pixel-Position auf dem Bildschirm ein neuer 0-Punkt gesetzt. Durch **POKE V + 2,0** wird das Sprite an der neuen 0-Position, die nun auf den 256. Pixel gesetzt ist, angezeigt.
Um zurück zur linken Bildschirmseite zu gelangen, muß das höchste Bit des X-Positionszählers auf 0 gesetzt werden. Geben Sie hierzu für Sprite 1 folgendes ein:

```
POKE V+16, PEEK(V+16)AND 253
```

Fassen wir nun zusammen, wie die X-Positionierung funktioniert: Die X-Position für ein beliebiges Sprite wird mit einer Zahl von 0 bis 255 gePOKEt. Für Positionen rechts von der 255. Position auf dem Bildschirm benötigen Sie eine zusätzliche

Anweisung POKE (V + 16), durch die das höchste Bit der X-Position gesetzt und die Zählung beim 256. Pixel auf dem Bildschirm erneut bei 0 gestartet wird.
Durch diese POKE-Anweisung beginnt die X-Numerierung ab der 256. Position erneut bei 0. (**Beispiel: POKE V+16, PEEK(V+16) OR 1** und **POKE V,1** müssen enthalten sein, um Sprite 0 an die 257. Position auf dem Bildschirm zu setzen.) Um zurück zur linken X-Position zu gelangen, müssen Sie wieder "umschalten". Geben Sie hierzu **POKE V+16, PEEK(V+16)AND 254** ein.

POSITIONIEREN MEHRERER SPRITES AUF DEM BILDSCHIRM

Nachstehend sehen Sie ein Programm, das 3 verschiedene Sprites (0, 1 und 2) in verschiedenen Farben definiert und in verschiedenen Positionen auf dem Bildschirm darstellt:

```
10 PRINT"[SHIFT CLR/HOME]":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50
60 POKEV+2,12:POKEV+3,229
70 POKEV+4,255:POKEV+5,50
```

Der Einfachheit halber sind alle drei Sprites als durchgehende Quadrate definiert, die ihre Daten alle aus demselben Speicherbereich erhalten. Wichtig ist hierbei, wie alle drei Sprites positioniert werden. Das weiße Sprite 0 befindet sich in der oberen linken Ecke. Das gelbe Sprite 1 in der unteren Ecke, jedoch halb außerhalb des Bildschirms. Denken Sie daran, 24 ist die äußerste linke X-Position im sichtbaren Bereich . . . durch eine X-Position unter 24 wird das Sprite ganz oder teilweise aus dem Bildschirm "hinausgeschoben". Wir haben hier die X-Position 12 benutzt, so daß die Hälfte des Sprites außerhalb des Bildschirms liegt. Das orangene Sprite 2 liegt an der rechten X-Grenze (Position 255) . . . Wenn Sie nun aber ein Sprite anzeigen wollen, das im Bereich rechts von der X-Position 255 liegt?

ANZEIGE EINES SPRITES AUSSERHALB DER 255. X-POSITION

Um ein Sprite außerhalb der 255. X-Position anzuzeigen, ist eine besondere POKE-Anweisung erforderlich. Diese setzt das höchste Bit der X-Position und beginnt bei der 256. Pixel-Position auf dem Bildschirm. Das funktioniert folgendermaßen:
Geben Sie zunächst POKE V + 16 mit der Zahl für das Sprite, das Sie benutzen, ein (überprüfen Sie die rechte X-Reihe "RIGHT X" in der Tabelle X-Y . . . wir benutzen Sprite 0). Nun ordnen wir eine X-Position zu. Hierbei müssen wir darauf achten, daß der X-Zähler ab der 256. Bildschirmposition wieder bei 0 beginnt. Ändern Sie Zeile 50 wie folgt:

```
50 POKE V+16,1:POKE V,24:POKE V+1,75
```

Durch diese Zeile wird V + 16 mit der Zahl gePOKEt, die zum "Öffnen" der rechten Bildschirmseite benötigt wird. Die neue X-Position 24 für Sprite 0 beginnt nun 24 Pixel rechts neben der Position 255. Um die rechte Bildschirmkante zu überprüfen, ändern Sie Zeile 60 wie folgt:

```
60 POKE V+16,1:POKE V,65:POKE V+1,75
```

Probieren Sie die Eingaben in der Spritetabelle aus, damit Sie die Eingaben herausfinden, die für die Positionierung und Bewegung der Sprites auf dem Bildschirm erforderlich sind. Auch das Kapitel über "Bewegen von Sprites" wird Ihnen bei der Spritepositionierung helfen.

SPRITEPRIORITÄTEN

Verschiedene Sprites können sich vor- bzw. hintereinander auf dem Bildschirm bewegen. Dieser dreidimensionale Effekt wird durch die Spriteprioritäten erzielt, die bestimmen, welches Sprite bei einer eventuellen Überdeckung auf dem Bildschirm Vorrang gegenüber dem anderen hat.
Die Regelung, "wer zuerst kommt, mahlt zuerst", gilt auch hier: Das Sprite mit der niedrigeren Zahl hat automatisch Priorität über Sprites mit höheren Zahlen. Werden z. B. Sprite 0 und Sprite 1 an der gleichen Stelle des Bildschirms angezeigt, so erscheint Sprite 0 vor Sprite 1. Sprite 0 hat daher stets Vorrang vor allen anderen Sprites, da es dasjenige mit der niedrigsten Zahl ist. Sprite 1 hat Priorität über die Sprites 2 bis 7, Sprite 2 Vorrang vor den Sprites 3 bis 7 usw. Sprite 7 (das letzte Sprite) hat die niedrigste Priorität und wird daher bei einer Überdeckung stets hinter allen anderen Sprites erscheinen.
Ändern Sie die Zeilen 50, 60 und 70 des Programms wie folgt:

```
10 PRINT"□":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50:POKEV+16,0
60 POKEV+2,34:POKEV+3,60
70 POKEV+4,44:POKEV+5,70
```

(SHIFT CLR/HOME)

Sie müssen nun ein weißes Sprite über einem gelben Sprite sehen und über diesen beiden muß ein orangenes Sprite angezeigt sein. Nun wissen Sie, wie die Prioritäten gesetzt sind, und können die Sprites beliebig bewegen. Dies hilft bei der Programmierung von Trickfilmen.

ZEICHNEN EINES SPRITES

Das Zeichnen eines COMMODORE-Sprites verläuft genauso wie das Ausmalen eines Malbuchs. Jedes Sprite besteht aus winzigen Punkten, die Pixel genannt werden. Um ein Sprite zu zeichnen, brauchen Sie lediglich einige der Pixel "auszumalen".

Sehen Sie sich das nachstehende Gitter in Abb. 3.6. an. So sieht ein leeres Sprite aus:

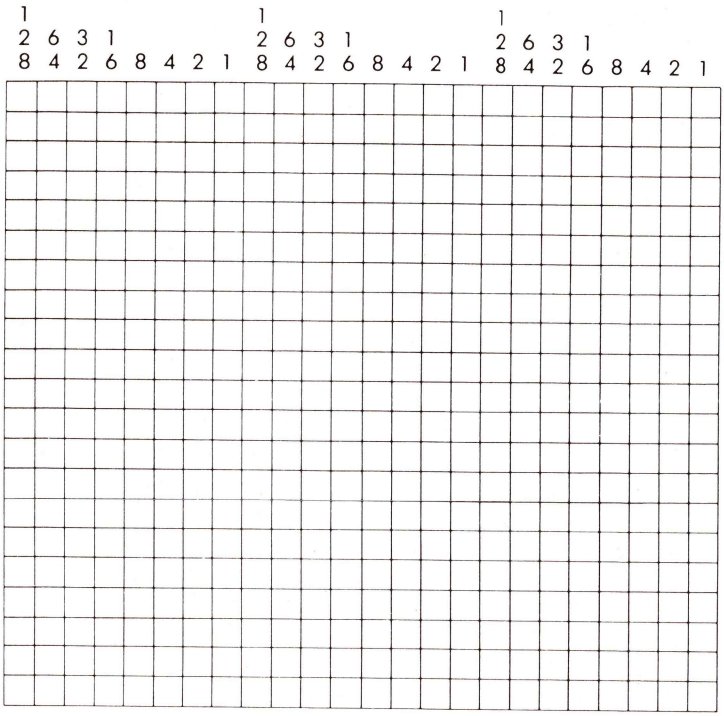


Abb. 3.6. Gitter für die Spriteerstellung

Jedes kleine "Kästchen" stellt ein Pixel im Sprite dar. Es gibt 24 horizontale mal 21 vertikale oder insgesamt 504 Pixel pro Sprite. Um dem Sprite nun eine bestimmte Form zu geben, müssen Sie diese Pixel mit einem speziellen Programm ausmalen . . . Wie können jedoch mehr als 500 Pixel gesteuert werden? Hierbei kann Ihnen die Computerprogrammierung helfen. Sie müssen nicht 504 einzelne Zahlen, sondern lediglich 63 Zahlen für jedes Sprite eingeben. Das funktioniert folgendermaßen . . .

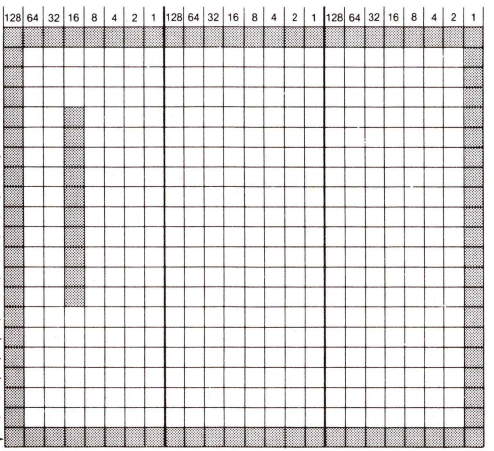
ERSTELLEN EINES SPRITES . . . SCHRITT FÜR SCHRITT

Um die Erstellung von Sprites für Sie so einfach wie möglich darzustellen, wollen wir das Ganze schrittweise erklären.

SCHRITT 1:

Schreiben Sie das Sprite-Erstellungs-Programm wie hier gezeigt auf ein Stück Papier . . . bitte beachten Sie, daß in Zeile 100 ein Abschnitt mit DATAs beginnt, der die 63 Zahlen für die Spriteerstellung enthält.

```
10 PRINT"□":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
100 DATA255,255,255
101 DATA128,0,1
102 DATA128,0,1
103 DATA128,0,1
104 DATA144,0,1
105 DATA144,0,1
106 DATA144,0,1
107 DATA144,0,1
108 DATA144,0,1
109 DATA144,0,1
110 DATA144,0,1
111 DATA144,0,1
112 DATA144,0,1
113 DATA144,0,1
114 DATA128,0,1
115 DATA128,0,1
116 DATA128,0,1
117 DATA128,0,1
118 DATA128,0,1
119 DATA128,0,1
120 DATA255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

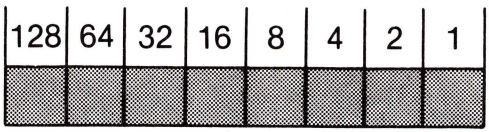


SCHRITT 2:

Malen Sie die Pixel im Gitter auf Seite 161 aus (oder nehmen Sie ein Blatt Millimeterpapier . . . denken Sie daran, daß ein Sprite aus 24 horizontalen mal 21 vertikalen Kästchen besteht). Benutzen Sie einen Bleistift und drücken Sie nicht zu fest auf, damit Sie dieses Gitter wieder benutzen können (oder machen Sie sich einige Fotokopien des Gitters). Sie können beliebige Bilder erstellen. Am Anfang wollen wir jedoch als Beispiel einen einfachen Kasten zeichnen.

SCHRITT 3:

Sehen Sie sich die ersten ACHT Pixel an. Über jeder Spalte von Pixels steht eine Zahl (128, 64, 32, 16, 8, 4, 2, 1). Die besondere Art der Addition, die wir benutzen wollen, stammt aus der BINÄR-ARITHMETIK, die bei Computern oft verwendet wird. Nachstehend sehen Sie genau die ersten acht Pixel in der oberen linken Spriteecke:



SCHRITT 4:

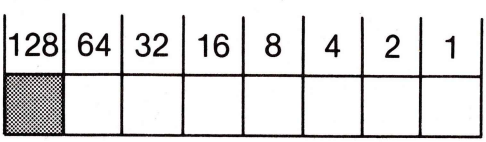
Addieren Sie die Zahlen der ersten ausgemalten Pixel. Die erste Gruppe der acht Pixel ist vollständig ausgemalt, so daß sich eine Summe von 255 ergibt.

SCHRITT 5:

Geben Sie diese Zahl als ERSTE DATA-ANWEISUNG in Zeile 100 des Sprite-Erstellungsprogramms ein. Geben Sie 255 für die zweite und dritte Achtergruppe ein.

SCHRITT 6:

Sehen Sie sich die ERSTEN ACHT PIXEL IN DER ZWEITEN SPRITE-REIHE an. Addieren Sie die Werte der ausgemalten Pixel. Da in unserem Beispiel nur ein Pixel ausgemalt ist, ergibt sich die Summe 128. Geben Sie diesen Wert als erste Datenzahl in Zeile 101 ein.



SCHRITT 7:

Addieren Sie die Werte der nächsten Gruppe von acht Pixels (die Summe ist 0, da hier alle Pixel leer sind). Geben Sie diese Zahl in Zeile 101 ein. Nun nehmen wir uns die nächste Gruppe vor und führen das gleiche für alle Achtergruppen durch (es gibt

drei Gruppen pro Reihe und insgesamt 21 Reihen). Es ergibt sich also eine Gesamtanzahl von 63. Jede Zahl gibt eine Gruppe zu je acht Pixel an, und 63 Gruppen mit acht Pixel ergeben insgesamt 504 vollständig unabhängige Pixel. Das Programm läßt sich vielleicht noch besser wie folgt erklären. Jede Programmzeile stellt eine Reihe im Sprite dar. Jede der drei Zahlen in jeder Reihe steht für eine Gruppe mit je acht Pixel. Und jede Zahl weist den Computer an, welches Pixel ausgemalt und welches leer sein soll.

SCHRITT 8:

KOMPRIMIEREN SIE DAS PROGRAMM. HIERZU WERDEN DIE DATA-ANWEISUNGEN ENTSPRECHEND NACHSTEHENDEM BEISPIELPROGRAMM ZUSAMMENGEFASST. Beachten Sie, daß Sie das Spriteprogramm zunächst auf ein Blatt Papier schreiben sollten. Das hat einen guten Grund. Die DATA-Anweisungszeilen 100 bis 120 im Programm in Schritt 1 sollen Ihnen lediglich zeigen, welche Zahl zu welcher Pixelgruppe Ihres Sprites gehört. Das endgültige Programm wird wie folgt komprimiert:

SHIFT CLR/HOME

```
10 PRINT"□":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
100 DATA255,255,255,128,0,1,128,0,1,128,0,1,144,0,
1,144,0,1,144,0,1,144,0,1
101 DATA144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,
144,0,1,128,0,1,128,0,1
102 DATA128,0,1,128,0,1,128,0,1,128,0,1,255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

BEWEGEN DER SPRITES AUF DEM BILDSCHIRM

Jetzt ist das Sprite fertig, und wir können es nun zu interessanten Dingen benutzen. Um das Sprite über den Bildschirm zu bewegen, fügen Sie folgende zwei Zeilen in Ihr Programm ein:

```
50 POKE V+5,100:FOR X=24TO255:POKE V+4,X:NEXT:POKE V+16,4
55 FOR X=0TO65:POKE V+4,X:NEXT X:POKE V+16,0:GOTO 50
```

Durch **ZEILE 50** wird die Y-Position bei 100 gePOKEt (probieren Sie auch 50 oder 229 aus). Dann wird eine FOR . . .NEXT-Schleife aufgebaut, durch die das Sprite nacheinander in die X-Position 0 bis 255 gePOKEt wird. Beim Erreichen der 255. Position wird das MSB gePOKEt (POKE V + 16,2), das zum Erreichen des rechten Bildschirmrandes benötigt wird.

ZEILE 55 enthält ebenfalls eine FOR . . .NEXT-Schleife, durch die das Sprite in die letzten 65 Bildschirmpositionen gePOKEt wird. Bitte beachten Sie, daß der X-Wert auf 0 zurückgestellt wurde. Da Sie jedoch das höchste Bit der X-Position gesetzt hatten (POKE V + 16,2), beginnt X auf der rechten Bildschirmseite.
Diese Zeile wird immer wieder durchlaufen (GOTO 50). Soll das Sprite sich nur einmal über den Bildschirm bewegen und dann verschwinden, nehmen Sie GOTO 50 einfach heraus.
Nachstehend sehen Sie eine Zeile, durch die das Sprite vor- und zurückbewegt wird:

```
50 POKE V+5,100:FOR X=24TO255:POKE V+4,X:NEXT: POKE
   V+16,4:FOR X=0TO65: POKE V+4,X: NEXT X
55 FOR X=65TO0 STEP-1:POKE V+4,X:NEXT:POKE V+16,0: FOR
   X=255TO24 STEP-1: POKE V+4,X:NEXT
60 GOTO 50
```

Sehen Sie, wie dieses Programm funktioniert? Es ist das gleiche wie das vorherige. Nur wird hier beim Erreichen der rechten Bildschirmseite das Programm stets umgekehrt, so daß das Sprite sich wieder in die andere Richtung bewegt. Dies wird durch STEP-1 bewirkt. Das Programm wird angewiesen, das Sprite in die X-Werte von 65 bis 0 auf der rechten Bildschirmseite und dann von 155 bis 0 auf der linken Bildschirmseite zu POKEn. Hierbei wird jeweils um den Schritt −1 zurückgegangen.

VERTIKALES ROLLEN

Diese Art der Spritebewegung wird "ROLLEN" genannt. Um das Sprite auf diese Weise nach oben oder unten in die Y-Position zu bewegen, brauchen Sie lediglich eine Zeile. Löschen Sie die Zeilen 50 und 55, indem Sie die Zeilennummern eingeben und danach RETURN drücken.

```
50 ( RETURN )
55 ( RETURN )
```

Geben Sie nun ZEILE 50 wie folgt ein:

```
50 POKE V+4,24:FOR Y=0TO255:POKE V+5,Y:NEXT
```

DIE TANZMAUS – EIN SPRITE-PROGRAMMBEISPIEL

Gelegentlich sind die in Programmieranleitungen beschriebenen Techniken nur schwer zu verstehen. Aus diesem Grund haben wir ein Spriteprogramm erstellt, das wir "Michaels Tanzmaus" nennen. Dieses Programm benutzt drei verschiedene Sprites in einem Zeichentrick mit Geräuscheffekten. Damit Sie genau verstehen, wie dieses Programm funktioniert, haben wir jeden Befehl erklärt:

```
5 S=54272:POKES+24,15:POKES,220:POKES+1,68:POKES+5,
15:POKES+6,215
10 POKES+7,120:POKES+8,100:POKES+12,15:POKES+13,215
15 PRINT"[SHIFT CLR/HOME]":V=53248:POKEV+21,1
20 FORS1=12288TO12350:READQ1:POKES1,Q1:NEXT
25 FORS2=12352TO12414:READQ2:POKES2,Q2:NEXT
30 FORS3=12416TO12478:READQ3:POKES3,Q3:NEXT
35 POKEV+39,15:POKEV+1,68
40 PRINTTAB(160)"[CTRL 2]I AM THE DANCING MOUSE![C= 7]"
45 P=192
50 FORX=0TO347STEP3
55 RX=INT(X/256):LX=X-RX*256
60 POKEV,LX:POKEV+16,RX
70 IFP=192THENGOSUB200
75 IFP=193THENGOSUB300
80 POKE2040,P:FORT=1TO60:NEXT
85 P=P+1:IFP>194THENP=192
90 NEXT
95 END
100 DATA30,0,120,63,0,252,127,129,254,127,129,254,
127,189,254,127,255,254
101 DATA63,255,252,31,187,248,3,187,192,1,255,128,
3,189,192,1,231,128,1,255,0
102 DATA31,255,0,0,124,0,0,254,0,1,199,32,3,131,
224,7,1,192,1,192,0,3,192,0
103 DATA30,0,120,63,0,252,127,129,254,127,129,254,
127,189,254,127,255,254
104 DATA63,255,252,31,221,248,3,221,192,1,255,128,
3,255,192,1,195,128,1,231,3
105 DATA31,255,255,0,124,0,0,254,0,1,199,0,7,1,128,
7,0,204,1,128,124,7,128,56
106 DATA30,0,120,63,0,252,127,129,254,127,129,254,
127,189,254,127,255,254
107 DATA63,255,252,31,221,248,3,221,192,1,255,134,
3,189,204,1,199,152,1,255,48
108 DATA1,255,224,1,252,0,3,254,0
109 DATA7,14,0,204,14,0,248,56,0,112,112,0,0,60,0,
-1
200 POKES+4,129:POKES+4,128:RETURN
300 POKES+11,129:POKES+11,128:RETURN
```

ZEILE 5:

S=54272	Setzt die Variable S gleich 54272, also der Anfangsspeicheradresse des SOUND CHIP. Statt nun einen direkten Speicherplatz zu POKEn, werden wir ab jetzt POKE S plus einen Wert eingeben.
POKES+24,15	Entspricht POKE 54296,15. Hierdurch wird die höchste Lautstärke eingestellt.
POKES,220	Entspricht POKE 54272,220. Setzt das Low Byte (LOW FREQUENCY) in Stimme 1 für eine Note, die ungefähr dem hohen C in Oktave 6 entspricht.
POKES+1,68	Entspricht POKE 54273,68. Jetzt das High Byte in Stimme 1 für eine Note, die *etwa* dem hohen C in Oktave 6 entspricht.
POKES+5,15	Entspricht POKE 54277,15. Setzt das Attack/Decay für Stimme 1 und besteht in diesem Fall aus dem max. Abklingpegel ohne Einsetzen. Hierdurch entsteht der Echo-Effekt.
POKES+6,215	Entspricht POKE 54278,215. Setzt das Sustain/Release für Stimme 1 (215 stellt eine Kombination zwischen Sustain- und Releasezeit dar).

ZEILE 10:

POKES+7,120	Entspricht POKE 54279,120. Setzt "High Frequency" für Stimme 2.
POKES+8,100	Entspricht POKE 54280,100. Setzt "Low Frequency" für Stimme 2.
POKES+12,15	Entspricht POKE 54284,15. Setzt Attack/Decay für Stimme 2 auf den gleichen Pegel wie für Stimme 1.
POKES+13,215	Entspricht POKE 54285,215. Setzt das Sustain/Release für Stimme 2 auf den gleichen Pegel wie für Stimme 1.

ZEILE 15:

PRINT" SHIFT CLR /HOME "	Löscht den Bildschirm bei Programmbeginn.
V=53248	Definiert die Variable "V" als Startadresse des VIC-Chip, der die Sprites steuert. Von nun an werden alle Spriteplätze als V plus einen Wert definiert.
POKEV+21,1	Schaltet die Spritenummer 1 ein (Aktivierung).

ZEILE 20:

FORS1=12288 TO 12350	In diesem Zeichentrick benutzen wir ein Sprite (Sprite 0). Wir werden jedoch DREI verschiedene Spritedaten für die Definition von drei unterschiedlichen Formen benutzen. Für den Zeichentrick schalten wir die Zeiger für Sprite 0 auf drei verschiedene Speicherplätze, in denen die Daten für die Definition der unterschiedlichen Formen gespeichert sind. Dasselbe Sprite wird hintereinander schnell in drei verschiedenen Formen definiert. Hierdurch entsteht der Tanzmaustrickfilm. Sie können Dutzende von Spriteformen in DATA-Anweisungen benutzen und diese Formen mit einem oder mehreren Sprites benutzen. Sie brauchen daher nicht ein Sprite auf eine Form zu begrenzen (und umgekehrt). Ein Sprite kann viele verschiedene Formen haben, indem einfach die Pointer für dieses Sprite auf verschiedene Adressen zeigen. In den Speicherplätzen sind dann die Spritedaten der verschiedenen Formen gespeichert. Diese Zeile bedeutet, daß wir die Date für die "Spriteform 1" in die Speicherplätze 12288 bis 12350 eingegeben haben.
READ Q1	Liest nacheinander 63 Zahlen der DATA-Anweisung, beginnend bei Zeile 100. Q1 ist ein beliebiger Variablenname. Es könnte auch A, Z1 oder eine andere numerische Variable benutzt werden.
POKES1,Q1	POKEt die erste Zahl der DATA-Anweisungen (erstes "Q1 ist 30) in den ersten Speicherplatz (12288). Entspricht POKE 12288,30.
NEXT	Weist den Computer an, die Befehle zwischen den Teilen FOR und NEXT der Schleife auszuführen. (READ Q1 und POKES1,Q1 mit den NEXT-Zahlen). D. h., durch die NEXT-Anweisung liest der Computer NEXT Q1 von den DATA-Anweisungen. NEXT Q1 ist 0. Außerdem wird S1 um 1 erhöht, dies entspricht 12289. Das Ergebnis ist POKE12289,0 . . . durch den NEXT-Befehl wird die Schleife bis zu den letzten Werten der Serie durchgeführt, also bis zu POKE 12350,0.

ZEILE 25:

FORS2=12352 TO 12414	Die zweite Form von Sprite 0 wird durch die DATAs definiert, die in die Adressen 12352 bis 12414 geschrieben werden. Bitte beachten Sie, daß Adresse 12351 übersprungen wird. Dies ist der 64. Platz der Definition der ersten Spritegruppe. Er enthält jedoch keine Spritedaten. Beachten Sie bei der Spritedefinition, daß 64 Plätze benutzt werden. Spritedaten werden jedoch nur in die ersten 63 Plätze gePOKEt.
READQ2	Liest die 63 Zahlen, die nach der Zahl der ersten Spriteform folgen. Durch diese READ-Anweisung wird die nächste Zahl im DATA-Bereich gesucht, und 63 Zahlen werden nacheinander gelesen.
POKES2,Q2	Hierdurch wird das Datum (Q2) in die Speicherplätze (S2) für unsere zweite Spriteform gePOKEt, die bei Adresse 12352 beginnt.
NEXT	Entspricht Zeile 20.

ZEILE 30:

FORS3=12416 TO 12478	Die dritte Form von Sprite 0 wird durch die DATAs in den Adressen 12416 bis 12478 definiert.
READQ3	Liest nacheinander die letzten 63 Zahlen als Q3.
POKES3,Q3	POKEt diese Zahlen in die Plätze 12416 bis 12478.
NEXT	Entspricht den Zeilen 20 und 25.

ZEILE 35:

POKEV+39,15	Setzt für Sprite 0 die Farbe hellgrau.
POKEV+1,68	Setzt die obere rechte Ecke des Spritequadrats in die Vertikalposition 68 (Y). Zum Vergleich: Position 50 ist die obere linke Y-Eck-Position auf dem sichtbaren Bildschirm.

ZEILE 40:

PRINTTAB(160)

Hierdurch wird der Cursor um 160 Leerstellen ab der oberen linken Bildschirmecke versetzt – dies entspricht vier Reihen. Hierdurch beginnt die PRINT-Meldung in der 5. Zeile auf dem Bildschirm.

"CTRL WHT

Die Tasten CTRL und WHT gleichzeitig drücken. Geschieht dies innerhalb von Anführungszeichen, so erscheint ein "umgekehrtes E". Hierdurch wird die Farbe aller nachfolgenden Eingaben Weiß.

I AM THE DANCING MOUSE!

Dies ist eine einfache PRINT-Anweisung.

C= 7 "

Hierdurch wird die Farbe nach Ende der PRINT-Anweisung von Schwarz auf Hellblau geändert. Durch gleichzeitiges Drücken der Tasten C= und 7 innerhalb von Anführungszeichen wird eine "negativ dargestellte Raute" angezeigt.

ZEILE 45:

P=192

Setzt die Variable P gleich 192. Die Zahl 192 ist der zu benutzende Zeiger. In diesem Fall wird Sprite 0 aus den Speicherplätzen ausgelesen, die ab Adresse 12288 beginnen. Durch "Verstellen" des Zeigers auf die Adressen der beiden anderen Spriteformen kann mit nur einem Sprite ein Trickfilm mit drei verschiedenen Formen erstellt werden.

ZEILE 50:

FORX=0TO347
STEP3

Bewegt das Sprite von Position 0 bis Position 347 in 3er-Schritten (hierdurch entsteht schnelle Bewegung).

ZEILE 55:

RX=INT(X/256)	RX ist der ganzzahlige Anteil von X/256, was bedeutet, daß RX auf 0 gerundet wird, wenn X kleiner als 256 ist, und auf 1, wenn X die Position 256 erreicht. Wir werden RX gleich für die Anweisung POKE V + 16 mit einer 0 oder 1 benutzen, um die rechte Bildschirmseite "einzuschalten".
LX=X−RX*256	Wenn sich das Sprite an der X-Position 0 befindet, sieht die Gleichung wie folgt aus: LX=0 − (0 mal 256) = 0. Wenn sich das Sprite an der X-Position 1 befindet, sieht die Gleichung wie folgt aus: LX=1 − (0 mal 256) = 1. Wenn sich das Sprite an der X-Position 256 befindet, sieht die Gleichung so aus: LX=256 − (1 mal 256) = 0. Hierdurch wird X zurück auf 0 gesetzt. Dies ist erforderlich, wenn die Bewegung bis zum rechten Rand reichen soll (POKE V + 16,1).

ZEILE 60:

POKEV,LX	Mit der Anweisung POKE V wird die horizontale Position (X) von Sprite 0 auf den Bildschirm gesteuert. (Siehe Spritetabelle auf Seite 176.) Wie oben gezeigt, ändert sich der Wert von LX (horizontale Spriteposition) von 0 bis 255. Wenn er 255 erreicht, wird er automatisch aufgrund der Gleichung LX in Zeile 55 auf 0 zurückgestellt.
POKEV+16,RX	Durch POKE V + 16 wird stets die rechte Bildschirmseite nach Erreichen der Position 256 eingeschaltet, um die horizontalen Positionierungskoordinaten auf 0 zurückzustellen. RX ist entweder 0 oder 1, je nach der durch die Gleichung RX in Zeile 55 bestimmten Spriteposition.

ZEILE 70:

IFP=192THEN GOSUB200	Ist der Sprite-Pointer auf 192 gesetzt (erste Spriteform), dann wird die Wellenformsteuerung des ersten Geräuscheffekts in Zeile 200 auf 129 und 128 gesetzt.

ZEILE 75:

IFP=193THEN GOSUB300

Ist der Sprite-Pointer auf 193 (zweite Spriteform) gesetzt, dann wird die Wellenformsteuerung für den zweiten Geräuscheffekt (Stimme 2) auf 124 und 128 in Zeile 300 gesetzt.

ZEILE 80:

POKE2040,P

Setzt den Sprite-Pointer auf Adresse 192 (erinnern Sie sich noch an P=192 in Zeile 45? P wird nun hier benutzt).

FORT=1TO60: NEXT

Eine einfache Zeitverzögerungs-Schleife, die die Geschwindigkeit festlegt, mit der die Maus tanzt. (Probieren Sie eine höhere bzw. geringere Geschwindigkeit durch Erhöhung/Reduzierung der Zahl 60 aus.)

ZEILE 85:

P=P+1

Nun erhöhen wir den Zeigerwert, indem wir den Originalwert P um 1 erhöhen.

IFP>194THEN P=192

Wir wollen das Sprite nur auf drei Adreßbereiche zeigen lassen. 192 zeigt auf die Adressen 12288 bis 12350, 193 auf die Adressen 12352 bis 12414 und 194 auf die Adressen 12416 bis 12478. Diese Zeile weist den Computer an, P zurück auf 192 zu setzen, sobald P 195 wird. Auf diese Weise kann P nie wirklich 195 werden. P ist 192, 193, 194 und wird dann zurück auf 192 gesetzt. Der Zeiger zeigt nacheinander auf die drei Spriteformen in den drei 64-Byte-Gruppen der Adreßbereiche mit den Daten.

ZEILE 90:

NEXTX

Nachdem das Sprite eine der drei durch die DATAs bestimmten Formen erhalten hat, kann es sich über den Bildschirm bewegen. Es überspringt jeweils drei X-Positionen (und bewegt sich nicht ruhig um jeweils eine Position weiter, was auch möglich ist). Hierdurch tanzt die Maus schneller über den Bildschirm. NEXT X schließt die Schleife FOR . . . X in Zeile 50 ab.

ZEILE 95:

END	Beendet das Programm, wenn das Sprite sich aus dem Bildschirm hinaus bewegt.

ZEILEN 100–109:

DATA	Die Spriteformen werden nacheinander aus den DATA-Anweisungen gelesen. Zunächst werden die 63 Zahlen, die die Spriteform 1 enthalten, gelesen, danach die 63 Zahlen für Spriteform 2 und dann für Spriteform 3. Die Daten werden in die drei aufeinanderfolgenden Adreßbereiche gelesen. Nach dem Einlesen in diese Adressen braucht Sprite 0 lediglich noch auf die drei Speicherplätze zu zeigen. Das Sprite nimmt dann automatisch die entsprechende Form an. Da es auf diese Weise nacheinander entsprechend den Daten in den drei Speicheradressen unterschiedliche Formen annimmt, können wir einen Trickfilmeffekt erzeugen. Wenn Sie wissen wollen, wie diese Zahlen das einzelne Sprite beeinflussen, verändern Sie die ersten drei Zahlen in den Zeilen 100 bis 255. Bezüglich weiterer Einzelheiten schlagen Sie bitte im Abschnitt über die Definition der Spriteformen nach.

ZEILE 200:

POKES+4,129	Die auf 129 gesetzte Wellenformsteuerung schaltet den Geräuscheffekt ein.
POKES+4,128	Die auf 128 gesetzte Wellenformsteuerung schaltet den Geräuscheffekt aus.
RETURN	Läßt das Programm zu Zeile 70 zurückspringen, nachdem die Eingaben für die Wellenformsteuerung geändert wurden.

ZEILE 300:

POKES+11,129	Die auf 129 gesetzte Wellenformsteuerung schaltet den Geräuscheffekt ein.
POKES+11,128	Die auf 128 gesetzte Wellenformsteuerung schaltet den Geräuscheffekt aus.
RETURN	Läßt das Programm zurück zum Ende von Zeile 75 zurückspringen.

TABELLE ZUM EINFACHEN KONSTRUIEREN VON SPRITES

	SPRITE 0	SPRITE 1	SPRITE 2	SPRITE 3	SPRITE 4	SPRITE 5	SPRITE 6	SPRITE 7
Sprite einschalten	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128
Speichern im Adreßbereich (Zeiger setzen)	2040, 192	2041, 193	2042, 194	2043, 195	2044, 196	2045, 197	2046, 198	2047, 199
Plätze für Sprite-Pixels (12288–12798)	12288 to 12350	12352 to 12414	12416 to 12478	12480 to 12542	12544 to 12606	12608 to 12670	12672 to 12734	12736 to 12798
Spritefarbe	V+39,C	V+40,C	V+41,C	V+42,C	V+43,C	V+44,C	V+45,C	V+46,C
Linke X-Position setzen (0–255)	V+0,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Rechte X-Position setzen (0–255)	V+16,1 V+0,X	V+16,2 V+2,X	V+16,4 V+4,X	V+16,8 V+6,X	V+16,16 V+8,X	V+16,32 V+10,X	V+16,64 V+12,X	V+16,128 V+14,X
Y-Position setzen	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
Sprite horizontal (X) vergrößern	V+29,1	V+29,2	V+29,4	V+29,8	V+29,16	V+29,32	V+29,64	V+29,128
Sprite vertikal (Y) vergrößern	V+23,1	V+23,2	V+23,4	V+23,8	V+23,16	V+23,32	V+23,64	V+23,128
Setzen des Mehrfarbenmodus	V+28,1	V+28,2	V+28,4	V+28,8	V+28,16	V+28,32	V+28,64	V+28,128
Mehrfarbe 1 (erste Farbe)	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C
Mehrfarbe 2 (zweite Farbe)	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C
Setzen der Sprite-Prioritäten	Die Sprites mit der niedrigeren Zahl haben stets Vorrang vor den Sprites mit der höheren Zahl. So hat z. B. Sprite 0 Vorrang vor allen anderen Sprites und Sprite 7 die letzte Priorität. Sprites mit niedrigeren Zahlen erscheinen daher stets vor Sprites mit höheren Zahlen.							
Kollision (zwischen Sprites)	V+30 IF PEEK(V+30)ANDX=X THEN [action]							
Kollision (zwischen Sprites und Hintergrund)	V+31 IF PEEK(V+31)ANDX=X THEN [action]							

HINWEISE ZUR SPRITE-ERSTELLUNG

Verschiedene Sprite-Speicher-Zeiger und Speicheradressen bei Verwendung des Kassettenpuffers

Speichern im Adreßbereich (Zeiger setzen)	SPRITE 0 2040,13	SPRITE 1 2041,14	SPRITE 2 2042,15	Wenn Sie 1 bis 3 Sprites benutzen, können Sie die Speicherplätze im Kassettenpuffer (832 bis 1023) benutzen. Bei mehr als 3 Sprites empfehlen wir jedoch, die Plätze 12288 bis 12798 (siehe Tabelle) zu benutzen.
Sprite-Pixel-Adressen für Speicherblöcke 13–15	832 bis 894	896 bis 958	960 bis 1022	

EINSCHALTEN DER SPRITES:

Durch POKE V+21 ,X (X = Zahl aus der Tabelle) können Sie jedes beliebige Sprite einschalten. Durch Einschalten von nur einem Sprite werden jedoch andere Sprites ausgeschaltet. Um zwei oder mehrere Sprites einzuschalten, müssen die Zahlen der betreffenden Sprites addiert werden. (Durch POKE V+21,6 werden z. B. die Sprites 1 und 2 eingeschaltet.) Nachstehend wird erklärt, wie Sie ein Sprite ein- und ausschalten können, ohne andere Sprites zu beeinträchtigen (besonders nützlich bei Trickfilmen).

BEISPIEL:

Um nur Sprite 0 auszuschalten, geben Sie ein: POKE V+21, PEEK V+21AND(255–1). Ändern Sie die Zahl 1 in (255–1) in 1,2,4,8,16,32,64 oder 128 um (für die Sprites 0 bis 7). Um das Sprite wieder einzuschalten und nicht die bereits eingeschalteten übrigen Sprites zu beeinflussen, geben Sie POKE V+21, PEEK(V+21)OR 1 ein und ändern Sie OR 1 in OR 2 (Sprite 2), =R 4 (Sprite 3) usw. um.

X-POSITIONSWERTE AUSSERHALB VON 255:

X-Positionen reichen von 0 bis 255 . . . und beginnen dann wieder bei 0. Um ein Sprite über die X-Position 255 hinaus bis an den rechten Bildschirmrand zu bewegen, ist zunächst die Anweisung POKE V+16 erforderlich. Dann wird ein neuer X-Wert von 0 bis 63 gePOKEt, der das Sprite in eine der X-Positionen auf der rechten Bildschirmseite setzt. Um zurück zu den Positionen 0 bis 255 zu gelangen, ist POKE V+16,0 und das POKEn eines X-Werts zwischen 0 und 255 erforderlich.

Y-POSITIONSWERTE:

Y-Positionen gehen von 0 bis 255. Hierbei liegt 0 bis 49 über dem OBEREN Bildschirmrand, 50 bis 229 IM sichtbaren Bereich und 230 bis 255 AUSSERHALB des unteren Bildschirmrandes.

SPRITEFARBEN:

Damit Sprite 0 weiß wird, geben Sie folgendes ein: POKE V+39,1 (benutzen Sie die FARB-POKE-EINGABE in der vorstehenden Tabelle sowie die nachstehenden Farb-Codes):

0–SCHWARZ	4–PURPUR	8–ORANGE	12–MITTELGRAU
1–WEISS	5–GRÜN	9–BRAUN	13–HELLGRÜN
2–ROT	6–BLAU	10–HELLROT	14–HELLBLAU
3–ZYAN	7–GELB	11–DUNKELGRAU	15–HELLGRAU

SPEICHERPLATZ:

Für jedes Sprite müssen Sie einen getrennten 64-Byte-Satz im Computerspeicher "reservieren". Hiervon werden 63 Bytes für die Spritedaten benutzt. Die nachstehend gezeigten Speicherbereiche entsprechen den Spritezeigern in obiger Tabelle. Jedes Sprite kann entsprechend Ihren Wünschen definiert werden. Wenn alle Sprites gleich sein sollen, müssen sie auf die gleichen Spritedaten zeigen.

VERSCHIEDENE SPRITEZEIGER-EINGABEN:

Diese Spritezeiger-Eingaben sind NUR ALS EMPFEHLUNGEN zu verstehen.
Achtung: Sie können Spritezeiger beliebig im RAM-Speicher setzen. Werden sie jedoch zu "niedrig" im Speicher gesetzt, kann ein langes BASIC-Programm Ihre Spritedaten überschreiben oder umgekehrt. Um ein besonders langes BASIC-Programm vor einer Überschreibung durch Spritedaten oder umgekehrt zu schützen, können die Sprites in einem höheren Speicherbereich abgelegt werden (z. B. 2040, 192 für Sprite 0 an den Plätzen 12288 bis 12350 . . . 2041, 193 an den Plätzen 12352 bis 12414 für Sprite 1 usw.). Durch geschickte Wahl der Speicheradressen, aus denen die Sprites ihre Daten empfangen, können Sie 64 verschiedene Sprites sowie ein ansehnliches BASIC-Programm zusammen benutzen.
Definieren Sie hierfür verschiedene Spriteformen in Ihren DATA-Anweisungen und definieren Sie dann ein bestimmtes Sprite neu, indem Sie den "Zeiger" so ändern, daß für das betreffende Sprite verschiedene Speicherbereiche mit verschiedenen Spritebilddaten benutzt werden. Sehen Sie sich hierzu auch das Programm "Tanz-

maus“ an. Sollen zwei oder mehrere Sprites die gleiche Form haben (Sie können immer noch Position und Farbe jedes Sprites ändern), benutzen Sie den gleichen Spritezeiger und damit den gleichen Speicherbereich für die betreffenden Sprites (So können z. B. die Sprites 0 und 1 auf den gleichen Speicherplatz zeigen. Hierzu dient die Anweisung POKE 2040,192 und POKE 2041,192).

PRIORITÄTEN:

Priorität bedeutet, daß ein Sprite vor oder hinter einem anderen Sprite auf dem Bildschirm angezeigt wird. Sprites mit höherer Priorität erscheinen stets vor (bzw. über) den Sprites mit niedrigerer Priorität. Hierbei haben Sprites mit niedrigerer Zahl stets den Vorrang vor solchen mit höherer Zahl. D. h., Sprite 0 hat Priorität über alle anderen Sprites und Sprite 7 die niedrigste Priorität. Entsprechend hat Sprite 1 Vorrang vor den Sprites 2 bis 7. Befinden sich zwei Sprites in der gleichen Bildschirmposition gegeben, so erscheint das mit der höheren Priorität vor dem mit der niedrigeren. Das Sprite mit der niedrigeren Priorität ist entweder verdeckt oder “scheint durch“.

ARBEITEN IM MEHRFARBENMODUS:

Sie können mehrfarbige Sprites erstellen. Im Mehrfarbenmodus müssen Sie jedoch statt einzelner Punkte in Ihrem Spritebild stets Pixel-Paare benutzen (d. h., jeder farbige “Punkt“ oder “Block“ im Sprite besteht aus mindestens zwei nebeneinander liegenden Pixels). Es stehen vier Farben zur Auswahl: Spritefarbe (siehe obige Tabelle), Hilfsfarbe 1, Hilfsfarbe 2 und “Hintergrundfarbe“ (die Hintergrundfarbe wird durch eine 0-Eingabe angewählt. In diesem Fall scheint der Hintergrund durch.). Betrachten Sie einen horizontalen 8-Pixel-Satz in einem Spritemuster. Je nachdem, ob das linke, rechte oder beide Pixel ausgefüllt sind, wird die Farbe jedes Pixel-Paares bestimmt.



HINTERGRUND (Wenn beide Pixel leer (0) sind, scheint die Bildschirmfarbe durch.)



MEHRFARBIG 1 (Wenn das rechte Pixel in einem Pixel-Paar ausgefüllt ist, werden beide in der Hilfsfarbe 1 dargestellt.)

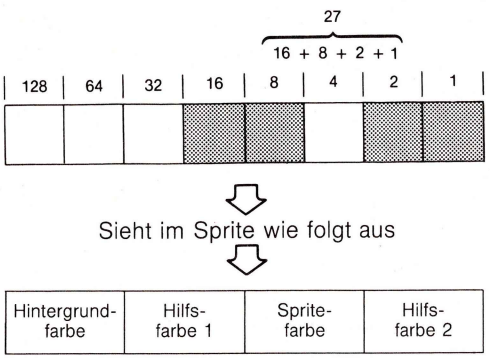


SPRITEFARBE (Wenn das linke Pixel in einem Pixel-Paar ausgemalt ist, haben beide die Spritefarbe.)



MEHRFARBIG 2 (Wenn beide Pixel in einem Pixel-Paar ausgemalt sind, werden beide in der Hilfsfarbe 2 dargestellt.)

Sehen Sie sich die nachstehende horizontale 8-Pixel-Reihe an. Nach dem oben Gesagten erscheinen die ersten zwei Pixel in der Hintergrundfarbe, die zweiten in der Hilfsfarbe 1, und für die dritten zwei Pixel gilt die Spritefarbe. Die vierten zwei Pixel erscheinen in der Hilfsfarbe 2. Die Farbe der einzelnen Pixel-Paare hängt also davon ab, welche Bits in dem Paar ausgemalt und welche leer sind. Wenn Sie festgelegt haben, welche Farben Sie für die einzelnen Pixel-Paare wünschen, müssen die Werte der ausgemalten Pixel im 8-Pixel-Satz addiert und danach diese Zahl in den geeigneten Speicherplatz gePOKEt werden. Ist z. B. die nachstehende 8-Pixel-Reihe die erste Reihe in einem Sprite, die bei Speicherplatz 832 beginnt, so lautet der Wert der ausgemalten Pixel 16+8+2+1 = 27. Es gilt also folgende Anweisung: POKE 832,27.



KOLLISION:

In dieser Zeile wird geprüft, ob ein bestimmtes Sprite mit irgendeinem anderen Sprite kollidiert.
X entspricht 1 für Sprite 0, 2 für Sprite 1, 4 für Sprite 2, 8 für Sprite 3, 16 für Sprite 4, 32 für Sprite 5, 64 für Sprite 6 und 128 für Sprite 7.
Über folgende Zeile können Sie erkennen, ob Sprites miteinander kollidiert haben:
IF PEEK(V+30)ANDX=XTHEN (hier Aktion eingeben).

BENUTZEN VON GRAPHIKZEICHEN IN DATENANWEISUNGEN:

Folgendes Programm ermöglicht Ihnen die Erstellung eines Sprites mit leeren und ausgemalten Kreisen (SHIFT Q) in DATA-Anweisungen. Die in die Sprite-Datenregister gePOKEten Sprites und Zahlen werden angezeigt.

```
                SHIFT CLR/HOME
10 PRINT"□":FORI=0TO63:POKE832+I,0:NEXT
20 GOSUB60000
999 END
60000 DATA"          ●●●●●●●          "
60001 DATA"        ●●●●●●●●●●●●       "
60002 DATA"       ●●●●●●●●●●●●●●      "
60003 DATA"       ●●●●●    ●●●●●      "
60004 DATA"      ●●●●● ●●●   ●●●●     "
60005 DATA"      ●●●●● ●●● ●●●●●      "
60006 DATA"      ●●●●● ●●●   ●●●●     "
60007 DATA"       ●●●●●    ●●●●●      "
60008 DATA"       ●●●●●●●●●●●●●●      "
60009 DATA"       ●●●●●●●●●●●●●●      "
60010 DATA"       ● ●●●●●●●●● ●       "
60011 DATA"        ● ●●●●●●● ●        "
60012 DATA"        ●   ●●●●●   ●      "
60013 DATA"          ●   ●●●   ●      "
60014 DATA"          ●   ●●●   ●      "
60015 DATA"           ●   ●   ●       "
60016 DATA"           ●   ●   ●       "
60017 DATA"            ●●●●●          "
60018 DATA"            ●●●●●          "
60019 DATA"            ●●●●●          "
60020 DATA"              ●●●          "
60100 V=53248:POKEV,200:POKEV+1,100:POKEV+21,1:
POKEV+39,14:POKE2040,13
60105 POKEV+23,1:POKEV+29,1
60110 FORI=0TO20:READA$:FORK=0TO2:T=0:FORJ=0TO7:B=0
60140 IFMID$(A$,J+K*8+1,1)="●"THENB=1
60150 T=T+B*2↑(7-J):NEXT:PRINTT;:POKE832+I*3+K,T:
NEXT:PRINT:NEXT
60200 RETURN
```

KAPITEL 4

MUSIK-PROGRAMMIERUNG MIT DEM COMMODORE 64

- Einleitung
  Lautstärkenregelung
  Tonfrequenzen
- Arbeiten mit mehreren Stimmen
- Ändern der Wellenformen
- Hüllkurvengenerator
- Filtern
- Fortschrittliche Techniken
- Synchronisation und Ringmodulation

EINFÜHRUNG

Ihr COMMODORE-Computer ist mit einem hochentwickelten elektronischen Musiksynthesizer ausgerüstet. Er verfügt über drei Stimmen, ist vollständig adressierbar, **ATTACK/DECAY/SUSTAIN/RELEASE (ADSR),** Filtern, Modulation und "weißes Rauschen" sind einstellbar. Diese Funktionen stehen Ihnen direkt über wenige, einfache BASIC- und/oder Assembler-Anweisungen und -Funktionen zur Verfügung. Auf diese Weise können Sie komplexe Geräuscheffekte und Songs mit relativ einfachen Programmen erzielen.

In diesem Kapitel werden alle Möglichkeiten des Chip 6581 "SID", dem Geräusch- und Musiksynthesizer Ihres COMMODORE-Computers beschrieben. Es werden sowohl Theorie als auch die praktischen Aspekte beschrieben. Sie müssen weder ein erfahrener Programmierer noch ein Musikexperte sein, um mit dem Musiksynthesizer erstaunliche Ergebnisse zu erzielen. Sie werden hier zahlreiche Programmierbeispiele mit detaillierten Erklärungen finden.

Der Tongenerator wird durch POKE-Anweisungen in die entsprechenden Speicherplätze gesteuert. Die einzelnen Adressen sind in Anhang O aufgelistet. Die verschiedenen Konzepte werden wir schrittweise erklären. Sie werden danach in der Lage sein, nahezu unendlich viele verschiedene Geräusche zu erzeugen und selbständig mit dem Musiksynthesizer zu experimentieren.

Jeder Abschnitt dieses Kapitels beginnt mit einem Beispiel, dieses wird dann Zeile für Zeile genau beschrieben. Auf diese Weise sehen Sie, wie die einzelnen Funktionen richtig eingesetzt werden. Die technischen Erläuterungen können Sie durchlesen, wenn Sie wissen wollen, was tatsächlich passiert.

Wichtig bei den Musikprogrammen ist die POKE-Anweisung. Durch POKE wird in dem betreffenden Speicherplatz (MEM) ein festgelegter Wert (NUM) geschrieben.

```
POKE MEM,NUM
```

Die für die Musiksynthese benutzten Speicherplätze (MEM) beginnen beim COMMODORE 64 bei 54272 ($D400). Beim Arbeiten mit den 6581 (SID) Chip-Registern müssen Sie wissen, was in den POKE-Speicherplätzen 54272 bis einschließlich 54296 steht. Es ist jedoch auch möglich, zum Arbeiten mit diesen Adressen sich lediglich 54272 zu merken und danach eine Zahl von 0 bis 27 zu addieren. Auf diese Weise ist ein POKEn aller Speicherplätze von 54272 bis 54296 des SID-Chips möglich. In den POKE-Anweisungen dürfen die Zahlen (NUM) 0 bis einschließlich 255 benutzt werden.
Wenn Sie bereits Erfahrung auf dem Gebiet der Musik haben, können Sie auch noch die PEEK-Funktion nutzen. PEEK ist eine Funktion, mit der der derzeitige Wert ermittelt werden kann, der im angezeigten Speicherplatz steht.

X=PEEK(MEM)

Der Wert der Variablen X wird gleich dem derzeitigen Inhalt des Speicherplatzes MEM gesetzt.
Natürlich beinhalten Ihre Programme weitere BASIC-Befehle, die jedoch im Abschnitt "BASIC-ANWEISUNGEN" dieses Handbuchs erklärt sind.
Wir wollen nun anfangen und ein einfacheres Programm, das nur eine der drei Stimmen benutzt, ausprobieren. Geben Sie über die Tastatur NEW, dann dieses Programm und dann RUN ein. Speichern Sie das Programm danach auf der DATASSETTE ™1 oder Diskette.

PROGRAMMBEISPIEL 1:

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT:REM CLEAR SOUND CHIP
20 POKES+5,9:POKES+6,0
30 POKES+24,15                :REM SET VOLUME TO
MAXIMUM
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Das soeben eingegebene Programm wird nun Zeile für Zeile beschrieben. Lesen Sie dies durch, wenn Sie bestimmte Programmteile nicht genau verstanden haben.

BESCHREIBUNG DER EINZELNEN ZEILEN VON PROGRAMMBEISPIEL 1:

Zeile(n)	Beschreibung
5	S Anfangsadresse des Sound-Chip.
10	Löschen aller Sound-Chip-Register.
20	Eingabe von ATTACK/DECAY (Anstieg/Abfall) für Stimme 1 (A=0, D=9).
	Eingabe für SUSTAIN/RELEASE (Dauer/Ausklingen) von Stimme 1 (S=0, R=0).
30	Lautstärke auf Maximum.
40	Lesen "hohe Frequenz", "niedrige Frequenz", Dauer der Note.
50	Ist "hohe Frequenz" kleiner als 0, dann ist die Melodie zu Ende.
60	POKEn von hoher und niedriger Frequenz für Stimme 1.
70	Gate für Sägezahnwellenform für Stimme 1.
80	Zeitschleife für Dauer der Note.
90	Auslösen der Sägezahnwellenform für Stimme 1.
100	Rückkehr zur nächsten Note.
110–180	Songdaten: hohe Frequenz, niedrige Frequenz, Dauer der einzelnen Noten (Anzahl der Durchläufe).
190	Letzte Melodienote; die drei "–1" zeigen das Ende der Melodie an.

LAUTSTÄRKEREGELUNG

Das Chip-Register 24 enthält die Gesamtlautstärkeregelung. Die Lautstärke kann auf einen beliebigen Wert zwischen 0 und 15 eingestellt werden. Die anderen vier Bits werden später beschrieben. Jetzt brauchen Sie bloß zu wissen, daß für Lautstärke die Werte 0 bis 15 gelten. Schauen Sie sich Zeile 30 im Programmbeispiel 1 an.

TONFREQUENZEN

Töne entstehen durch Wellenbewegung der Luft. Stellen Sie sich vor, Sie werfen einen Stein ins Wasser und beobachten dann, wie die Wellen von innen nach außen verlaufen.
Wenn solche Wellen in der Luft entstehen, können wir sie hören. Die Sekundenzahl für einen Wellenzyklus (n = Anzahl der Sekunden) erhalten wir, indem wir die Zeit

zwischen einer Wellenspitze zur nächsten messen. Der Kehrwert dieser Zahl (1/n) gibt Ihnen die Zyklen pro Sekunde an. Zyklen pro Sekunde sind besser als Frequenz bekannt. Die Tonhöhe wird anhand der Frequenz bestimmt. Der Tongenerator des COMMODORE-Computers benutzt zwei Adressen zur Frequenzbestimmung. Die Frequenzwerte, die Sie für acht Oktaven benötigen, sind im Anhang E aufgelistet.
Für eine nicht in dieser Tabelle aufgeführte Frequenz benutzen Sie "F_{out}" (Frequenzausgabe) sowie nachstehende Gleichung zur Darstellung der Frequenz (F_n) des gewünschten Tons. Denken Sie daran, daß jede Note zwei Angaben, "hohe" und "niedrige" Frequenz, benötigt.

$$F_n = F_{out}/.06097$$

Wenn Sie herausgefunden haben, wie F_n für Ihre "neue" Note lautet, können Sie nun die Werte für hohe und niedrige Frequenz für diese Note erstellen. Hierzu wird F_n zunächst abgerundet, so daß keine Stellen mehr rechts neben dem Dezimalpunkt stehen. Sie haben nun eine ganze Zahl. Nun wird der Anteil der hohen Frequenz (F_{hi}) anhand der Gleichung $F_{hi}=F_n/256$ und der der niedrigen Frequenz (F_{lo}) durch $F_{lo}=F_n-(256*F_{hi})$ bestimmt.
Sie haben nun bereits mit einer Computerstimme gespielt. Wenn Sie wollen, können Sie nun Ihre Lieblingsmelodie programmieren und Dirigent Ihres eigenen Computerorchesters werden.

ARBEITEN MIT MEHREREN STIMMEN

Der COMMODORE-Computer verfügt über drei unabhängig steuerbare Stimmen (Oszillatoren). Im ersten Programmbeispiel haben wir nur eine dieser Stimmen benutzt. Später werden Sie noch lernen, wie die Klangfarben der einzelnen Stimmen geändert werden können. Nun sollen aber erst einmal alle drei Stimmen für uns singen.
Dieses Programm zeigt Ihnen, wie Noten für das Computerorchester übersetzt werden. Geben Sie es ein, und speichern Sie es danach auf DATASSETTE™ oder Diskette. Vor der Eingabe dieses Programms unbedingt NEW eingeben.

PROGRAMMBEISPIEL 2:

```
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH(2,200),L(2,200),C(2,200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
50 POKES+10,8:POKES+22,128:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
130 IFNM=0THEN250
140 WA=V(K):WB=WA-1:IFNM<0THENNM=-NM:WA=0:WB=0
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-256*HF%
210 IFDR%=1THENH(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:
I=I+1:GOTO120
220 FORJ=1TODR%-1:H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:
I=I+1:NEXT
230 H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WB
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
500 POKES+5,0:POKES+6,240
510 POKES+12,85:POKES+13,133
520 POKES+19,10:POKES+20,197
530 POKES+24,31
540 FORI=0TOIM
550 POKES,L(0,I):POKES+7,L(1,I):POKES+14,L(2,I)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,I):POKES+11,C(1,I):POKES+18,C(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38539,40830
610 DATA43258,45830,48556,51443
620 DATA54502,57743,61176,64814
1000 DATA594,594,594,596,596
1010 DATA1618,587,592,587,585,331,336
1020 DATA1097,583,585,585,585,587,587
1030 DATA1609,585,331,337,594,594,593
1040 DATA1618,594,596,594,592,587
1050 DATA1616,587,585,331,336,841,327
1060 DATA1607
1999 DATA0
2000 DATA583,585,583,583,327,329
2010 DATA1611,583,585,578,578,578
2020 DATA196,198,583,326,578
2030 DATA326,327,329,327,329,326,578,583
```

```
2040 DATA1606,582,322,324,582,587
2050 DATA329,327,1606,583
2060 DATA327,329,587,331,329
2070 DATA329,328,1609,578,834
2080 DATA324,322,327,585,1602
2999 DATA0
3000 DATA567,566,567,304,306,308,310
3010 DATA1591,567,311,310,567
3020 DATA306,304,299,308
3030 DATA304,171,176,306,291,551,306,308
3040 DATA310,308,310,306,295,297,299,304
3050 DATA1586,562,567,310,315,311
3060 DATA308,313,297
3070 DATA1586,567,560,311,309
3080 DATA308,309,306,308
3090 DATA1577,299,295,306,310,311,304
3100 DATA562,546,1575
3999 DATA0
```

Programmbeispiel 2 wird nun Zeile für Zeile erklärt. Nun interessiert uns, wie die drei Stimmen gesteuert werden.

ERKLÄRUNG VON PROGRAMMBEISPIEL 2:

Zeile(n)	Beschreibung
10	Setzen von S als Sound-Chip-Startadresse und Löschen aller Sound-Chipregister.
20	Dimensionieren der Felder für die Toninformation, 1/16-Takt pro Element.
30	Dimensionieren des Felds für die Basisfrequenzen der einzelnen Noten.
40	Speichern des Wellenform-Steuerbytes für die einzelnen Stimmen.
50	Impulsbreite für Stimme 2 eingeben. Obere Grenzfrequenz für Filter eingeben. Eingabe von Filterresonanz und Filter von Stimme 3.
60	Einlesen der Basisfrequenz der einzelnen Noten.
100	Beginn der Decodier-Schleife der einzelnen Stimmen.
110	Pointer-Initialisierung auf Steuerfeld.
120	Lesen der codierten Note.
130	Ist die codierte Note 0, dann nächste Stimme.
140	Wellenformsteuerung der Stimme. Bei Pause, Wellenformsteuerung auf 1 setzen.
150	Decodieren von Dauer und Oktave.
160	Note decodieren.

Zeile(n)	Beschreibung
170	Basisfrequenz für diese Note lesen.
180	Wenn höchste Oktave, Schleife für Division überspringen.
190	Basisfrequenz fortgesetzt durch 2 dividieren.
200	Bytes für "hohe Frequenz" und "niedrige Frequenz" lesen.
210	Wenn 16. Note, Steuerfeld setzen: hohe Frequenz, niedrige Frequenz und Wellenformsteuerung (Stimme ein).
220	Für alle Schläge außer dem letzten Steuerfeld eingeben: hohe Frequenz, niedrige Frequenz, Wellenformsteuerung (Stimme ein).
230	Für letzten Schlag in Steuerfeld eingeben: Hochfrequenz, Niedrigfrequenz, Wellenformsteuerung (Stimme aus).
240	Pointer auf Steuerfeld um 1 erhöhen. Nächste Note lesen.
250	Wenn länger als zuvor, Parameter /M rückstellen.
260	Zurücksprung für nächste Stimme.
500	ATTACK/DECAY für Stimme 1 eingeben (A = 0, D = 0). SUSTAIN/RELEASE für Stimme 1 eingeben (S = 15, R = 0).
510	ATTACK/DECAY für Stimme 2 eingeben (A = 5, D = 5). SUSTAIN/RELEASE für Stimme 2 eingeben (S = 8, R = 5).
520	ATTACK/DECAY für Stimme 3 eingeben (A = 0, D = 10). SUSTAIN/RELEASE für Stimme 3 eingeben (S = 12, R = 5).
530	Lautstärke 15 eingeben, Tiefpaßfilter ein.
540	Schleifenbeginn jedes 1/16-Taktes.
550	POKE der niedrigen Frequenz vom Steuerfeld für alle Stimmen.
560	POKE der hohen Frequenz vom Steuerfeld für alle Stimmen.
570	POKE der Wellenformsteuerung vom Steuerfeld für alle Stimmen.
580	Zeitschleife für 1/16-Takt und Rücksprung für nächsten 1/16-Takt.
590	Pause, dann Lautstärke abschalten.
600–620	Basisfrequenzdaten.
1000–1999	Daten für Stimme 1.
2000–2999	Daten für Stimme 2.
3000–3999	Daten für Stimme 3.

Die in den DATA-Anweisungen verwendeten Werte ergeben sich aus der Notentabelle in Anhang E und nachstehender Tabelle:

NOTENTYP	DAUER
1/16	128
1/8	256
Punktiert 1/8	384
1/4	512
1/4+1/16	640
Punktiert 1/4	768
1/2	1024
1/2+1/16	1152
1/2+1/8	1280
Punktiert 1/2	1536
Ganz	2048

Die Notenzahl der Notentabelle wird zu obiger Dauer addiert. Dann kann jede Note mit nur einer Nummer eingegeben werden, die dann von Ihrem Programm decodiert wird. Dies ist nur eine Art der Notencodierung. Sie können auch eine Methode wählen, die Sie für günstiger halten. Eine Note wird anhand folgender Gleichung codiert:

1) Dauer (in 16tel eines Taktes) multipliziert mit 8.
2) Das Ergebnis von 1) wird zu der gewählten Oktave addiert (0–7).
3) Das Ergebnis von 2) wird dann mit 16 multipliziert.
4) Die gewählte Note (0–11) zu dem Ergebnis von 3) addieren.

Das heißt:

$$((((D*8)+0)*16)+N)$$

Wobei D=Dauer, O=Oktave und N=Note.
Durch Verwendung eines negativen Werts für die Dauer (1/16 eines Taktes * 128) wird eine Pause programmiert.

STEUERN MEHRERER STIMMEN

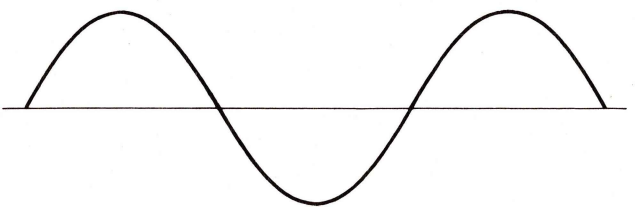
Wenn Sie mehrere Stimmen verwenden wollen, müssen Sie diese zeitlich miteinander koordinieren. In diesem Programm wird das wie folgt gelöst:

1) Teilung jedes Taktes in 16 Teile.
2) Speicherung der Inhalte der einzelnen Taktteile in drei getrennte Felder.

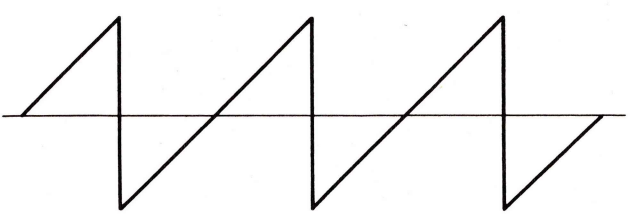
Die Bytes der hohen und niedrigen Frequenz werden durch Division der Frequenz der höchsten Oktave durch 2 (Zeile 180 und 190) berechnet. Das Byte für Wellenformsteuerung ist ein Startsignal für den Beginn einer Note oder das Halten einer bereits gespielten Note. Es ist auch das Endsignal für eine Note. Die Wellenformwahl wird einmal für jede Stimme in Zeile 40 durchgeführt.
Auch dies ist nur eine der Möglichkeiten, wie Sie mehrere Stimmen steuern können. Sie können Ihre eigene Methode erfinden. Sie sollten jedoch nun in der Lage sein, ein Notenblatt zu nehmen und die Noten der drei Stimmen herauszufinden.

ÄNDERN DER WELLENFORMEN

Die Klangfarbe eines Tones wird hauptsächlich durch seine Wellenform bestimmt. Wenn Sie einen Kieselstein ins Wasser werfen, dann verteilen sich die Wellen gleichmäßig über den Teich. Diese Wellen sehen fast wie die erste Wellenform aus, mit der wir uns befassen wollen: der sinusförmigen Welle, kurz Sinuswelle genannt (siehe nachstehende Abbildung).



Damit wir die praktische Anwendung nicht aus dem Auge verlieren, nehmen wir wieder das erste Programmbeispiel und untersuchen die unterschiedlichen Wellenformen. Die Änderungen können Sie nämlich leichter hören, wenn wir zunächst nur eine Stimme benutzen. Laden Sie das erste Musikprogramm von der DATASSETTE™ oder von der Diskette und führen Sie es aus. Dieses Programm arbeitet mit der Sägezahnwelle (siehe nachstehende Abbildung).

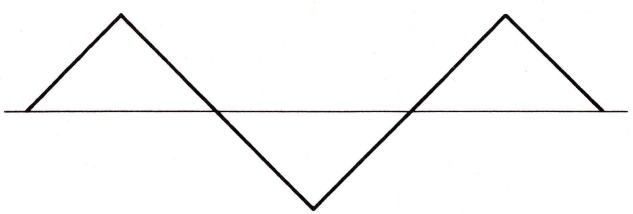


Ändern Sie den Notenstartwert in Zeile 70 von 33 in 17 und den Notenstopwert in Zeile 90 von 32 in 16 um. Das Programm muß nun folgendermaßen aussehen:

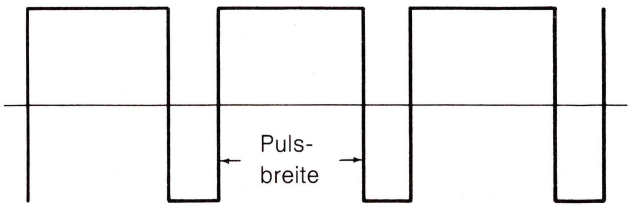
PROGRAMMBEISPIEL 3 (BEISPIEL 1, VERÄNDERT):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,9:POKES+6,0
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Starten Sie nun das Programm.
Beachten Sie, daß die Soundqualität nun anders ist, der Ton klingt nun viel hohler. Wir haben nämlich die Sägezahnwelle in eine Dreieckswelle umgewandelt (siehe nachstehende Abbildung).



Die dritte Wellenform wird variable Pulswelle genannt (siehe nachstehende Abbildung).



Dies ist eine Rechteckwelle, bei der Sie die Länge des Impulszyklus bestimmen können. Hierzu bestimmen Sie die Wellenhöhe. Für Stimme 1 geschieht dies mit den Registern 2 und 3: Register 2 enthält das niederwertige Byte der Pulsbreite (L_{pw} = 0 bis 255). Register 3 enthält die oberen vier Bits (H_{pw} = 0 bis 15).
Zusammen geben diese Register eine 12-Bit-Zahl für Ihre Pulsbreite an, die Sie anhand folgender Gleichung bestimmen können:

$$PW_n = H_{pw}*256 + L_{pw}$$

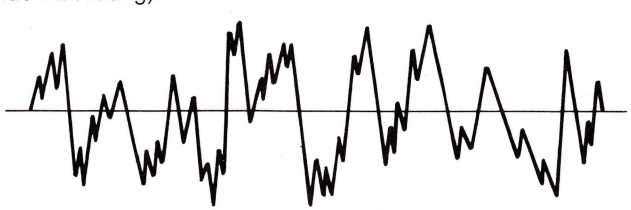
Die Pulsbreite wird durch folgende Gleichung bestimmt:

$$PW_{out} = (PW_n/40.95)\ \%$$

Hat PW_n den Wert 2048, dann ergibt sich eine Rechteckwelle. Dies bedeutet, daß Register 2 (L_{pw}) gleich 0 und Register 3 (H_{pw}) gleich 8 ist.
Fügen Sie nun in Ihr Programm diese Zeile ein:

```
15 POKES+3,8:POKES+2,0
```

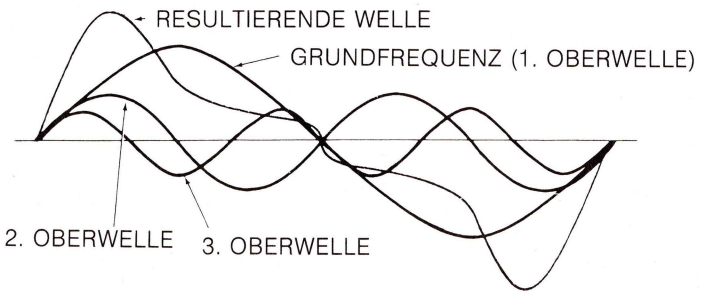
Ändern Sie dann den Startwert in Zeile 70 in 65 und den Stopwert in Zeile 90 in 64 um. Führen Sie dann das Programm aus. Ändern Sie nun die Pulsbreite (Register 3 in Zeile 15) von 8 in 1 um. Merken Sie den wesentlichen Unterschied in der Klangfarbe?
Eine weitere Wellenform, die erzeugt werden kann, ist ein weißes Rauschen (siehe nachstehende Abbildung).



Es wird meistens für Geräuscheffekte usw. benutzt. Um zu hören, wie es klingt, ändern Sie den Startwert in Zeile 70 auf 129 und den Stopwert in Zeile 90 auf 128 um.

VERSTÄNDNIS DER WELLENFORMEN

Eine gespielte Note besteht aus einer Grundfrequenz sowie den Oberwellen.
Die Grundfrequenz bestimmt die Tonhöhe. Oberwellen sind Sinuswellen mit Frequenzen, die ein ganzzahliges Vielfaches der Grundfrequenz sind. Eine Tonwelle besteht aus der Grundfrequenz und allen Oberwellen.

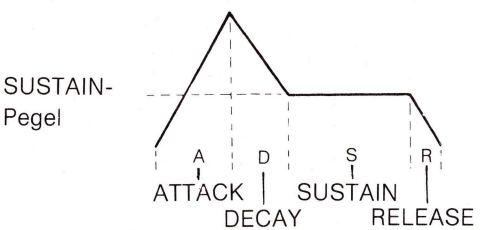


Theoretisch kann man sagen, daß die Grundfrequenz die Oberwelle Nr. 1 ist. Die Frequenz der zweiten Oberwelle entspricht zweimal der Grundfrequenz, die der dritten Oberwelle dreimal der Grundfrequenz usw. Die Anteile der einzelnen Oberwellen eines Tones bestimmen die Klangfarbe.
Ein akustisches Instrument wie z. B. eine Gitarre oder Violine hat eine komplizierte Oberwellenstruktur. Die Oberwellenstruktur kann sich auch beim Spielen einer einzelnen Note ändern. Sie haben nun bereits mit Wellenformen des COMMODORE-Synthesizers gespielt. Wir wollen uns nun anschauen, welche Rolle die Oberwellen bei Dreiecks-, Sägezahn- und Rechteckwellen spielen.
Eine Dreieckswelle enthält lediglich ungerade Oberwellen. Der Anteil jeder Oberwelle ist proportional zum Kehrwert des Quadrats der Oberwellenzahl. Die Oberwelle Nr. 3 ist also 1/9 leiser als Oberwelle Nr. 1, denn Oberwelle Nr. 3 zum Quadrat ist 9 (3x3), und der Kehrwert von 9 lautet 1/9.
Das entspricht der Beobachtung, daß eine Dreieckswelle einer Sinuswelle ähnlich ist.
Sägezahnwellen enthalten alle Oberwellen. Der Anteil jeder Oberwelle ist proportional zum Kehrwert der Oberwellenzahl. So ist die Oberwelle Nr. 2 z. B. einhalbmal so laut wie Oberwelle Nr. 1.

Die symmetrische Rechteckwelle enthält ungerade Oberwellen proportional zum Kehrwert der Oberwellenzahl. Andere Rechteckwellen haben verschiedene Oberwelleninhalte. Durch Änderung der Pulsbreite kann die Klangfarbe einer Rechteckwelle enorm geändert werden.
Durch sorgfältige Wahl der Wellenform können Sie mit einer Oberwellenstruktur beginnen, die fast so aussieht wie die des Tones, den Sie erzielen möchten. Zur Verfeinerung des Tons können Sie eine weitere Funktion des COMMODORE 64 benutzen, die "Filtern" genannt und später noch beschrieben wird.

HÜLLKURVENGENERATOR

Die Lautstärke eines Tons ändert sich ab dem Moment, ab dem Sie den Ton zuerst hören, bis er so schwach wird, daß Sie ihn schließlich nicht mehr hören können. Wenn ein Ton beginnt, so steigt er von der Null-Lautstärke bis zu Spitzenlautstärke. Dies nennt man ATTACK. Dann fällt er von der Spitze wieder zur mittleren Lautstärke ab. Dies nennt man DECAY. Der mittlere Bereich wird durch den SUSTAIN-Pegel beschrieben. Schließlich fällt die Note vom SUSTAIN-Pegel wieder auf Null-Lautstärke ab. Dies nennt man RELEASE. Die vier Phasen einer Note werden nachstehend dargestellt:



Die oben erwähnten Punkte bestimmen die verschiedenen Qualitäten und Einschränkungen der einzelnen Noten.
Die Parameter ATTACK/DECAY/SUSTAIN/RELEASE werden kurz ADSR genannt. Sie können durch Benutzen eines Satzes von Speicherplätzen im Sound-Chip gesteuert werden. Zunächst laden Sie wieder das erste Programmbeispiel, führen es aus und merken sich, wie es klingt. Ändern Sie Zeile 20 des Programms dann wie folgt:

PROGRAMMBEISPIEL 4 (BEISPIEL 1, AUFBEREITET):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Register 5 und 6 definieren ADSR für Stimme 1. ATTACK ist das obere Nybble von Register 5. Nybble ist ein halbes Byte, d. h. die unteren vier oder oberen vier Bits in jedem Register. DECAY ist das untere Nybble. Sie können für ATTACK eine beliebige Zahl zwischen 0 und 15 wählen, sie mit 16 multiplizieren und dann für DECAY eine beliebige Zahl zwischen 0 und 15 addieren. Die Werte, die diesen Zahlen entsprechen, werden nachstehend aufgeführt.
SUSTAIN-Pegel ist das obere Nybble von Register 6. 0 bis 15 stehen zur Verfügung. Hierdurch wird bestimmt, welchem Anteil der SUSTAIN-Pegel entspricht. Die RELEASE-Rate ist das untere Nybble von Register 6.
Nachstehend finden Sie die Bedeutungen der Werte für ATTACK, DECAY und RELEASE:

WERT	ATTACK (ZEIT/ZYKLUS)	DECAY/RELEASE (ZEIT/ZYKLUS)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Nachfolgend sehen Sie einige Einstellungen, die Sie in dem Programmbeispiel ausprobieren können. Probieren Sie diese aus und experimentieren Sie. Die Vielzahl an möglichen Tönen ist erstaunlich! Um den Ton einer Geige zu erzeugen, ändern Sie Zeile 20 wie folgt:

```
20 POKES+5,88:POKES+6,89:REM A=5;D=8;S=5;R=9
```

Ändern Sie die Wellenform in eine Dreieckswellenform um, und schon bekommen Sie durch folgende Zeilen das Geräusch eines Xylophons.

```
20 POKES+5,9:POKES+6,9:REM A=0;D=9;S=0;R=9
70 POKES+4,17
90 POKES+4,16: FORT=1TO50:NEXT
```

Ändern Sie die Wellenform nun in eine Rechteckwelle um, und erzeugen Sie durch folgende Zeilen einen Klavierklang.

```
15 POKES+3,8:POKES+2,0
20 POKES+5,9:POKES+6,0: REM A=0;D=9;S=0;R=0
70 POKES+4,65
90 POKES+4,64:FORT=1TO50:NEXT
```

Die aufregendsten Geräusche sind jedoch die, die nur der Synthesizer erzeugen kann, also keine Nachahmung akustischer Instrumente. Probieren Sie z. B. folgendes:

```
20 POKES+5,144:POKES+6,243:REM A=9;D=0;S=15;R=3
```

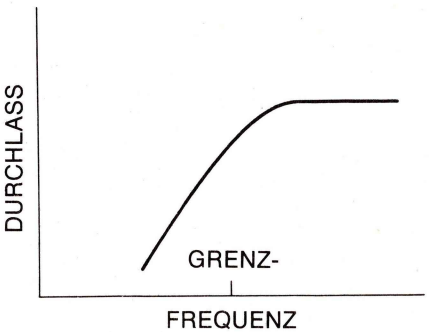
FILTERN

Der Oberwellengehalt einer Wellenform kann durch Verwendung eines Filters verändert werden. Der SID ist mit drei verschiedenen Filtern ausgerüstet. Sie können unabhängig voneinander oder auch kombiniert benutzt werden. Wir nehmen wieder unser einfaches Programmbeispiel und geben verschiedene Filtersteuerungen ein.
Fügen Sie Zeile 15 in das Programm ein, um die Grenzfrequenz des Filters einzugeben. Die Grenzfrequenz ist der Filterbezugspunkt. Die obere und untere Grenzfrequenz wird in Register 21 und 22 eingegeben. Um den Filter für Stimme 1 einzuschalten, Register 23 POKEn.
Nun Zeile 30 ändern, um anzuzeigen, daß ein Hochpaßfilter benutzt wird (siehe SID-Registerverzeichnis).

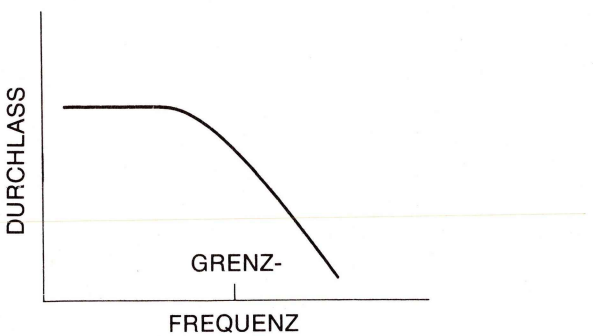
PROGRAMMBEISPIEL 5 (BEISPIEL 1, AUFBEREITET):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
15 POKES+22,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

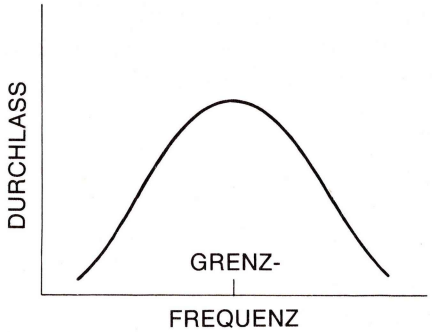
Probieren Sie dieses Programm nun aus. Beachten Sie, daß die niedrigeren Töne leiser sind und blechern klingen. Der Grund hierfür liegt darin, daß Sie einen Hochpaßfilter benutzen, der Frequenzen unterhalb der Grenzfrequenz dämpft.
Der SID des COMMODORE-Computers hat drei verschiedene Filter. Wir haben den Hochpaßfilter benutzt. Frequenzen bei oder über der Grenzfrequenz werden durchgelassen, die unter der Grenzfrequenz jedoch gedämpft.



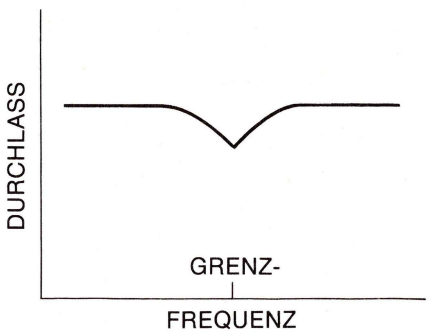
Der SID hat auch einen Tiefpaßfilter. Wie der Name schon sagt, läßt dieser Filter alle Frequenzen unter der Grenzfrequenz durch und dämpft die darüberliegenden ab.



Schließlich hat Chip SID auch noch einen Bandpaßfilter, der nur ein schmales Frequenzband um die Grenzfrequenz herum durchläßt und alle anderen Frequenzen dämpft.



Hoch- und Tiefpaßfilter können miteinander kombiniert werden, so daß eine Bandsperre entsteht, bei der nur die Grenzfrequenz gedämpft wird.



Zusätzlich zur gesamten Lautstärkenregelung bestimmt Register 24 auch noch, welche Filterart Sie benutzen wollen. Bit 6 steuert den Hochpaßfilter (0 = aus, 1 = ein), Bit 5 den Bandfilter und Bit 4 den Tiefpaßfilter. Die unteren drei Bits der Grenzfrequenz werden durch Register 21 (L_{cf}) bestimmt (L_{cf} = 0 bis 7). Die acht Bits der oberen Grenzfrequenz hingegen werden durch Register 22 (H_{cf}) bestimmt (H_{cf} = 0 bis 255).
Durch den Einsatz der Filter kann die Oberwellenstruktur jeder Wellenform geändert werden. Sie können also stets den Ton erzielen, den Sie möchten. Zusätzlich zur Filterung kann auch noch durch die ADSR-Hüllkurve ein interessanter Effekt erzielt werden.

FORTSCHRITTLICHE TECHNIKEN

Eine dynamische Änderung der Parameter des SID während einer Note zur Erzielung von interessanten und originellen Effekten ist möglich. Hierzu stehen digitalisierte Ausgaben vom Oszillator 3 und Hüllkurvengeber 3 in den Registern 27 bzw. 28 zur Verfügung.

Die Ausgabe von Oszillator 3 (Register 27) hängt direkt mit der gewählten Wellenform zusammen. Wurde die Sägezahnwelle von Oszillator 3 gewählt, dann liefert dieses Register eine Serie von inkrementierten (schrittweise erhöhten) Zahlen von 0 bis 255. Die Rate wird hierbei durch die Frequenz von Oszillator 3 bestimmt. Wurde die Dreieckswelle gewählt, dann wächst die Ausgabe von 0 bis 255 an und nimmt wieder bis auf 0 ab. Wurde die Pulswelle gewählt, dann springt die Ausgabe zwischen 0 und 255 hin und her. Durch Wahl des Rauschgenerators wird schließlich eine Anzahl von Zufalls-Zahlen angegeben. Wird Oszillator 3 für Modulation benutzt, soll die Ausgabe normalerweise NICHT hörbar sein. Durch das Setzen von Bit 7 von Register 24 wird die Audio-Ausgabe von Stimme 3 abgeschaltet. Register 27 gibt stets die sich ändernde Oszillatorausgabe wieder und wird nicht durch den Hüllkurvengeber (ADSR) beeinflußt.

Register 25 ermöglicht Ihnen den Zugriff auf die Ausgabe des Hüllkurvengebers von Oszillator 3. Hierbei gilt das gleiche wie bei der Ausgabe von Oszillator 3. Um dieses Register auslesen zu können, muß zunächst der Oszillator eingeschaltet werden.

Vibrato (eine schnelle Frequenzänderung) kann erzielt werden, indem man die Ausgabe von Oszillator 3 zu der Frequenz eines anderen Oszillators addiert. Dies wird im Programmbeispiel 6 gezeigt.

PROGRAMMBEISPIEL 6:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+3,8
40 POKES+5,41:POKES+6,89
50 POKES+14,117
60 POKES+18,16
70 POKES+24,143
80 READFR,DR
90 IFFR=0THENEND
100 POKES+4,65
110 FORT=1TODR*2
120 FQ=FR+PEEK(S+27)/2
130 HF=INT(FQ/256):LF=FQANDZ55
140 POKES+0,LF:POKES+1,HF
150 NEXT
160 POKES+4,64
170 GOTO80
500 DATA4817,2,5103,2,5407,2
510 DATA8583,4,5407,2,8583,4
520 DATA5407,4,8583,12,9634,2
530 DATA10207,2,10814,2,8583,2
540 DATA9634,4,10814,2,8583,2
550 DATA9634,4,8583,12
560 DATA0,0
```

Dieses Programm wird nachstehend Zeile für Zeile erklärt.

ERKLÄRUNG VON PROGRAMMBEISPIEL 6:

Zeile(n)	Beschreibung
10	Eingabe von S als Soundchip-Startadresse.
20	Löschen aller Soundchipregister.
30	Eingabe der Impulsbreite für Stimme 1.
40	Eingabe von ATTACK/DECAY für Stimme 1 (A=2, D=9). Eingabe von SUSTAIN/RELEASE für Stimme 1 (S=5, R=9).
50	Eingabe der niedrigen Frequenz für Stimme 3.
60	Eingabe der Dreieckswelle für Stimme 3.
70	Eingabe der Lautstärke 15, Abschalten der Audioausgabe von Stimme 3.
80	Lesen der Frequenz und Dauer der Note.
90	Ist die Frequenz gleich 0, dann Stop.
100	POKE Start-Impulswellenformsteuerung, Stimme 1.
110	Start-Zeitschleife für Dauer.
120	Neue Frequenz von Oszillatorausgang 3 lesen.
130	Hohe und niedrige Frequenz lesen.
140	POKE Hoch- und Niedrigfrequenz für Stimme 1.
150	Ende der Zeitschleife.
160	POKE Stop-Impulswellenformsteuerung, Stimme 1.
170	Zurück zur nächsten Note.
500–550	Frequenzen und Notenlängen des Musikstücks.
560	Nullen zeigen Ende des Stücks an.

Durch dynamische Effekte können außerdem auch noch eine Vielzahl an Geräuscheffekten erzielt werden. Folgendes Sirenenprogramm ändert z. B. dynamisch die Frequenzausgabe von Oszillator 1, indem es die Ausgabe der Dreieckswellenform von Oszillator 3 benutzt.

PROGRAMMBEISPIEL 7:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+14,5
40 POKES+18,16
50 POKES+3,1
60 POKES+24,143
70 POKES+6,240
80 POKES+4,65
90 FR=5389
100 FORT=1TO200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKES+0,LF:POKES+1,HF
140 NEXT
150 POKES+24,0
```

Dieses Programm wird nachstehend Zeile für Zeile erklärt.

ERKLÄRUNG VON PROGRAMMBEISPIEL 7:

Zeile(n)	Beschreibung
10	Eingabe von S als Soundchip-Startadresse.
20	Löschen der Soundchipregister.
30	Eingabe der niedrigen Frequenz für Stimme 3.
40	Eingabe der Dreieckswelle für Stimme 3.
50	Eingabe der Impulsbreite für Stimme 1.
60	Eingabe von Lautstärke 15, Abschalten der Audioausgabe von Stimme 3.
70	Eingabe von ATTACK/DECAY für Stimme 1 (S=15, R=0).
80	POKE Start-Impulswellenformsteuerung für Stimme 1.
90	Eingabe der niedrigsten Frequenz für Sirene.
100	Beginn der Zeitschleife.
110	Neue Frequenz mit Oszillatorausgabe 3 lesen.
120	Hohe und niedrige Frequenzen lesen.
130	POKE hohe und niedrige Frequenzen für Stimme 1.
140	Ende der Zeitschleife.
150	Abschalten der Lautstärke.

Die Rauschwellenform kann zur Erzeugung verschiedener Geräuscheffekte benutzt werden. Dieses Beispiel ahmt durch Verwendung einer gefilterten Rauschwellenform ein Händeklatschen nach:

PROGRAMMBEISPIEL 8:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,240:POKES+1,33
40 POKES+5,8
50 POKES+22,104
60 POKES+23,1
70 POKES+24,79
80 FORN=1TO15
90 POKES+4,129
100 FORT=1TO250:NEXT:POKES+4,128
110 FORT=1TO30:NEXT:NEXT
120 POKES+24,0
```

Dieses Beispiel wird nun Zeile für Zeile erklärt.

ERKLÄRUNG VON PROGRAMMBEISPIEL 8:

Zeile(n)	Beschreibung
10	Eingabe von S als Soundchip-Startadresse.
20	Löschen aller Soundchipregister.
30	Eingabe von hoher und niedriger Frequenz für Stimme 1.
40	Eingabe von ATTACK/DECAY für Stimme 1 (A=0, D=8).
50	Eingabe der oberen Grenzfrequenz für Filter.
60	Filter für Stimme 1 einschalten.
70	Eingabe von Lautstärke 15, Hochpaßfilter.
80	Zählt 15 Händeklatschen.
90	Eingabe für Anfang der Rauschwellenformsteuerung.
100	Warten, dann Stop der Rauschwellenformsteuerung.
110	Warten, dann nächstes Klatschen.
120	Lautstärke abschalten.

SYNCHRONISATION UND RINGMODULATION

Der 6581 SID ermöglicht Ihnen die Erstellung von komplexeren Oberwellenstrukturen durch Synchronisierung und Ringmodulation zweier Stimmen.
Synchronisierung ist im wesentlichen eine logische UND-Verbindung zweier Wellenformen. Ist eine dieser Wellenformen 0, dann ist die Ausgabe 0.
Durch folgendes Programmbeispiel wird ein Moskito nachgeahmt:

PROGRAMMBEISPIEL 9:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,100
40 POKES+5,219
50 POKES+15,28
60 POKES+24,15
70 POKES+4,19
80 FORT=1TO5000:NEXT
90 POKES+4,18
100 FORT=1TO1000:NEXT:POKES+24,0
```

Das Programm wird nun Zeile für Zeile erklärt.

ERKLÄRUNG VON PROGRAMMBEISPIEL 9:

Zeile(n)	Beschreibung
10	Eingabe von S als Soundchip-Startadresse.
20	Löschen der Soundchipregister.
30	Eingabe der hohen Frequenz für Stimme 1.
40	Eingabe von ATTACK/DECAY für Stimme 1 (A=13, D=11).
50	Eingabe der hohen Frequenz für Stimme 3.
60	Eingabe von Lautstärke 15.
70	Eingabe für Beginn der Dreiecks-/Synchronwellenformsteuerung für Stimme 1.
80	Zeitschleife.
90	Stop der Dreiecks-/Synchronwellenformsteuerung für Stimme 1.
100	Warten, dann Lautstärke abschalten.

Synchronisation wird in Zeile 70 eingeschaltet, in der die Bits 0, 1 und 4 von Register 4 gesetzt werden. Bit 1 schaltet die Synchronisation zwischen Stimme 1 und Stimme 3 ein. Die Bits 0 und 4 dienen wie gewöhnlich der Austastung von Stimme 1 und dem Setzen der Dreieckswellenform.

Bei der Ringmodulation (für Stimme 1 durch Setzen von Bit 3 des Registers 4 in Zeile 70) wird die Dreiecksausgabe von Oszillator 1 durch eine "ringmodulierte" Kombination von Oszillator 1 und 3 ersetzt. Hierdurch entsteht eine nicht-harmonische Oberwellenstruktur, mit der z. B. Klingel- oder Gonggeräusche nachgeahmt werden können. Durch folgendes Programm wird ein Glockenspiel imitiert:

PROGRAMMBEISPIEL 10:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,130
40 POKES+5,9
50 POKES+15,30
60 POKES+24,15
70 FORL=1TO12:POKES+4,21
80 FORT=1TO1000:NEXT:POKES+4,20
90 FORT=1TO1000:NEXT:NEXT
```

Das Programm wird nun Zeile für Zeile erklärt.

ERKLÄRUNG VON PROGRAMMBEISPIEL 10:

Zeile(n)	Beschreibung
10	Setzen von S als Soundchip-Startadresse.
20	Löschen der Soundchipregister.
30	Setzen der hohen Frequenz für Stimme 1.
40	Setzen von ATTACK/DECAY für Stimme 1 (A=0, D=9).
50	Setzen der hohen Frequenz für Stimme 3.
60	Setzen von Lautstärke 15.
70	Zählen der Klingelimpulse, Setzen von Start für Dreieck, Ringmodulation, Wellenformsteuerung für Stimme 1.
80	Zeitschleife, Stop der Dreieckswellenform, Ringmodulation.
90	Zeitschleife, nächster Klingelimpuls.

Die Effekte, die Sie durch Setzen der Parameter des SID des COMMODORE 64 erzielen können, sind zahllos und breit gefächert. Nur durch Experimentieren können Sie die Einsatzmöglichkeiten des Gerätes herausfinden und schätzen-lernen. Die in diesem Kapitel gegebenen Beispiele stellen wirklich nur die oberste Spitze eines Eisbergs dar.

KAPITEL 5

MASCHINEN-SPRACHE

- Was ist Maschinensprache?
- Wie schreibt man Programme in Maschinensprache?
- Hexadezimaldarstellung
- Adressierart
- Indizieren
- Unterprogramme
- Hinweise für den Anfänger
- Vorbereitungen für eine große Aufgabe
- Befehlssatz des Mikroprozessors MCS6510
- Speicherverwaltung beim COMMODORE 64
- KERNAL
- KERNAL-Funktionen nach Einschalten der Stromversorgung
- Arbeiten mit Maschinensprache und BASIC
- COMMODORE 64-Memory Map (Speicherbelegung)

WAS IST MASCHINENSPRACHE?

Das Herz jedes Mikrocomputers ist ein Mikroprozessor. Hierbei handelt es sich um einen Spezial-Mikro-Chip, der das "Gehirn" des Computers ausmacht. Der COMMODORE 64 bildet hierbei keine Ausnahme. Jeder Mikroprozessor versteht seine speziellen Befehle, die man unter dem Begriff Maschinensprache zusammenfaßt. Maschinensprache ist die einzige Programmiersprache, die Ihr COMMODORE 64 versteht. Es ist sozusagen die Muttersprache der Maschine.

Wenn Maschinensprache die einzige Sprache ist, die der COMMODORE 64 verstehen kann, wie kann er dann die CBM-BASIC-Programmiersprache verstehen? CBM-BASIC ist nicht die Maschinensprache vom COMMODORE 64. Wie kann der COMMODORE 64 dann BASIC-Anweisungen wie z. B. PRINT und GOTO verstehen?

Um diese Frage zu beantworten, müssen wir zunächst einmal klarlegen, was im COMMODORE 64 passiert. Außer dem Gehirn, dem Mikroprozessor des COMMODORE 64, gibt es noch das Maschinensprache-Programm, das in einem speziellen Speicher abgelegt ist und nicht geändert werden kann. Und, was weitaus wichtiger ist, es verschwindet nicht beim Abschalten des Geräts so wie von Ihnen geschriebene Programme. Dieses Maschinenprogramm wird BETRIEBSSYSTEM genannt. Der COMMODORE 64 weiß nach dem Einschalten, was er zu tun hat, weil das BETRIEBSSYSTEM automatisch startet.

Das BETRIEBSSYSTEM "organisiert" den Speicherbereich für die verschiedenen Aufgaben. Außerdem erkennt es, welche Tasten Sie auf der Tastatur angeschlagen haben, und zeigt diese auf dem Bildschirm an. Das BETRIEBSSYSTEM ist noch für zahlreiche weitere Funktionen zuständig. Das BETRIEBSSYSTEM kann man sich also als eine Art "Intelligenz und Persönlichkeit" des COMMODORE 64 vorstellen. Nach dem Einschalten des Geräts übernimmt das BETRIEBSSYSTEM also die Kontrolle. Nach Erledigung seiner Aufgabe zeigt es an:

```
READY.
■
```

Das BETRIEBSSYSTEM des COMMODORE 64 ermöglicht Ihnen danach eine Eingabe über die Tastatur und ein Arbeiten mit dem Bildschirm-Editor. Der Bildschirm-Editor ermöglicht ein Bewegen des Cursors, Löschen, Einfügen usw. und ist nur ein kleiner Teil des gesamten Betriebssystems.
Alle in CBM-BASIC zur Verfügung stehenden Befehle werden einfach durch ein weiteres umfassendes Maschinensprache-Programm erkannt. Dieses umfangreiche Programm führt dann je nach BASIC-Befehl den entsprechenden Teil Maschinensprache aus. Dieses Programm nennt man BASIC-INTERPRETER, da es die Befehle nacheinander interpretiert, bis es auf einen nicht zu verstehenden Befehl trifft. Dann wird die vertraute Meldung angezeigt:

```
?SYNTAX ERROR

READY.
■
```

WIE SIEHT DER MASCHINENCODE AUS?

Zum Ändern von Speicherplätzen müssen Sie mit den PEEK- und POKE-Anweisungen von CBM-BASIC vertraut sein. Sie haben diese sicherlich schon für die Graphikdarstellung sowie für Soundeffekte benutzt. Jeder Speicherplatz wird durch eine eigene Nummer gekennzeichnet. Diese Nummer ist auch als "Adresse" eines Speicherplatzes bekannt. Wenn Sie sich den Speicher des COMMODORE 64 als eine Straße mit mehreren Häusern vorstellen, dann ist die Zahl an jeder Tür die Adresse. Nun wollen wir feststellen, welche Häuser für welche Zwecke benutzt werden.

EINFACHE LISTE DER SPEICHERBELEGUNG DES COMMODORE 64

ADRESSE	BESCHREIBUNG
0 & 1	– 6510-Register
2 bis: 1023	– Speicheranfang – Vom Betriebssystem beanspruchter Speicher
1024 bis: 2039	– Bildschirmspeicher
2040 bis: 2047	– SPRITE-Pointer
2048 bis: 40959	– Dies ist Ihr Speicher. Hier sind Ihre BASIC- und/oder Maschinensprachenprogramme gespeichert.
40960 bis: 49151	– 8K-CBM-BASIC-Interpreter
49152 bis: 53247	– Besonderer RAM-Programmbereich
53248 bis: 53294	– VIC-II-Register
55296 bis: 56296	– Farb-RAM
56320 bis: 57343	– Ein-/Ausgaberegister
57344 bis: 65535	– 8K-CBM-KERNAL-Betriebssystem

Keine Sorge, wenn Sie jetzt noch nicht richtig die Beschreibung der einzelnen Speicherteile verstehen. Dies wird später noch genau behandelt.
Maschinensprache-Programme bestehen aus Anweisungen mit oder ohne Operanden (Parameter). Jede Anweisung benötigt einen Speicherplatz, und in den ein oder zwei Adressen hinter der Anweisung ist der Operand enthalten.
In Ihren BASIC-Programmen benötigen Wörter wie z. B. PRINT und GOTO nur je einen Speicherplatz (und nicht etwa einen Speicherplatz für jedes einzelne Zeichen). Der Inhalt des Speicherplatzes, der ein bestimmtes BASIC-Schlüsselwort darstellt, wird "token" genannt. In der Maschinensprache gibt es verschiedene "tokens" für die verschiedenen Anweisungen, die ebenfalls nur ein Byte beanspruchen (Speicherplatz = Byte).
Maschinensprache-Anweisungen sind sehr einfach. Aus diesem Grund kann man mit einer einzelnen Anweisung auch nicht sehr viel anfangen. Durch Maschinensprache-Anweisungen wird entweder der Inhalt eines Speicherplatzes oder eines der internen Register im Mikroprozessor geändert. Diese internen Register bilden die Grundlage der Maschinensprache.

DIE REGISTER IM MIKROPROZESSOR 6510

AKKUMULATOR

Das ist DAS Register des Mikroprozessors. Durch verschiedene Maschinensprache-Anweisungen können Sie den Inhalt eines Speicherplatzes im Akkumulator abspeichern, den Akkumulatorinhalt in einen anderen Speicherplatz kopieren, die Akkumulatorinhalte oder die Registerinhalte direkt und ohne Beeinflussung anderer Speicherplätze ändern. Der Akkumulator ist außerdem das einzige Register, in dem Rechenoperationen ausgeführt werden können.

INDEXREGISTER X

Dies ist ein sehr wichtiges Register. Es gibt Anweisungen für nahezu alle Operationen, die mit dem Akkumulator möglich sind. Es gibt jedoch auch Anweisungen, die nur für das X-Register wirksam sind. Die verschiedenen Maschinensprache-Anweisungen ermöglichen Ihnen ein Kopieren eines Speicherplatzinhalts in das X-Register, ein Kopieren des X-Registerinhalts in einen Speicherplatz sowie die direkte Änderung des X- und anderer Register ohne Beeinflussung anderer Speicherplätze.

INDEXREGISTER Y

Dies ist ebenfalls ein sehr wichtiges Register. Es gibt Anweisungen für nahezu alle Operationen, die mit dem Akkumulator und dem X-Register möglich sind. Es gibt jedoch auch Anweisungen, die nur für Register Y wirksam sind.
Verschiedene Maschinensprache-Anweisungen ermöglichen Ihnen ein Kopieren eines Speicherplatzinhaltes in Register Y, das Kopieren des Y-Registerinhalts in einen Speicherplatz sowie die direkte Änderung des Y- oder anderer Register ohne Beeinflussung der übrigen Speicherplätze.

STATUSREGISTER

Dieses Register besteht aus acht "Flags" (Flag = Anzeige, ob ein Ereignis eingetreten ist oder nicht).

PROGRAMMZÄHLER

Dieser enthält die Adresse der derzeitig ausgeführten Maschinensprache-Anweisung. Da das Betriebssystem beim COMMODORE 64 (und übrigens auch bei allen anderen Computern) ständig aktiv ist, ändert sich auch der Programmzähler ständig. Er kann nur zusammen mit dem Mikroprozessor gestoppt werden.

STAPELZEIGER (STACKPOINTER)

Dieses Register enthält die Adresse des ersten freien Stapelplatzes. Der Stapel (Stack) wird für die temporäre Speicherung von Maschinensprache-Programmen sowie vom Computer beansprucht.

EIN-/AUSGABEPORT

Dieses Register belegt die Speicherplätze 0 (Datenrichtungs-Register) und 1 (Datenregister). Es handelt sich um ein 8-Bit-Ein-/Ausgabeport. Beim COMMODORE 64 wird dieses Register zur Speicherverwaltung benutzt. Der Chip kann dann mehr als 64 K RAM- und ROM-Speicherkapazität kontrollieren.
Die Einzelheiten dieser Register werden hier nicht erklärt. Dies erfolgt später bei Erklärung der jeweiligen Funktionsweise.

WIE SCHREIBT MAN MASCHINENSPRACHE-PROGRAMME?

Da der COMMODORE 64 nicht die Möglichkeit zum Schreiben und Editieren von Maschinensprache-Programmen bietet, müssen Sie hierzu entweder ein Programm benutzen oder selbst ein BASIC-Programm schreiben, das Ihnen das Schreiben in Maschinensprache ermöglicht.
Die am weitesten verbreitete Methode zum Schreiben von Maschinensprache-Programmen sind Assembler-Programme. Diese Software-Pakete ermöglichen Ihnen das Schreiben von Maschinensprache-Anweisungen in standardmäßigem Mnemonik-Format. Hierdurch lassen sich Programme in Maschinensprache wesentlich leichter lesen als eine komplizierte Zahlenreihe. Wir fassen zusammen: Ein Programm, das Ihnen das Schreiben von Maschinensprache-Programmen im Mnemonik-Format ermöglicht, wird Assembler genannt. Entsprechend nennt man ein Programm, bei dem ein Maschinensprache-Programm im Mnemonik-Format angezeigt wird, Disassembler. Für den COMMODORE 64 steht eine Diskette mit einem Maschinensprache-Monitor (mit Assembler/Disassembler usw.) zur Verfügung:

MONITOR 64

Diese Diskette MONITOR 64, die Sie bei Ihrem COMMODORE-Händler bekommen, enthält ein Programm, das Ihnen den Übergang von CBM-BASIC in die Maschinensprache ermöglicht. Auf diese Weise kann der Inhalt der internen Register des Mikroprozessors 6510 sowie einzelne Speicherbereiche auf dem Bildschirm angezeigt und mit Hilfe des Bildschirm-Editors aufbereitet werden. Außerdem gehören hierzu Assembler, Disassembler sowie weitere Funktionen, die Ihnen das Schreiben und Editieren von Maschinensprache-Programmen erleichtern. Sie müssen nicht unbedingt einen Assembler zum Schreiben in Maschinensprache benutzen, er erleichtert Ihnen diese Aufgabe jedoch wesentlich. Für das Schreiben von Programmen in Maschinensprache empfehlen wir mit Nachdruck den Kauf eines Assemblers, da Sie sonst das Maschinensprache-Programm in den Speicher POKEn müssen.
In dieser Anleitung werden ab jetzt die Beispiele in dem Format von MONITOR 64 gegeben. Fast alle Assembler-Formate sind gleich. Aus diesem Grund werden die hier gezeigten Maschinensprache-Beispiele wahrscheinlich mit beliebigen Assemblern kompatibel sein. Bevor wir jedoch weitere Merkmale von MONITOR 64 besprechen, müssen wir zunächst das Hexadezimal-Zahlensystem erklären.

HEXADEZIMALDARSTELLUNG

Die Hexadezimaldarstellung wird von den meisten Maschinensprache-Programmierern benutzt, wenn es sich um eine Zahl oder Adresse im Maschinensprache-Programm handelt.
Einige Assembler erlauben den Bezug auf Adressen und Zahlen in Dezimaldarstellung (Basis 10), Binärdarstellung (Basis 2) oder sogar Oktal (Basis 8). Natürlich ist auch die Hexadezimaldarstellung (Basis 16) (oder, wie viele einfach sagen, "Hex") möglich. Der Assembler übernimmt für Sie die Umwandlungen.
Hexadezimaldarstellung sieht zunächst kompliziert aus, wird jedoch, wie die meisten Dinge, nach etwas Übung schnell verständlich.
Dezimalzahlen (Zehnersystem, also Basis 10) sind Zahlen aus dem Bereich von 0 bis 9. Binärzahlen (Basis 2) haben Ziffern von 0 bis 1. Die größte Ziffer ist immer gleich der Basis minus 1. DIES GILT FÜR ALLE ZAHLENBASEN. Entsprechend müssen Hexadezimalzahlen Ziffern zwischen 0 und 15 haben. Für die Zahlen 10 bis 15 stehen jedoch keine einstelligen Ziffern zur Verfügung. Aus diesem Grund werden statt dessen die ersten sechs Buchstaben des Alphabets benutzt:

DEZIMAL	HEXADEZIMAL	BINÄR
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	A	00001010
11	B	00001011
12	C	00001100
13	D	00001101
14	E	00001110
15	F	00001111
16	10	00010000

Nachstehend sehen Sie ein weiteres Beispiel, wie eine Dezimalzahl (Basis 10) aufgebaut wird:

Basis mit steigender Potenz:	10^3	10^2	10^1	10^0
Entspricht:	1000	100	10	1
Bei 4569 (Basis 10)	4	5	6	9

$$=(4\times1000)+(5\times100)+(6\times10)+(9\times1)$$

Nun wollen wir uns ansehen, wie die Basis 16 (Hexadezimalzahl) entsteht:

Basis mit steigender Potenz:	16^3	16^2	16^1	16^0
Entspricht:	4096	256	16	1
Bei 11D9 (Basis 16)	1	1	D	9

$$=1\times4096+1\times256+13\times16+9\times1$$

Daher ist 4569 (Basis 10) = 11D9 (Basis 16).

Adressierbare Speicherplätze liegen zwischen 0–65535 (wie bereits erläutert). Dieser Bereich lautet in Hexadezimaldarstellung 0–FFFF.
Normalerweise steht vor Hexadezimalzahlen ein Dollarzeichen ($). Auf diese Weise wird es von Dezimalzahlen unterschieden. Wir wollen uns nun mit MONITOR 64 einige Hexadezimalzahlen anschauen, indem wir den Inhalt eines Speicherplatzes anzeigen lassen. Geben Sie folgendes über die Tastatur ein:

```
B■
   PC  SR  AC  XR  YR  SP
.; 0401  32  04  5E  00  F6 (Kann unterschiedlich sein)
```

Wird nun

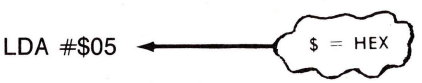
.M 0000 0020 (and press RETURN).

eingegeben und RETURN gedrückt, sehen Sie Reihen von neun Hexadezimalzahlen. Die erste vierstellige Zahl ist die Adresse des ersten in dieser Reihe gezeigten Speicherbytes. Die anderen acht Zahlen sind die Speicherplatzinhalte, beginnend bei der Startadresse.
Sie sollten tatsächlich lernen, hexadezimal zu "denken". Dies ist nicht allzu schwierig, da Sie sich keine Gedanken über die Rückumwandlung in Dezimaldarstellung machen müssen.
Wenn Sie z. B. sagen, ein bestimmter Wert ist bei $14ED anstatt 5357 gespeichert, so sollte dies für Sie keinen Unterschied machen.

DIE ERSTE MASCHINENSPRACHE-ANWEISUNG

LDA – SPEICHERÜBERTRAGUNG ZUM AKKUMULATOR

Bei der Assembler-Sprache 6510 sind Mnemoniks stets drei Zeichen. LDA bedeutet "... zum Akkumulator übertragen". Was in den Akkumulator geladen werden soll, wird durch den/die Parameter der entsprechenden Anweisung bestimmt. Der Assembler "weiß", welcher Befehl durch die einzelnen Mnemoniks dargestellt ist. Beim "Assemblieren" einer Anweisung werden einfach die entsprechenden Zahlencodes und die entsprechenden Parameter in den Speicher (ab der angegebenen Adresse) geladen.
Einige Assembler zeigen Fehlermeldungen oder Warnungen an, wenn Sie einen Ausdruck assemblieren wollen, der bei diesem Assembler oder beim Mikroprozessor 6510 nicht möglich ist.
Wird vor den Parameter der Anweisung das Symbol "#" gesetzt, so bedeutet dies, daß in das in der Anweisung angegebene Register der hinter "#" stehende Wert geladen werden soll. Z. B.:



Durch diese Anweisung wird $05 (Dezimal 5) in das Akkumulatorregister geladen. Der Assembler lädt in die entsprechende Adresse für diese Anweisung $A9 (gleich Zahlencode für diese besondere Anweisung in dieser Betriebsart). $05 wird in den nächsten Platz nach dem Platz mit der Anweisung ($A9) geladen.
Steht vor einem Anweisungsparameter ein "#", d. h., ist der Parameter ein "Wert" und nicht ein Verweis auf einen Speicherplatz oder ein Register, dann benutzt man den Unmittelbar-Modus. Hierzu wollen wir den Vergleich mit einem anderen Modus machen:
Soll der Inhalt von Speicherplatz $102E in den Akkumulator geladen werden, benutzen Sie den "Absolut-Modus" der Anweisung:

```
LDA $102E
```

Der Assembler kann zwischen den zwei verschiedenen Modi unterscheiden, da beim letzteren vor dem Parameter kein "#" steht. Der Mikroprozessor 6510 kann zwischen dem Unmittelbar- und dem Absolut-Modus der LDA-Anweisung unterscheiden, da diese unterschiedliche Zahlencodes haben. Die Zahlencodes für LDA im Sofort-Betrieb lauten $A9 und im Absolut-Betrieb $AD.
Aus den Mnemoniks von Anweisungen ergibt sich normalerweise bereits, was diese Anweisungen bedeuten. Was, glauben Sie, bedeutet z. B. LDX?
Wer hat da gesagt "Lade Register X mit ..."? Sie sind Klassenbester.

Aber auch wer dies nicht gleich wußte, braucht sich keine Sorgen zu machen. Zum Erlernen der Maschinensprache braucht man Geduld.
Die verschiedenen internen Register kann man sich als spezielle Speicherplätze vorstellen. In ihnen kann nämlich auch ein Byte abgespeichert werden. Wir brauchen das Binär-Zahlensystem (Basis 2) nicht zu erklären, da hierfür die gleichen Prinzipien wie bei der Hexadezimal- und Dezimaldarstellung gelten. Ein "Bit" ist jedoch eine Binärstelle, und acht Bits ergeben ein Byte! Die max. mögliche Zahl in einem Byte ist daher die größtmögliche achtstellige Binärzahl. Diese Zahl ist 11111111 (binär), was $FF (hexadezimal) oder 255 (dezimal) entspricht. Sie haben sich sicherlich schon gewundert, warum in einem Speicherplatz nur Zahlen von 0 bis 255 eingegeben werden können. Versuchen Sie, POKE 7680,260 (BASIC-Anweisung "die Zahl 260 in den Speicherplatz 7680 speichern"), weiß der BASIC-Interpreter, daß nur die Zahlen 0 bis 255 zulässig sind, und der COMMODORE 64 zeigt dann an:

```
?ILLEGAL QUANTITY ERROR

READY.
■
```

Wenn ein Byte auf $FF (hex) begrenzt ist, wie wird dann der Adreßparameter in der Absolut-Anweisung "LDA$102E" im Speicher ausgedrückt? Er wird in zwei Bytes abgelegt (da er natürlich nicht in eines paßt). Die unteren (rechten) zwei Stellen der Hexadezimaladresse bilden das "untere Byte" der Adresse (Low Byte). Entsprechend bilden die oberen (äußerst linken) zwei Stellen das "obere Byte" (High Byte). Für den 6510 müssen die Adressen zunächst durch das untere und danach durch das obere Byte angegeben werden. Die Anweisung "LDA$102E" wird also im Speicher durch drei aufeinanderfolgende Werte dargestellt.

```
$AD, $2E, $10
```

Nun müssen Sie nur noch eine weitere Anweisung lernen, und Sie können Ihr erstes Programm schreiben. Diese Anweisung ist BRK. Eine genaue Erklärung dieser Anweisung finden Sie im "Programmierhandbuch zum M.O.S. 6502". Sie können sich dies nun als END-Anweisung in Maschinensprache vorstellen.
Wenn wir mit MONITOR 64 ein Programm schreiben und die BRK-Anweisung am Ende eingeben, kehrt das Programm nach der Ausführung in den Monitor zurück. Dies passiert nicht, wenn Ihr Programm einen Fehler enthält oder wenn die BRK-Anweisung nicht erreicht wird.

SCHREIBEN DES ERSTEN PROGRAMMS

Wenn Sie schon über die POKE-Anweisung in BASIC Zeichen auf den Bildschirm gebracht haben, werden Sie wissen, daß die Zeichen-Codes zum POKEn sich von den CBM-ASCII-Zeichenwerten unterscheiden. Geben Sie z.B.:

```
PRINT ASC("A")
```

ein und drücken RETURN , dann antwortet der COMMODORE 64 mit:

```
65

READY.
■
```

Um jedoch auf den Bildschirm ein "A" zu POKEn (Code = 1), wird folgendes eingegeben:

SHIFT CLR /HOME Zum Löschen des Bildschirms

POKE 1024, 1 (und RETURN) (1024 ist der Bildschirmspeicheranfang)

Das "P" der POKE-Anweisung muß nun ein "A" sein.
Nun wollen wir dies in Maschinensprache probieren. Geben Sie folgendes im MONITOR 64 ein:
(Der Cursor muß nun neben einem "Punkt" blinken.)

.A 1400 LDA #$01 (und RETURN drücken)

Der COMMODORE 64 zeigt an:

```
.A 1400 LDA #$01
.A 1402 ■
```

Über die Tastatur eingeben:

```
.A 1402 STA $0400
```

(Über die STA-Anweisung wird der Akkumulatorinhalt an einen bestimmten Speicherplatz gespeichert.)
Der COMMODORE 64 zeigt nun an:

```
.A 1405 ■
```

Geben Sie nun folgendes ein:

```
.A 1405 BRK
```

Den Bildschirm löschen und

```
G 1400
```

eingeben.
Das "G" muß sich nun in "A" verwandeln.
Und schon haben Sie Ihr erstes Programm in Maschinensprache geschrieben. Durch dieses Programm wird ein Zeichen "A" in den ersten Bildschirmspeicherplatz geladen.
Wir wollen uns nun mit weiteren Anweisungen und Funktionsweisen befassen.

ADRESSIERART

ZERO-PAGE

Wie bereits erklärt, werden Absolutadressen durch ein oberes und unteres Byte ausgedrückt. Das obere Byte bezeichnet man oft auch als Speicherseite (Page). Z. B. ist Adresse $1637 in Seite $16 (22), $0277 in Seite $02 (2). Es gibt jedoch noch eine besondere Art der Adressierung, die Zero-Page-Adressierung. Wie der Name bereits besagt, bezieht es sich auf das Adressieren von Speicherplätzen auf der Zero-Page (Seite Null).
Diese Adressen haben daher STETS ein oberes Byte null. Die Zero-Page-Adressierung benötigt daher zur Beschreibung der Adresse lediglich ein Byte, und nicht wie bei der Absolut-Adressierung zwei Bytes. Die Zero-Page-Adressierung weist den Mikroprozessor an, die obere Adresse als null anzusehen. Durch diese Adressierart ist daher ein Bezug auf Speicherplätze möglich, deren Adressen zwischen $0000 und $00FF liegen. Dies scheint jetzt noch nicht allzu wichtig zu sein, Sie werden das Prinzip der Zero-Page-Adressierung jedoch bald brauchen.

STAPEL (STACK)

Der Mikroprozessor 6510 hat einen Stack. Dieser wird für die temporäre Speicherung vom Programmierer und auch vom Mikroprozessor selber benutzt. Er "merkt sich" z. B. auch bestimmte Reihenfolgen. Die GOSUB-Anweisung in BASIC, die den Aufruf eines Unterprogramms ermöglicht, muß sich z. B. die Aufrufebene merken. Erfolgt dann im Unterprogramm die RETURN-Anweisung, dann weiß der BASIC-Interpreter, an welche Stelle er zurückgehen und die Ausführung fortsetzen muß. Wird die GOSUB-Anweisung in einem Programm vom BASIC-Interpreter gelesen, dann gibt er seine derzeitige Position vor dem Übergang zum Unterprogramm in den Stack ein. Bei der Ausführung von RETURN wird diese Information wieder vom Stack gelesen.
Der Interpreter weiß also nun, an welcher Stelle er sich vor dem Unterprogramm-Aufruf befunden hat. Der Interpreter arbeitet z. B. mit der **PHA**-Anweisung (Speicherung des Akkumulators im Stackregister) und mit **PLA** (Speicherung eines Stackwerts im Akkumulator). Auch das Statusregister kann auf diese Weise über die Anweisungen **PHP** bzw. **PLP** gespeichert werden. Der Stack ist 256 Byte lang und befindet sich auf Speicherseite 1. Er liegt im Adressenbereich von $0100 bis $01FF. Er wird rückwärts verwaltet, d. h., die erste Position im Stack liegt bei $01FF und die letzte bei $0100. Ein weiteres Register im Mikroprozessor 6510 nennt man Stapelzeiger (Stackpointer). Dieser zeigt stets auf den nächsten verfügbaren Stapelplatz. Eine Eingabe erfolgt daher stets in den Stapelplatz, auf den der Stapelzeiger zeigt. Der Zeiger wird dann zur nächsten Position (in Rückwärtsrichtung) bewegt. Wird eine Information vom Speicher abgerufen, dann wird der Stapelzeiger inkrementiert, und das vom Zeiger angegebene Byte wird in das jeweilige Register gesetzt. Nun haben wir also unmittelbare, Zero-Page- und Absolut-Anweisungen behandelt. Dabei haben wir uns auch schon ein bißchen mit der implizierten Adressierung beschäftigt. Hierunter versteht man, daß eine Information durch die Anweisung selbst impliziert wird, d. h., auf welche Register, Kennzeichen und Speicher sich die Anweisung bezieht. Die behandelten Beispiele sind PHA, PLA, PHP und PLP, die sich auf Stapelverarbeitung, Akkumulator bzw. Statusregister beziehen.

Anmerkung: Nachfolgend steht X für X-Register, A für Akkumulator, Y für Y-Indexregister, S für Stapelzeiger und P für Prozessorstatus.

INDIZIEREN

Das Indizieren ist beim Arbeiten mit dem Mikroprozessor 6510 von außerordentlicher Bedeutung. Hierunter versteht man das "Erstellen einer Adresse aus einer Basisadresse plus Inhalt von X- oder Y-Indexregister".
Enthält X z. B. $05 und der Mikroprozessor führt eine LDA-Anweisung im "absoluten X-indizierten Modus" mit der Basisadresse (z. B. $9000) durch, dann lautet der tatsächliche Platz, der in Register A geladen wird, $9000 + $05 = $9005. Das Mnemonik-Format einer absoluten indizierten Anweisung entspricht dem einer absoluten Anweisung. Der Unterschied liegt lediglich darin, daß die Indexangabe ",X" oder ",Y" zur Adresse addiert wird.

BEISPIEL:

```
LDA $9000,X
```

Beim Mikroprozessor 6510 stehen für die Adressierung die Methoden absolut indiziert, Zero-Page-indiziert, indirekt indiziert sowie indiziert indirekt zur Verfügung.

INDIREKT INDIZIERT

Hierbei ist als Index nur das Y-Register möglich. Die tatsächliche Adresse darf nur in der Zero-Page liegen, und die Anweisungsart nennt man indirekt, weil die Zero-Page-Adresse der Anweisung das untere Byte der tatsächlichen Adresse und das darauffolgende Byte das obere Byte enthält.

BEISPIEL:

Nehmen wir z. B. an, daß Adresse $01 $45 und Adresse $02 $1E enthält. Wenn die Anweisung zum Laden des Akkumulators im indirekt indizierten Modus ausgeführt und die angegebene Zero-Page-Adresse $01 ist, dann lautet die tatsächliche Adresse:

```
Niederwertiges Byte=  Inhalt von $01
Höherwertiges Byte =  Inhalt von $02
Y-Register         =  $00
```

Die tatsächliche Adresse ist also $1E45 + Y = $1E45.
Dieser Modus enthält in der Tat ein indirektes Prinzip, auch wenn dies zunächst nur schwer zu verstehen ist.

INDIZIERT INDIREKT

Hierbei kann nur das X-Register als Index benutzt werden. Hierbei gilt das gleiche wie bei der indirekten Indizierung, außer daß hierbei die Zero-Page-Adresse des Zeigers indiziert wird und nicht die tatsächliche Basisadresse. Aus diesem Grund ist die tatsächliche Basisadresse auch wirklich diese Adresse, da der Index bereits für die indirekte Indizierung benutzt wurde. Die indizierte indirekte Adressierung würde oft benutzt, wenn sich eine Tabelle mit indirekten Zeigern auf der Zero-Page befindet und das X-Register dann den zu benutzenden indirekten Zeiger angeben würde.

BEISPIEL:

Nehmen wir an, Adresse $02 enthält $45 und Adresse $03 $10. Wird die Anweisung zum Laden des Akkumulators im indiziert indirekten Betrieb ausgeführt und ist die angegebene Zero-Page-Adresse $02, dann lautet die tatsächliche Adresse:

Niederwertiges Byte= Inhalt von ($02 + X)
Höherwertiges Byte = Inhalt von ($03 + X)
X-Register = $00

Der tatsächliche Zeiger zeigt daher auf = $02 + X = $02.
Die tatsächliche Adresse ist daher die indirekte Adresse in $02, also $1045.
Das Prinzip wird schon durch die Bezeichnung dieses Modus beschrieben, auch wenn dies zunächst nur schwer zu verstehen ist. Wir wollen das Problem von einer anderen Seite betrachten:

```
LDA #$00      - load low order actual base address
STA $06       - set the low byte of the indirect address
LDA #$16      - load high order indirect address
STA $07       - set the high byte of the indirect address
LDX #$05      - set the indirect index (X)
LDA ($01,X)   - load indirectly indexed by X
```

Anmerkung: Von diesen beiden indirekten Adressierarten wird am häufigsten die erste (indirekt indiziert) benutzt.

VERZWEIGUNGEN UND ÜBERPRÜFUNGEN

Ein weiteres wichtiges Prinzip der Maschinensprache ist die Möglichkeit, bestimmte Bedingungen zu überprüfen und zu erkennen. Dies entspricht der Struktur "IF . . . THEN, IF . . . GOTO" in CBM-BASIC.
Die verschiedenen "flags" im Statusregister werden durch die Anweisungen unterschiedlich beeinflußt. Ein Flag wird z. B. gesetzt, wenn eine Anweisung als Ergebnis eine Null hat und wird gelöscht, wenn das Ergebnis ungleich null ist. Die Anweisung

```
LDA #$00
```

führt zum Setzen eines *"Nullergebnis"-Flags,* da durch die Anweisung eine Null in den Akkumulator geladen wird.
Es gibt eine Gruppe von Anweisungen, durch die es bei bestimmten Bedingungen zu einer Verzweigung zu einem anderen Programmteil kommt. Eine Verzweigungsanweisung ist z. B. **BEQ** *(Verzweigung, wenn das Ergebnis gleich 0).* Eine Verzweigung erfolgt, *wenn die Bedingung erfüllt ist.* Ist die Bedingung nicht erfüllt, wird das Programm mit der nächsten Anweisung fortgesetzt. Es erfolgt keine Verzweigung durch das Ergebnis der vorherigen Anweisung(en), sondern eine interne Überprüfung des Statusregisters. Wie bereits erwähnt, befindet sich im Statusregister das *"Nullergebnis"-Flag.* Durch die BEQ-Anweisung erfolgt eine Verzweigung, wenn dieses Flag (Z) gesetzt ist. Für jede Verzweigungsanweisung gibt es ein entsprechendes Gegenstück. Das Gegenstück für BEQ lautet z. B. **BNE** (*Verzweigung,* wenn das *Ergebnis ungleich 0,* d. h. wenn Z nicht gesetzt ist).
Die Indexregister haben eine Anzahl zugehöriger Anweisungen, durch die ihre Inhalte geändert werden. Durch die Anweisung **INX** wird z. B. das *X-Indexregister inkrementiert.* Enthält das X-Indexregister vor der Inkrementierung $FF (max. Anzahl für das X-Register), dann erfolgt ein Sprung zurück zu 0. Soll ein Programm solange fortgesetzt werden, bis die Inkrementierung des X-Index erfolgte, so kann für diese "Schleife" also die BNE-Anweisung benutzt werden.
Das Gegenstück zu INX ist **DEX**, also das *Dekrement des X-Indexregisters.* Ist das X-Indexregister 0, dann erfolgt durch DEX ein Sprung zu $FF. Entsprechend gelten **INY** und **DEY** für das *Y-Indexregister.*
Wenn das Programm nun aber nicht warten soll, bis X oder Y=0 sind (bzw. nicht 0 sind)? Hierfür gibt es die *Vergleichsanweisungen* **CPX** und **CPY**, mit denen der Maschinensprache-Programmierer die Indexregister mit bestimmten Werten und sogar den Inhalten von Speicherplätzen überprüfen kann. Wollen Sie z. B. sehen, ob das X-Register $40 enthält, benutzen Sie folgende Anweisung:

CPX #$40	– Vergleiche X mit "WERT" $40.
BEQ (andere Programm-stelle)	– Verzweigung zu einer anderen Programmstelle, wenn diese Bedingung erfüllt ist.

Die Vergleichs- und Verzweigungsanweisungen spielen eine wichtige Rolle bei jedem Maschinensprache-Programm.
Beim MONITOR 64 ist der in einer Verzweigungsanweisung angegebene Operand die Adresse des Programmteils, zu dem gegebenenfalls eine Verzweigung erfolgt. Der Operand gibt jedoch lediglich die Sprungweite an, durch die man von der derzeitigen Programmposition zur angegebenen Adresse gelangt. Die Sprungweite kann maximal 1 Byte umfassen, so daß der mögliche Bereich für eine Verzweigungsanweisung beschränkt ist. Eine Verzweigung kann 127 Bytes vorwärts bzw. 128 rückwärts erfolgen.

Anmerkung: Hieraus ergibt sich ein Gesamtbereich von 255 Bytes, der natürlich dem max. Wertebereich eines Bytes entspricht.

MONITOR 64 zeigt Ihnen, wenn Sie bei der Verzweigung den Bereich überschreiten. Er kann diese Anweisung nämlich nicht assemblieren. Darüber brauchen Sie sich jedoch jetzt keine Sorgen zu machen, denn mit solchen Verzweigungen werden Sie sich vorläufig noch nicht beschäftigen. Die Verzweigung ist eine "schnelle" Anweisung der Maschinensprache, da die Verschiebung gegen eine Absolutadresse erfolgt. MONITOR 64 ermöglicht die Eingabe einer Absolutadresse und berechnet dann die korrekte Verschiebung. Dies ist nur einer von vielen Vorteilen des Arbeitens mit einem Assembler.

Anmerkung: Es kann nicht jede einzelne Verzweigungsanweisung behandelt werden. Bezüglich weiterer Einzelheiten siehe Literaturverzeichnis in Anhang F.

UNTERPROGRAMME

In Maschinensprache können Sie (genau wie beim Arbeiten mit BASIC) Unterprogramme aufrufen. Die entsprechende Anweisung lautet **JSR** (Sprung zum Unterprogramm), gefolgt von der angegebenen Absolutadresse.
Das Betriebssystem enthält ein Maschinensprache-Unterprogramm, durch das ein Zeichen auf dem Bildschirm angezeigt wird. Der CBM-ASCII-Code des Zeichens muß vor dem Aufruf im Akkumulator enthalten sein. Die Adresse dieses Unterprogramms lautet $FFD2.

Um "HI" auf dem Bildschirm anzuzeigen, ist folgendes Programm nötig:

```
.A 1400 LDA #$48      - Laden des CBM-ASCII-Codes von "H".
.A 1402 JSR $FFD2     - Anzeigen.
.A 1405 LDA #$49      - Laden des CBM-ASCII-Codes von "I".
.A 1407 JSR $FFD2     - Dieses auch anzeigen.
.A 140A LDA #$0D      - Eine Zeilenschaltung anzeigen.
.A 140C JSR $FFD2
.A 140F BRK           - Rückkehr zu MONITOR 64.
.G 1400               - Anzeige von "HI" und Rückkehr zu
                        MONITOR 64.
```

Dieses Programm zum Anzeigen eines Zeichens ist Teil der KERNAL-Sprungtabelle. **JMP** entspricht der BASIC-Anweisung GOTO. Hierbei erfolgt ein Sprung zur angegebenen Absolutadresse. Der KERNAL besteht aus einer langen Liste von Standard-Unterprogrammen, über die sämtliche Ein- und Ausgaben beim COMMODORE 64 gesteuert werden. Jede Eingabe in KERNAL springt zu einem Unterprogramm im Betriebssystem. Diese "Sprungtabelle" liegt zwischen den Speicherplätzen $FF84 und $FFF5 im Betriebssystem. Eine genaue Erklärung finden Sie im Abschnitt "KERNAL" dieses Handbuchs. Um zu zeigen, wie einfach und leistungsstark der KERNAL ist, wollen wir hier jedoch einige Programmbeispiele behandeln. Die soeben gelernten Methoden wollen wir nun in einem anderen Programm benutzen. Dies erleichtert Ihnen, die Anweisungen im Zusammenhang zu sehen: Über dieses Programm wird das Alphabet mit Hilfe einer KERNAL-Routine angezeigt. Die einzige neue Anweisung lautet **TXA** *(Übertragung vom X-Register zum Akkumulator).*

```
.A 1400 LDX #$41      - X = CBM-ASCII von "A".
.A 1402 TXA           - A = X.
.A 1403 JSR $FFD2     - Zeichen anzeigen.
.A 1406 INX           - Zählung des nichtadressierbaren
                        Hilfsspeichers.
.A 1407 CPX #$5B      - Haben wir "Z" überschritten?
.A 1409 BNE $1402     - Nein, zurückgehen und fortsetzen.
.A 140B BRK           - Ja, Rückkehr zu MONITOR 64.
```

Damit der COMMODORE 64 das Alphabet anzeigt, geben Sie folgenden Befehl ein:

```
.G 1400
```

Die Kommentare neben dem Programm erklären Programmablauf und Logik. Ein Programm sollten Sie zunächst auf Papier schreiben und danach in möglichst kleinen Teilen ausprobieren.

HINWEISE FÜR DEN ANFÄNGER

Maschinensprache erlernt man am besten, indem man sich Maschinensprache-Programme von anderen anschaut. Solche Programme werden ständig in Zeitungen und Zeitschriften veröffentlicht. Beschäftigen Sie sich mit dem Programm, auch wenn dieses sich auf einen anderen Computer bezieht, der mit dem Mikroprozessor 6510 (oder 6502) arbeitet. Vergewissern Sie sich, ob Sie den Code verstehen. Dies erfordert Ausdauer, besonders wenn es sich um eine Ihnen noch nicht bekannte Technik handelt. Dies kann sich als äußerst mühsam erweisen, bei ausreichender Geduld gehen Sie doch als Sieger hervor.
Nachdem Sie andere Maschinensprache-Programme angesehen haben, MÜSSEN Sie unbedingt eigene schreiben. Hierbei kann es sich um Dienstprogramme für Ihr BASIC-Programm oder um ein reines Maschinensprache-Programm handeln.
Sie sollten auch die entweder in dem Computer oder in einem Programm verfügbaren Hilfsmittel benutzen, die Ihnen beim Schreiben, Aufbereiten sowie Überprüfen von Maschinensprache-Programmen helfen. Als Beispiel dient hier der KERNAL, der Ihnen die Tastenabfrage, Textanzeige, Steuerung von Peripherie-Geräten wie z. B. Diskettenstation, Drucker, Modem usw., Speicherverwaltung und Bildschirmsteuerung ermöglicht. Der KERNAL ist äußerst leistungsstark, und seine Benutzung kann daher mit Nachdruck empfohlen werden (siehe "KERNAL" Seite 264).

Vorteile der Maschinensprache beim Programm-Schreiben:

1. Geschwindigkeit – Maschinensprache ist hundert- und manchmal auch tausendmal schneller als z. B. BASIC.
2. Sicherheit – Ein Maschinensprache-Programm ist sozusagen "idiotensicher", d. h. der Benutzer kann nur das ausführen, was das Programm erlaubt. Bei BASIC kann der Benutzer den BASIC-Interpreter z. B. dadurch "aussteigen lassen", daß er eine 0 eingibt. Das kann u. U. zu folgender Anzeige führen:

```
?DIVISION BY ZERO ERROR IN LINE 830

READY.
■
```

Der Computer kann also nur dann voll genutzt werden, wenn man Maschinensprache-Programme benutzt.

VORBEREITUNGEN FÜR EINE GROSSE AUFGABE

Wenn man eine große Aufgabe in Maschinensprache vorbereitet, so wurden meist schon eine Menge Dinge unbewußt durchdacht. Sie überlegen, wie bestimmte Vorgänge in Maschinensprache ausgeführt werden. Ganz zu Beginn sollten Sie das Programm zunächst auf ein Blatt Papier schreiben. Benutzen Sie Blockschaltbilder des Speichers, Funktionsmodule des erforderlichen Codes sowie einen Programmablauf.
Nehmen wir an, Sie wollen ein Roulette-Spiel in Maschinensprache schreiben. Dies könnte wie folgt entworfen werden:

- Titel anzeigen.
- Fragen, ob der Spieler Anleitungen braucht.
- JA – anzeigen – geh zum Start.
- NEIN – geh zum Start.
- Beginn der Initialisierung.
- HAUPTanzeige, Roulette-Tisch.
- Wetteinsätze annehmen.
- Rad drehen.
- Rad verlangsamen und anhalten.
- Wetteinsätze überprüfen und Ergebnis feststellen.
- Spieler informieren.
- Hat der Spieler noch Geld?
- JA – Kehr zur Hauptanzeige zurück.
- NEIN – Spieler informieren und Rückkehr zum Start.

Dies ist der Hauptentwurf. Diese einzelnen Bausteine können dann noch weiter unterteilt werden. Ein großes Problem wird also in immer kleinere Teile unterteilt. Auf diese Weise können Sie sich auch an zunächst unlösbar erscheinende Probleme heranwagen.
Hier hilft jedoch nur eines: Üben, üben, üben.

ANWEISUNGSSATZ VOM MIKROPROZESSOR

ADC	Mit Übertrag addieren
AND	Logisches UND
ASL	Verschiebung um ein Bit nach links
BCC	Verzweigen bei gelöschtem Übertrag
BCS	Verzweigen bei gesetztem Übertrag
BEQ	Verzweigen falls Ergebnis Null
BIT	Speicherbits testen
BMI	Verzweigen falls Ergebnis negativ
BNE	Verzweigen falls Ergebnis ungleich Null
BPL	Verzweigen falls Ergebnis positiv
BRK	Unterbrechung
BVC	Verzweigen falls kein Überlauf
BVS	Verzweigen bei Überlauf
CLC	Löschen des Übertrag-Flags
CLD	Löschen des Dezimal-Modus
CLI	Löschen des Interrupt-Disable-Bits
CLV	Löschen des Überlauf-Flags
CMP	Vergleich von Speicher und Akkumulator
CPX	Vergleich von Speicher und Register X
CPY	Vergleich von Speicher und Register Y
DEC	Speicherdekrementierung um 1
DEX	Dekrementierung von Register X um 1
DEY	Dekrementierung von Register Y um 1
EOR	"Exklusiv-oder"-Vergleich von Speicher und Akkumulator
INC	Speicherinkrementierung um 1
INX	Inkrementierung von Register X um 1
INY	Inkrementierung von Register Y um 1
JMP	Sprung zu neuem Speicherplatz
JSR	Sprung zu Unterprogramm

MCS6510 – ALPHABETISCHE REIHENFOLGE

LDA	Speicherübertragung zum Akkumulator
LDX	Speicherübertragung zu Register X
LDY	Speicherübertragung zu Register Y
LSR	Verschiebung um 1 Bit nach rechts
NOP	Keine Operation
ORA	ODER-Verknüpfung von Speicher und Akkumulator
PHA	Speicherung des Akkumulators im Stapelregister
PHP	Speicherung des Prozessorstatus im Stapel
PLA	Akkumulator vom Stapel holen
PLP	Prozessorstatus vom Stapel holen
ROL	Rotiere um 1 Bit nach links (Speicher oder Akkumulator)
ROR	Rotiere um 1 Bit nach rechts (Speicher oder Akkumulator)
RTI	Rückkehr von Programmunterbrechung
RTS	Rückkehr vom Unterprogramm
SBC	Speicher mit Übertrag vom Akkumulator subtrahieren
SEC	Übertragungsflag setzen
SED	Dezimalmodus einschalten
SEI	Unterbrechungsmaske setzen
STA	Akkumulator in Speicher ablegen
STX	Register X in Speicher ablegen
STY	Register Y in Speicher ablegen
TAX	Akkumulator abspeichern in Register X
TAY	Akkumulator abspeichern in Register Y
TSX	Stapelzeiger S in Register X übertragen
TXA	Übertragung von Register X zum Akkumulator
TXS	Übertragung von Register X zum Stapelzeiger
TYA	Übertragung von Register Y zum Akkumulator

Nachfolgende Angaben beziehen sich auf die folgende Zusammenfassung:

A	Akkumulator
X, Y	Indexregister
M	Speicher
P	Prozessorstatus-Register
S	Stapelzeiger
$\surd$	Wechsel
–	Kein Wechsel
+	Addieren
$\wedge$	Logisches UND
–	Subtrahieren
$\forall$	Logisches ausschließendes ODER
$\uparrow$	Übertragung vom Stapel
$\downarrow$	Übertragung zum Stapel
$\rightarrow$	Übertragung zu
$\leftarrow$	Übertragung von
V	Logisches ODER
PC	Programmzähler
PCH	Programmzähler, höherwertiges Byte
PCL	Programmzähler, niederwertiges Byte
OPER	OPERAND
#	Unmittelbare Adressierung

Anmerkung: Am Anfang jeder Tabelle steht in Klammern eine Referenznummer (Ref: XX), die das jeweilige Kapitel im "Programmierhandbuch zum MOS 650 2" angibt.

ADC

Mit Übertrag addieren

ADC

Ablauf: A + M + C → A, C

N	Z	C	I	D	V
√	√	√	–	–	√

(Ref: 2.2.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
unmittelbar	ADC # Oper	69	2	2
Zero-Page	ADC Oper	65	2	3
Zero-Page, X	ADC Oper, X	75	2	4
Absolut	ADC Oper	6D	3	4
Absolut, X	ADC Oper, X	7D	3	4*
Absolut, Y	ADC Oper, Y	79	3	4*
(Indirekt, X)	ADC (Oper, X)	61	2	6
(Indirekt), Y	ADC (Oper), Y	71	2	5*

* 1 addieren, wenn Seitengrenze überschritten wird.

AND

Logisches UND

AND

Logisches UND zum Akkumulator

Ablauf: A ∧ M → A

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 2.2.3.0)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
unmittelbar	AND # Oper	29	2	2
Zero-Page	AND Oper	25	2	3
Zero-Page, X	AND Oper, X	35	2	4
Absolut	AND Oper	2D	3	4
Absolut, X	AND Oper, X	3D	3	4*
Absolut, Y	AND Oper, Y	39	3	4*
(Indirekt, X)	AND (Oper, X)	21	2	6
(Indirekt), Y	AND (Oper), Y	31	2	5

* 1 addieren, wenn Seitengrenze überschritten wird.

ASL

Verschiebung nach links um 1 Bit

ASL

Ablauf: C ← | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø | ← Ø

N	Z	C	I	D	V
√	√	√	–	–	–

(Ref: 10.2)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Akkumulator	ASL	A	ØA	1	2
Zero-Page	ASL	Oper	Ø6	2	5
Zero-Page, X	ASL	Oper, X	16	2	6
Absolut	ASL	Oper	ØE	3	6
Absolut, X	ASL	Oper, X	1E	3	7

BCC

Verzweigung bei gelöschtem Übertrag

BCC

Ablauf: Verzweigung bei C = Ø

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.3)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BCC	Oper	9Ø	2	2*

* 1 addieren bei Verzweigung auf der gleichen Seite.

* 2 addieren bei Verzweigung auf unterschiedlichen Seiten.

BCS

Verzweigung bei gesetztem Übertrag

BCS

Ablauf: Verzweigung bei C = 1

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.4)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BCS	Oper	BØ	2	2*

* 1 addieren bei Verzweigung auf der gleichen Seite.

* 2 addieren, wenn Verzweigung auf der nächsten Seite auftritt.

BEQ

Verzweigung falls Ergebnis Null

BEQ

Ablauf: Verzweigung bei Z = 1

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.5)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BEQ Oper	FØ	2	2*

* 1 addieren, wenn Verzweigung auf der gleichen Seite.

* 2 addieren, bei Verzweigung zur nächsten Seite.

BIT

Speicherbits testen

BIT

Ablauf: A ∧ M, $M_7 \rightarrow$ N, $M_6 \rightarrow$ V

Bit 6 und 7 werden zum Statusregister übertragen. Ist das Ergebnis von A ∧ M null, dann ist Z = 1, andernfalls ist Z = Ø.

N	Z	C	I	D	V
M_7	√	–	–	–	M_6

(Ref: 4.2.1.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Zero-Page	BIT Oper	24	2	3
Absolut	BIT Oper	2C	3	4

BMI

Verzweigung bei Minusresultat

BMI

Ablauf: Verzweigung bei N = 1

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BMI Oper	3Ø	2	2*

* 1 addieren, bei Verzweigung auf der gleichen Seite.

* 2 addieren, bei Verzweigung auf verschiedenen Seiten.

BNE — *Verzweigung falls Ergebnis ungleich Null* — BNE

Ablauf: Verzweigung bei Z = 0

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.6)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BNE Oper	DØ	2	2*

* 1 addieren, bei Verzweigung auf der gleichen Seite.
* 2 addieren, bei Verzweigung auf verschiedenen Seiten.

BPL — *Verzweigung bei Plusresultat* — BPL

Ablauf: Verzweigung bei N = Ø

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BPL Oper	1Ø	2	2*

* 1 addieren, bei Verzweigung auf der gleichen Seite.
* 2 addieren, bei Verzweigung auf verschiedenen Seiten.

BRK — *Unterbrechung* — BRK

Ablauf: Abbruch PC + 2 ↓ P ↓

N	Z	C	I	D	V
–	–	–	1	–	–

(Ref: 9.11)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	BRK	ØØ	1	7

1. Ein BRK-Befehl kann nicht durch Setzen von I maskiert werden.

BVC — *Verzweigung falls kein Überlauf* — BVC

Ablauf: Verzweigung bei V = 0

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.8)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BVC Oper	5Ø	2	2*

* 1 addieren, bei Verzweigung auf der gleichen Seite.
* 2 addieren, bei Verzweigung auf unterschiedlichen Seiten.

BVS — *Verzweigung bei Überlauf* — BVS

Ablauf: Verzweigung bei V = 1

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.1.1.7)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Relativ	BVS Oper	7Ø	2	2*

* 1 addieren, bei Verzweigung auf der gleichen Seite.
* 2 addieren, bei Verzweigung auf unterschiedlichen Seiten.

CLC — *Löschen des Übertrag-Flags* — CLC

Ablauf: Ø → C

N	Z	C	I	D	V
–	–	Ø	–	–	–

(Ref: 3.0.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	CLC	18	1	2

CLD

Löschen des Dezimal-Modus

CLD

Ablauf: Ø → D

N	Z	C	I	D	V
–	–	–	–	Ø	–

(Ref: 3.3.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	CLD	D8	1	2

CLI

Löschen des Interrupt-Disable-Bits

CLI

Ablauf: Ø → I

N	Z	C	I	D	V
–	–	–	Ø	–	–

(Ref: 3.2.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	CLI	58	1	2

CLV

Löschen des Überlauf-Flags

CLV

Ablauf: Ø → V

N	Z	C	I	D	V
–	–	–	–	–	Ø

(Ref: 3.6.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	CLV	B8	1	2

CMP

Vergleichen von Speicher und Akkumulator

CMP

Ablauf: A – M

N	Z	C	I	D	V
√	√	√	–	–	–

(Ref: 4.2.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	CMP # Oper	C9	2	2
Zero-Page	CMP Oper	C5	2	3
Zero-Page, X	CMP Oper, X	D5	2	4
Absolut	CMP Oper	CD	3	4
Absolut, X	CMP Oper, X	DD	3	4*
Absolut, Y	CMP Oper, Y	D9	3	4*
(Indirekt, X)	CMP (Oper, X)	C1	2	6
(Indirekt), Y	CMP (Oper), Y	D1	2	5*

* 1 addieren, wenn Seite überschritten wird.

CPX

Vergleich von Speicher und Register X

CPX

Ablauf: X – M

N	Z	C	I	D	V
√	√	√	–	–	–

(Ref: 7.8)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	CPX #Oper	EØ	2	2
Zero-Page	CPX Oper	E4	2	3
Absolut	CPX Oper	EC	3	4

CPY

Vergleich von Speicher und Register Y

CPY

Ablauf: Y – M

N	Z	C	I	D	V
√	√	√	–	–	–

(Ref: 7.9)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	CPY # Oper	CØ	2	2
Zero-Page	CPY Oper	C4	2	3
Absolut	CPY Oper	CC	3	4

DEC

Speicherdekrementierung um 1

DEC

Ablauf: M – 1 → M

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 10.7)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Zero-Page	DEC	Oper	C6	2	5
Zero-Page, X	DEC	Oper, X	D6	2	6
Absolut	DEC	Oper	CE	3	3
Absolut, X	DEC	Oper, X	DE	3	7

DEX

Dekrementierung von Register X um 1

DEX

Ablauf: X – 1 → X

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.6)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	DEX	CA	1	2

DEY

Dekrementierung von Register Y um 1

DEY

Ablauf: Y – 1 → Y

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.7)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	DEY	88	1	2

EOR *"Exklusiv-oder"-Vergleich von Speicher und Akkumulator* EOR

Ablauf: A ∀ M → A

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 2.2.3.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	EOR # Oper	49	2	2
Zero-Page	EOR Oper	45	2	3
Zero-Page, X	EOR Oper, X	55	2	4
Absolut	EOR Oper	4D	3	4
Absolut, X	EOR Oper, X	5D	3	4*
Absolut, Y	EOR Oper, Y	59	3	4*
(Indirekt, X)	EOR (Oper, X)	41	2	6
(Indirekt), Y	EOR (Oper), Y	51	2	5*

* 1 addieren, wenn Seite überschritten wird.

INC *Speicherinkrementierung um 1* INC

Ablauf: M + 1 → M

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 10.6)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Zero-Page	INC Oper	E6	2	5
Zero-Page, X	INC Oper, X	F6	2	6
Absolut	INC Oper	EE	3	6
Absolut, X	INC Oper, X	FE	3	7

INX *Inkrementierung von Register X um 1* INX

Ablauf: X + 1 → X

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.4)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	INX	E8	1	2

INY

Inkrementierung von Register Y um 1

INY

Ablauf: Y + 1 → Y

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.5)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	INY	C8	1	2

JMP

Sprung zu neuem Speicherplatz

JMP

Ablauf: (PC + 1) → PCL
(PC + 2) → PCH

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 4.0.2; Ref: 9.8.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Absolut	JMP Oper	4C	3	3
Indirekt	JMP (Oper)	6C	3	5

JSR

Sprung zu neuer Speicherrückkehradresse

JSR

Ablauf: PC + 2 ↓, (PC + 1) → PCL
(PC + 2) → PCH

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 8.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Absolut	JSR Oper	20	3	6

LDA

Speicherübertragung zum Akkumulator

LDA

Ablauf: M → A

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 2.1.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	LDA # Oper	A9	2	2
Zero-Page	LDA Oper	A5	2	3
Zero-Page, X	LDA Oper, X	B5	2	4
Absolut	LDA Oper	AD	3	4
Absolut, X	LDA Oper, X	BD	3	4*
Absolut, Y	LDA Oper, Y	B9	3	4*
(Indirekt, X)	LDA (Oper, X)	A1	2	6
(Indirekt), Y	LDA (Oper), Y	B1	2	5*

* 1 addieren, wenn Seite überschritten wird.

LDX

Speicherübertragung zu Register X

LDX

Ablauf: M → X

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.0)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	LDX # Oper	A2	2	2
Zero-Page	LDX Oper	A6	2	3
Zero-Page, Y	LDX Oper, Y	B6	2	4
Absolut	LDX Oper	AE	3	4
Absolut, Y	LDX Oper, Y	BE	3	4*

* 1 addieren, wenn Seite überschritten wird.

LDY

Speicherübertragung zu Register Y

LDY

Ablauf: M → Y

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	LDY # Oper	AØ	2	2
Zero-Page	LDY Oper	A4	2	3
Zero-Page, X	LDY Oper, X	B4	2	4
Absolut	LDY Oper	AC	3	4
Absolut, X	LDY Oper, X	BC	3	4*

* 1 addieren, wenn Seite überschritten wird.

LSR

Verschiebung um 1 Bit nach rechts (Speicher oder Akkumulator)

LSR

Ablauf: Ø → | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | → C

N	Z	C	I	D	V
Ø	√	√	–	–	–

(Ref: 10.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Akkumulator	LSR A	4A	1	2
Zero-Page	LSR Oper	46	2	5
Zero-Page, X	LSR Oper, X	56	2	6
Absolut	LSR Oper	4E	3	6
Absolut, X	LSR Oper, X	5E	3	7

NOP

Keine Operation

NOP

Ablauf: Keine Operation (2 Zyklen)

N	Z	C	I	D	V
–	–	–	–	–	–

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	NOP	EA	1	2

ORA — *ODER-Verknüpfung von Speicher und Akkumulator* — ORA

Ablauf: A ∨ M → A

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 2.2.3.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	ORA # Oper	Ø9	2	2
Zero-Page	ORA Oper	Ø5	2	3
Zero-Page, X	ORA Oper, X	15	2	4
Absolut	ORA Oper	ØD	3	4
Absolut, X	ORA Oper, X	1D	3	4*
Absolut, Y	ORA Oper, Y	19	3	4*
(Indirekt, X)	ORA (Oper, X)	Ø1	2	6
(Indirekt), Y	ORA (Oper), Y	11	2	5

* 1 addieren, wenn Seite überschritten wird.

PHA — *Speicherung des Akkumulators im Stapelregister* — PHA

Ablauf: A ↓

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 8.5)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	PHA	48	1	3

PHP — *Speicherung des Prozessorstatus im Stapel* — PHP

Ablauf: P ↓

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 8.11)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	PHP	Ø8	1	3

PLA

Akkumulator vom Stapel holen

PLA

Ablauf: A ↑

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 8.6)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	PLA	68	1	4

PLP

Prozessorstatus vom Stapel holen

PLP

Ablauf: P ↑

N	Z	C	I	D	V
Vom Stapel					

(Ref: 8.12)

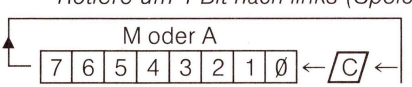
Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	PLP	28	1	4

ROL

Rotiere um 1 Bit nach links (Speicher oder Akkumulator)

ROL

Ablauf:

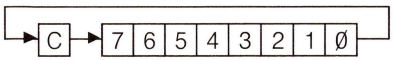


N	Z	C	I	D	V
√	√	√	–	–	–

(Ref: 10.3)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Akkumulator	ROL	A	2A	1	2
Zero-Page	ROL	Oper	26	2	5
Zero-Page, X	ROL	Oper, X	36	2	6
Absolut	ROL	Oper	2E	3	6
Absolut, X	ROL	Oper, X	3E	3	7

ROR — *Rotiere um 1 Bit nach rechts (Speicher oder Akkumulator)* — ROR

Ablauf: 

N	Z	C	I	D	V
√	√	√	–	–	–

(Ref: 10.4)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Akkumulator	ROR	A	6A	1	2
Zero-Page	ROR	Oper	66	2	5
Zero-Page, X	ROR	Oper, X	76	2	6
Absolut	ROR	Oper	6E	3	6
Absolut, X	ROR	Oper, X	7E	3	7

RTI — *Rückkehr von Programmunterbrechung* — RTI

Ablauf: P ↑ PC ↑

N	Z	C	I	D	V
Vom Stapel					

(Ref: 9.6)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	RTI	4Ø	1	6

RTS — *Rückkehr vom Unterprogramm* — RTS

Ablauf: PC ↑, PC + 1 → PC

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 8.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	RTS	6Ø	1	6

SBC *Speicherung und Übertragung vom Akkumulator subtrahieren* SBC

Ablauf: $A - M - \overline{C} \rightarrow A$

Anmerkung: $\overline{C}$ = Übertrag

N	Z	C	I	D	V
√	√	√	–	–	√

(Ref: 2.2.2)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Unmittelbar	SBC # Oper	E9	2	2
Zero-Page	SBC Oper	E5	2	3
Zero-Page, X	SBC Oper, X	F5	2	4
Absolut	SBC Oper	ED	3	4
Absolut, X	SBC Oper, X	FD	3	4*
Absolut, Y	SBC Oper, Y	F9	3	4*
(Indirekt, X)	SBC (Oper, X)	E1	2	6
(Indirekt), Y	SBC (Oper), Y	F1	2	5*

* 1 addieren, wenn Seite überschritten wird.

SEC *Übertragungsflag setzen* SEC

Ablauf: $1 \rightarrow C$

N	Z	C	I	D	V
–	–	1	–	–	–

(Ref: 3.0.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	SEC	38	1	2

SED *Dezimalmodus einschalten* SED

Ablauf: $1 \rightarrow D$

N	Z	C	I	D	V
–	–	–	–	1	–

(Ref: 3.3.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	SED	F8	1	2

SEI

Unterbrechungsmaske setzen

SEI

Ablauf: 1 → I

N	Z	C	I	D	V
–	–	–	1	–	–

(Ref: 3.2.1)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	SEI	78	1	2

STA

Akkumulator in Speicher ablegen

STA

Ablauf: A → M

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 2.1.2)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Zero-Page	STA	Oper	85	2	3
Zero-Page, X	STA	Oper, X	95	2	4
Absolut	STA	Oper	8D	3	4
Absolut, X	STA	Oper, X	9D	3	5
Absolut, Y	STA	Oper, Y	99	3	5
(Indirekt, X)	STA	(Oper, X)	81	2	6
(Indirekt), Y	STA	(Oper), Y	91	2	6

STX

Register X in Speicher ablegen

STX

Ablauf: X → M

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 7.2)

Adressierart	Assembler-Sprachenformat		OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Zero-Page	STX	Oper	86	2	3
Zero-Page, Y	STX	Oper, Y	96	2	4
Absolut	STX	Oper	8E	3	4

STY

Register Y in Speicher ablegen

STY

Ablauf: Y → M

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 7.3)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Zero-Page	STY Oper	84	2	3
Zero-Page, X	STY Oper, X	94	2	4
Absolut	STY Oper	8C	3	4

TAX

Akkumulator abspeichern in Register X

TAX

Ablauf: A → X

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.11)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	TAX	AA	1	2

TAY

Akkumulator abspeichern in Register Y

TAY

Ablauf: A → Y

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.13)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	TAY	A8	1	2

TSX

Stapelzeiger in Register X übertragen

TSX

Ablauf: S → X

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 8.9)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	TSX	BA	1	2

TXA

Übertragung von Register X zum Akkumulator

TXA

Ablauf: X → A

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.12)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	TXA	8A	1	2

TXS

Übertragung von Register X zum Stapelzeiger

TXS

Ablauf: X → S

N	Z	C	I	D	V
–	–	–	–	–	–

(Ref: 8.8)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	TXS	9A	1	2

TYA

Übertragung von Register Y zum Akkumulator

TYA

Ablauf: Y → A

N	Z	C	I	D	V
√	√	–	–	–	–

(Ref: 7.14)

Adressierart	Assembler-Sprachenformat	OP-Code	Anzahl der Bytes	Anzahl der Zyklen
Impliziert	TYA	98	1	2

ANWEISUNGS-ADRESSIERARTEN UND ZUGEHÖRIGE

	Akkumulator	Unmittelbar	Zero-Page	Zero-Page, X	Zero-Page, Y	Absolut	Absolut, X	Absolut, Y	Impliziert	Relativ	(Indirekt, X)	(Indirekt), Y	Absolut Indirekt
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	.	6	7	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	.	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

* Einen Zyklus addieren, wenn die Indizierung eine Seite überschreitet.

** Einen Zyklus bei Verzweigung addieren, einen weiteren addieren, wenn

AUSFÜHRUNGSZEITEN (IN TAKTZYKLEN)

	Akkumulator	Unmittelbar	Zero-Page	Zero-Page, X	Zero-Page, Y	Absolut	Absolut, X	Absolut, Y	Impliziert	Relativ	(Indirekt, X)	(Indirekt), Y	Absolut Indirekt
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

die Verzweigung die Seitengrenze überschreitet.

ØØ – BRK
Ø1 – ORA – (Indirekt, X)
Ø2 – Künftige Erweiterung
Ø3 – Künftige Erweiterung
Ø4 – Künftige Erweiterung
Ø5 – ORA – Zero-Page
Ø6 – ASL – Zero-Page
Ø7 – Künftige Erweiterung
Ø8 – PHP
Ø9 – ORA – Unmittelbar
ØA – ASL – Akkumulator
ØB – Künftige Erweiterung
ØC – Künftige Erweiterung
ØD – ORA – Absolut
ØE – ASL – Absolut
ØF – Künftige Erweiterung
1Ø – BPL
11 – ORA – (Indirekt), Y
12 – Künftige Erweiterung
13 – Künftige Erweiterung
14 – Künftige Erweiterung
15 – ORA – Zero-Page, X
16 – ASL – Zero-Page, X
17 – Künftige Erweiterung
18 – CLC
19 – ORA – Absolut, Y
1A – Künftige Erweiterung
1B – Künftige Erweiterung
1C – Künftige Erweiterung
1D – ORA – Absolut, X
1E – ASL – Absolut, X
1F – Künftige Erweiterung

2Ø – JSR
21 – AND (Indirekt, X)
22 – Künftige Erweiterung
23 – Künftige Erweiterung
24 – BIT – Zero-Page
25 – AND – Zero-Page
26 – ROL – Zero-Page
27 – Künftige Erweiterung
28 – PLP
29 – AND – Unmittelbar
2A – ROL – Akkumulator
2B – Künftige Erweiterung
2C – BIT – Absolut
2D – AND – Absolut
2E – ROL – Absolut
2F – Künftige Erweiterung
3Ø – BMI
31 – AND – (Indirekt), Y
32 – Künftige Erweiterung
33 – Künftige Erweiterung
34 – Künftige Erweiterung
35 – AND – Zero-Page, X
36 – ROL – Zero-Page, X
37 – Künftige Erweiterung
38 – SEC
39 – AND – Absolut, Y
3A – Künftige Erweiterung
3B – Künftige Erweiterung
3C – Künftige Erweiterung
3D – AND – Absolut, X
3E – ROL – Absolut, X
3F – Künftige Erweiterung

4Ø – RTI
41 – EOR – (Indirekt, X)
42 – Künftige Erweiterung
43 – Künftige Erweiterung
44 – Künftige Erweiterung
45 – EOR – Zero-Page
46 – LSR – Zero-Page
47 – Künftige Erweiterung
48 – PHA
49 – EOR – Unmittelbar
4A – LSR – Akkumulator
4B – Künftige Erweiterung
4C – JMP – Unmittelbar
4D – EOR – Unmittelbar
4E – LSR – Unmittelbar
4F – Künftige Erweiterung
5Ø – BVC
51 – EOR – (Indirekt), Y
52 – Künftige Erweiterung
53 – Künftige Erweiterung
54 – Künftige Erweiterung
55 – EOR – Zero-Page, X
56 – LSR – Zero-Page, X
57 – Künftige Erweiterung
58 – CLI
59 – EOR – Absolut, Y
5A – Künftige Erweiterung
5B – Künftige Erweiterung
5C – Künftige Erweiterung
5D – EOR – Absolut, X
5E – LSR – Absolut, X
5F – Künftige Erweiterung

6Ø – RTS
61 – ADC – (Indirekt, X)
62 – Künftige Erweiterung
63 – Künftige Erweiterung
64 – Künftige Erweiterung
65 – ADC – Zero-Page
66 – ROR – Zero-Page
67 – Künftige Erweiterung
68 – PLA
69 – ADC – Unmittelbar
6A – ROR – Akkumulator
6B – Künftige Erweiterung
6C – JMP – Indirekt
6D – ADC – Absolut
6E – ROR – Absolut
6F – Künftige Erweiterung
7Ø – BVS
71 – ADC – (Indirekt), Y
72 – Künftige Erweiterung
73 – Künftige Erweiterung
74 – Künftige Erweiterung
75 – ADC – Zero-Page, X
76 – ROR – Zero-Page, X
77 – Künftige Erweiterung
78 – SEI
79 – ADC – Absolut, Y
7A – Künftige Erweiterung
7B – Künftige Erweiterung
7C – Künftige Erweiterung
7D – ADC – Absolut, X
7E – ROR – Absolut, X
7F – Künftige Erweiterung

8Ø – Künftige Erweiterung
81 – STA – (Indirekt, X)
82 – Künftige Erweiterung
83 – Künftige Erweiterung
84 – STY – Zero-Page
85 – STA – Zero-Page
86 – STX – Zero-Page
87 – Künftige Erweiterung
88 – DEY
89 – Künftige Erweiterung
8A – TXA
8B – Künftige Erweiterung
8C – STY – Absolut
8D – STA – Absolut
8E – STX – Absolut
8F – Künftige Erweiterung
9Ø – BCC
91 – STA – (Indirekt), Y
92 – Künftige Erweiterung
93 – Künftige Erweiterung
94 – STY – Zero-Page, X
95 – STA – Zero-Page, X
96 – STX – Zero-Page, Y
97 – Künftige Erweiterung
98 – TYA
99 – STA – Absolut, Y
9A – TXS
9B – Künftige Erweiterung
9C – Künftige Erweiterung
9D – STA – Absolut, X
9E – Künftige Erweiterung
9F – Künftige Erweiterung

AØ – LDY – Unmittelbar
A1 – LDA – (Indirekt, X)
A2 – LDX – Unmittelbar
A3 – Künftige Erweiterung
A4 – LDY – Zero-Page
A5 – LDA – Zero-Page
A6 – LDX – Zero-Page
A7 – Künftige Erweiterung
A8 – TAY
A9 – LDA – Unmittelbar
AA – TAX
AB – Künftige Erweiterung
AC – LDY – Unmittelbar
AD – LDA – Unmittelbar
AE – LDX – Unmittelbar
AF – Künftige Erweiterung
BØ – BCC
B1 – LDA – (Indirekt), Y
B2 – Künftige Erweiterung
B3 – Künftige Erweiterung
B4 – LDY – Zero-Page, X
B5 – LDA – Zero-Page, X
B6 – LDX – Zero-Page, Y
B7 – Künftige Erweiterung
B8 – CLV
B9 – LDA – Absolut, Y
BA – TSX
BB – Künftige Erweiterung
BC – LDY – Absolut, X
BD – LDA – Absolut, X
BE – LDX – Absolut, Y
BF – Künftige Erweiterung

CØ – CPY – Unmittelbar
C1 – CPM – (Indirekt, X)
C2 – Künftige Erweiterung
C3 – Künftige Erweiterung
C4 – CPY – Zero-Page
C5 – CMP – Zero-Page
C6 – DEC – Zero-Page
C7 – Künftige Erweiterung
C8 – INY
C9 – CMP – Unmittelbar
CA – DEX
CB – Künftige Erweiterung
CC – CPY – Absolut
CD – CMP – Absolut
CE – DEC – Absolut
CF – Künftige Erweiterung
DØ – BNE
D1 – CMP – (Indirekt), Y
D2 – Künftige Erweiterung
D3 – Künftige Erweiterung
D4 – Künftige Erweiterung
D5 – CMP – Zero-Page, X
D6 – DEC – Zero-Page, X
D7 – Künftige Erweiterung
D8 – CLD
D9 – CMP – Absolut, Y
DA – Künftige Erweiterung
DB – Künftige Erweiterung
DC – Künftige Erweiterung
DD – CMP – Absolut, X
DE – DEC – Absolut, X
DF – Künftige Erweiterung

EØ – CPX – Unmittelbar
E1 – SBC – (Indirekt, X)
E2 – Künftige Erweiterung
E3 – Künftige Erweiterung
E4 – CPX – Zero-Page
E5 – SBC – Zero-Page
E6 – INC – Zero-Page
E7 – Künftige Erweiterung
E8 – INX
E9 – SBC – Unmittelbar
EA – NOP
EB – Künftige Erweiterung
EC – CPX – Absolut
ED – SBC – Absolut
EE – INC – Absolut
EF – Künftige Erweiterung
FØ – BEQ
F1 – SBC – (Indirekt), Y
F2 – Künftige Erweiterung
F3 – Künftige Erweiterung
F4 – Künftige Erweiterung
F5 – SBC – Zero-Page, X
F6 – INC – Zero-Page, X
F7 – Künftige Erweiterung
F8 – SED
F9 – SBC – Absolut, Y
FA – Künftige Erweiterung
FB – Künftige Erweiterung
FC – Künftige Erweiterung
FD – SBC – Absolut, X
FE – INC – Absolut, X
FF – Künftige Erweiterung

SPEICHERVERWALTUNG BEIM COMMODORE 64

Der COMMODORE 64 besitzt 64K-Byte RAM. Darüber hinaus hat er 20K-Byte ROM, in dem BASIC, Betriebssystem und Standardzeichensatz gespeichert sind. Außerdem hat er Zugriff auf Ein-/Ausgabevorrichtungen und benutzt dazu 4K-Speicherkapazität. Wie ist dies alles mit einem Computer mit 16-Bit-Adreß-Bus möglich, der normalerweise nur 64K adressieren kann?

Das Geheimnis liegt im 6510-Prozessor-Chip selbst. Der Chip hat ein Ein-/Ausgabeport. Über diesen Port wird gesteuert, ob RAM, ROM oder Ein-/Ausgabe in bestimmten Speicherabschnitten erscheint. Er dient auch der Steuerung der Datasette™, so daß nur die geeigneten Bits verändert werden dürfen.

Der Ein-/Ausgabeport des 6510 belegt Adresse 1. Das Datenrichtungsregister für diesen Port liegt im Speicherplatz 0. Der Port wird genau wie andere Ein-/Ausgabeports des Systems gesteuert. Das Datenrichtungsregister steuert, ob ein bestimmtes Bit eine Eingabe oder eine Ausgabe ist. Die tatsächliche Datenübertragung erfolgt über den Port selbst.

Die Positionen im Kontrollregister des 6510 sind wie folgt definiert:

NAME	BIT	RICHTUNG	BESCHREIBUNG
LORAM	0	AUSGABE	Steuerung: RAM/ROM von $A000–$BFFF (BASIC)
HIRAM	1	AUSGABE	Steuerung: RAM/ROM von $E000–$FFF (KERNAL)
CHAREN	2	AUSGABE	Steuerung: Ein-/Ausgabe/ ROM von $D000–$DFFF
	3	AUSGABE	Kassettenschreibleitung
	4	EINGABE	Kassettenschalter
	5	AUSGABE	Kassettenmotorsteuerung

Der richtige Wert für das Datenrichtungsregister lautet wie folgt:

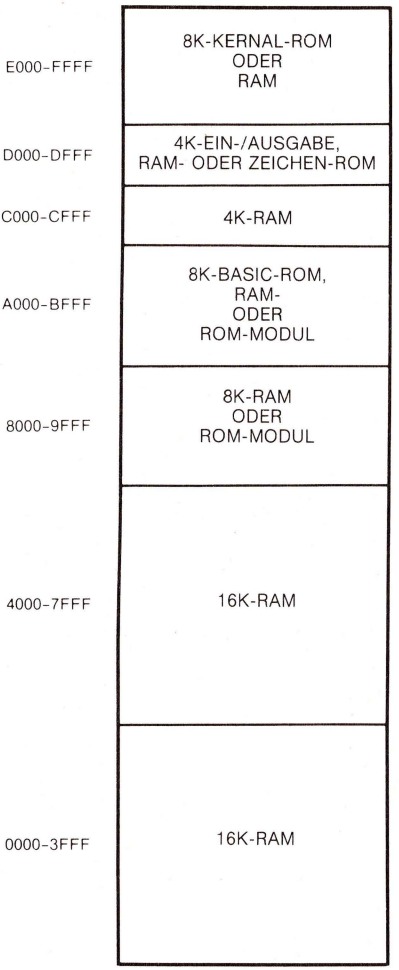
BITS	5	4	3	2	1	0
	1	0	1	1	1	1

(wobei 1 für Ausgabe und 0 für Eingabe steht).

Das entspricht dem Dezimalwert 47. Der COMMODORE 64 setzt automatisch das Datenrichtungsregister auf diesen Wert.
Die Steuerbits (Steuerleitungen) führen im allgemeinen die in ihrer Beschreibung angegebenen Funktionen aus. Für einen bestimmten Speicheraufbau werden die Steuerbits jedoch gelegentlich miteinander kombiniert.
LORAM (Bit 0) ist ein Steuerbit, mit dem der 8K-Byte-BASIC-ROM in den und aus dem Mikroprozessor-Adreß-Bereich geschaltet wird. Normalerweise gilt HIGH für dieses Bit bei BASIC-Betrieb. Ist dieses Bit LOW programmiert, verschwindet der BASIC-ROM aus dem Speicher und wird durch 8K-Bytes-RAM von $A000 bis $BFFF ersetzt.
HIRAM (Bit 1) ist eine Art Steuerleitung, mit der der 8K-Byte-KERNAL-ROM in den und aus dem Mikroprozessor-Adreß-Bereich geschaltet wird. Normalerweise ist diese Leitung im BASIC-Betrieb HIGH. Ist diese Leitung LOW programmiert, so verschwindet der KERNAL-ROM aus dem Speicher und wird durch 8K-Byte vom RAM von $E000 bis $FFFF ersetzt.
CHAREN (Bit 2) wird nur benutzt, um den 4K-Byte-Zeichengenerator-ROM in den oder aus dem Mikroprozessor-Adreß-Bereich zu schalten. Vom Prozessor aus gesehen, benutzt der Zeichen-ROM den gleichen Adreß-Bereich wie die Ein-/Ausgabe-Register ($D000–$DFFF). Wenn die CHAREN-Leitung auf 1 gesetzt ist (Normaleinstellung), dann erscheinen die Ein-/Ausgabe-Register im Mikroprozessor-Adreß-Bereich, und der Zeichen-ROM ist nicht zugänglich. Wird der CHAREN-Bit auf 0 gesetzt, erscheint der Zeichen-ROM. Nun sind die Ein-/Ausgabe-Register nicht zugänglich. (Der Mikroprozessor muß lediglich auf den Zeichen-ROM zugreifen, wenn der Zeichensatz vom ROM ins RAM gebracht wird. Dies erfordert besondere Umsicht . . . siehe Abschnitt "Programmierbare Zeichen" im Kapitel "Graphik").
CHAREN kann durch andere Steuerzeichen in bestimmten Speicheranordnungen unwirksam gemacht werden. CHAREN beeinflußt keine Speicheranordnungen ohne Ein-/Ausgabe-Register. Statt dessen erscheint von $D000 bis $DFFF der RAM.

Anmerkung: Bei jeder Speicherverteilung, die ROM enthält, werden die Daten beim Schreiben (POKE) in einer ROM-Adresse in den RAM "unter" dem ROM gespeichert. Die Daten werden also in den "versteckten" RAM geschrieben. Auf diese Weise ist ein Bildschirm mit hoher Auflösung unter dem ROM möglich und kann (ohne vorherige Übertragung in den Prozessor-Adreßraum) geändert werden. Beim Lesen einer ROM-Adresse wird natürlich aus dem ROM und nicht aus dem "versteckten" RAM gelesen.

SPEICHERMAPPE VOM COMMODORE 64



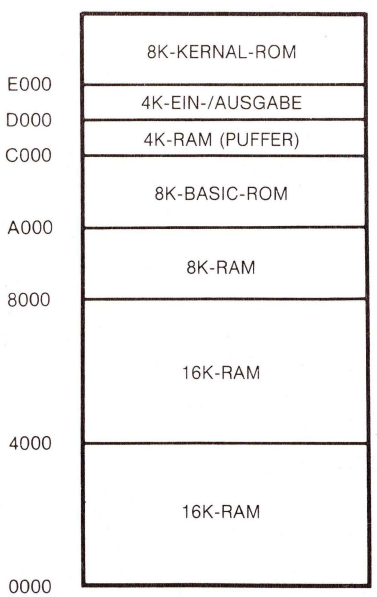
EIN-/AUSGABE IM DETAIL

D000–D3FF	VIC (Videosteuerung)	1K-Byte
D400–D7FF	SID (Musik-Synthesizer)	1K-Byte
D800–DBFF	Farb-RAM	1K-Nybble
DC00–DCFF	CIA1 (Tastatur)	256 Bytes
DD00–DDFF	CIA2 (Serieller Bus-User-Port/RS-232)	256 Bytes
DE00–DEFF	Offener Ein-/Ausgabeanschluß #1 (CP/M)	256 Bytes
DF00–DFFF	Offener Ein-/Ausgabeanschluß #2 (Diskette)	256 Bytes

Die beiden offenen Ein-/Ausgabeanschlüsse dienen der allgemeinen Ein-/Ausgabe, speziellen Ein-/Ausgabemodulen (z. B. IEEE) und wurden für den Z-80-Modul (CP/M Option) sowie für den Anschluß an ein schnelles Diskettensystem mit günstigem Kosten-Leistungs-Verhältnis entwickelt.
Dieses System sorgt für den "Automatikstart" des Programms bei Verwendung eines COMMODORE 64-Erweiterungsmoduls. Wenn die ersten neun Bytes des Modul-ROMs beginnend bei Adresse 32768 ($8000) bestimmte Daten enthalten, wird das ROM-Programm gestartet. Die ersten zwei Bytes müssen den Kaltstart-Vektor für das Programm enthalten. Die nächsten zwei Bytes in 32770 ($8002) enthalten den Warmstartvektor. Die folgenden 3 Bytes müssen die Buchstaben CBM enthalten, wobei für jeden Buchstaben Bit 7 gesetzt ist. Die letzten zwei Bytes müssen die Ziffern "80" in COMMODORE ASCII sein.

SPEICHERBELEGUNGEN DES COMMODORE 64

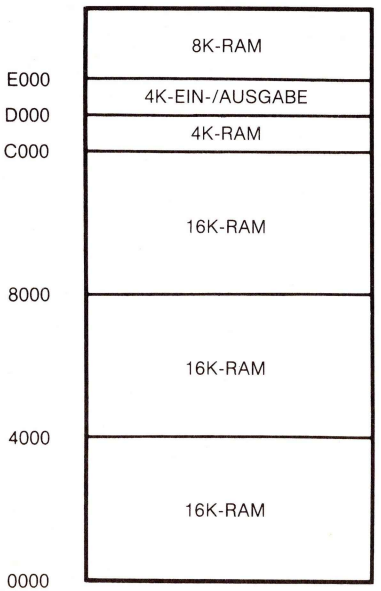
In den folgenden Schemata sind die möglichen Speicheranordnungen für den COMMODORE 64, der jeweilige Status der Leitungen und die entsprechende Verwendung der einzelnen Speicherverteilung aufgeführt.



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 1
HIRAM = 1
GAME = 1
EXROM = 1

Dies ist die Standard-BASIC-Speicherverteilung mit BASIC 2.0 und 38 KB durchgehenden Benutzer-RAM.



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 1
HIRAM = 0
GAME = 1
EXROM = X

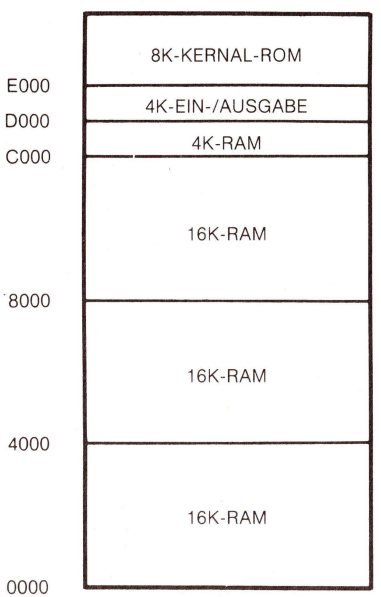
oder

LORAM = 1
HIRAM = 0
GAME = 0

(In dieser Speicherkonfiguration ist der Zeichen-ROM für die CPU nicht zugänglich.)

EXROM = 0

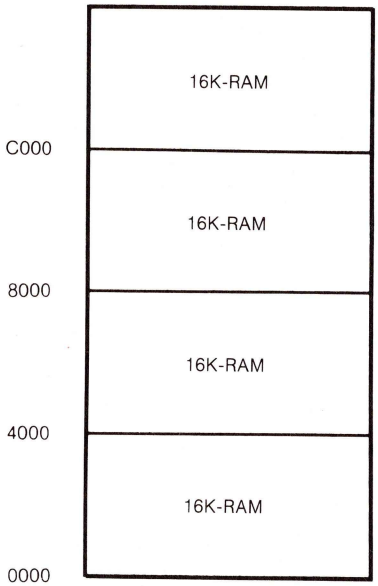
In dieser Konfiguration gibt es 60K-RAM sowie Ein-/Ausgaberegister. Der Besitzer muß seine eigenen Ein-/Ausgabetreiber-Routinen schreiben.



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 0
HIRAM = 1
GAME = 1
EXROM = X

Diese Konfiguration ist für das Arbeiten mit ladbaren Sprachen (einschl. CP/M) gedacht und hat 52K durchgehenden Benutzer-RAM, Ein-/Ausgaberegister und Ein-/Ausgabetreiber-Routinen.



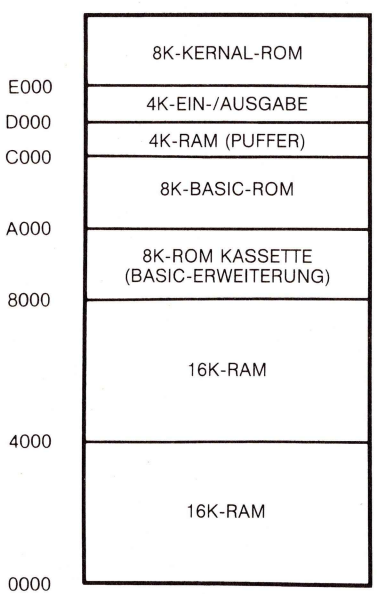
X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 0
HIRAM = 0
GAME = 1
EXROM = X

oder

LORAM = 0
HIRAM = 0
GAME = X
EXROM = 0

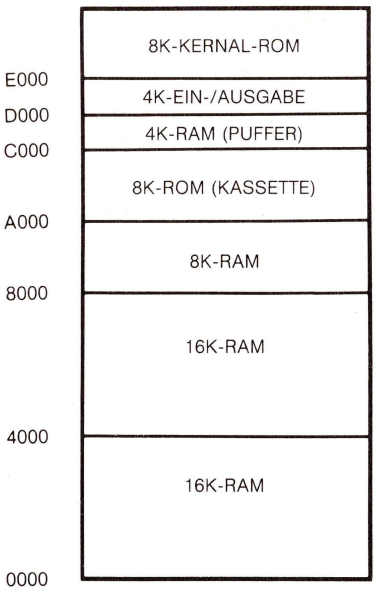
Diese Konfiguration erlaubt einen Zugriff auf den gesamten RAM-Bereich von 64K-Byte. Für jede Ein-/Ausgabe müssen die Ein-/Ausgabe-Speicherbereiche zurück in den Prozeßadreßbereich geschaltet werden.



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 1
HIRAM = 1
GAME = 0
EXROM = 0

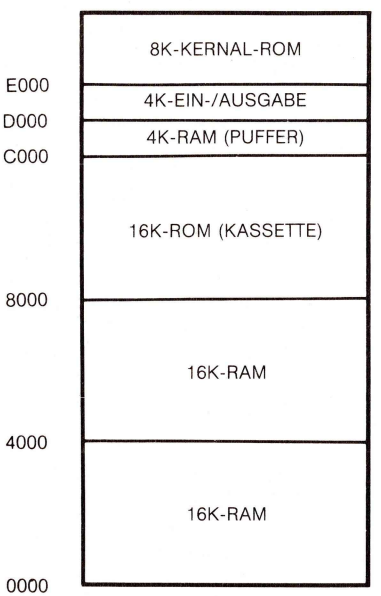
Dies ist der Standardaufbau eines BASIC-Systems mit BASIC-Erweiterungs-ROM. Diese Konfiguration hat 32K durchgehenden Benutzer-RAM und erlaubt eine BASIC-Erweiterung von bis zu 8K-Byte.



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 0
HIRAM = 1
GAME = 0
EXROM = 0

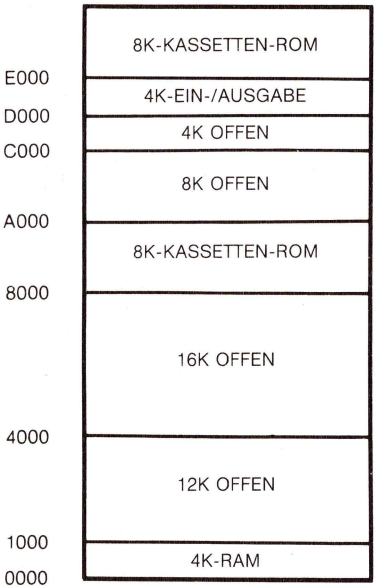
Diese Konfiguration hat 40K durchgehenden Benutzer-RAM und bis zu 8K-Byte für einen ROM-Anschluß für besondere ROM-Anwendungen, die kein BASIC erfordern.



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = 1
HIRAM = 1
GAME = 0
EXROM = 0

Diese Konfiguration hat 32K durchgehenden Benutzer-RAM und bis zu 16K-Byte für einen ROM-Anschluß für besondere ROM-Anwendungen, die kein BASIC erfordern (Textverarbeitung, andere Sprachen usw.).



X = NICHT BERÜCKSICHTIGEN
0 = NIEDRIG
1 = HOCH

LORAM = X
HIRAM = X
GAME = 0
EXROM = 1

Dies ist die ULTIMAX-Videospiel-Speicherkonfiguration. Beachten Sie, daß als 2K "Erweiterungs-RAM" für den ULTIMAX gegebenenfalls der RAM des Commodore 64 verwendet wird und der RAM im Modul ignoriert wird.

KERNAL

Auf dem Mikrocomputer-Sektor gibt es eine Frage, die Programmierer immer wieder beschäftigt: Was tun, wenn das Computer-Betriebssystem von der Herstellerfirma geändert wird? Langwierig erstellte Maschinensprache-Programme funktionieren möglicherweise nicht mehr und müssen grundlegend geändert werden. Um dieses Problem zu mindern, hat COMMODORE ein Prinzip entwickelt, um Programmierern die Arbeit zu erleichtern. Es handelt sich hierbei um den sog. **KERNAL.** Im wesentlichen ist KERNAL eine Standard-**SPRUNGTABELLE** für Eingabe, Ausgabe und Speicherverwaltungsprogramme im Betriebssystem. Bei einer Verbesserung des Systems können die Plätze der einzelnen Programme im ROM sich ändern. Die KERNAL-SPRUNGTABELLE wird jedoch auch stets entsprechend geändert.
Wenn Ihre Maschinensprache-Programme die Betriebssystemroutinen nur über den KERNAL benutzen, so können sie gegebenenfalls wesentlich einfacher gestaltet werden.
Der KERNAL ist das Betriebssystem des COMMODORE 64. Über ihn werden sämtliche Eingaben, Ausgaben sowie die Speicherverwaltung gesteuert.
Um Ihre Maschinensprache-Programme zu vereinfachen und sicherzustellen, daß die Programme aufgrund einer künftigen Verbesserung des Betriebssystems vom COMMODORE 64 nicht veralten, enthält der KERNAL eine Sprungtabelle. Durch die 39 Ein-/Ausgabe-Routinen und weitere Hilfsprogramme, die über diese Tabelle erreichbar sind, können Sie nicht nur Zeit sparen, sondern Ihre Programme von einem COMMODORE-Computer an den anderen anpassen.
Die Sprungtabelle befindet sich auf der letzten Speicherseite (Page) des gesamten Adreßraums.
Um die KERNAL-Sprungtabelle zu benutzen, geben Sie zunächst die erforderlichen Parameter für die KERNAL-Routine ein. Dann springen Sie über die **JSR-**Anweisung an die geeignete Stelle in die KERNAL-Sprungtabelle. Nach Beendigung der Routine überträgt der KERNAL die Steuerung wieder Ihrem Maschinensprache-Programm. Je nach verwendeter KERNAL-Routine werden Parameter durch bestimmte Register in Ihr Programm zurückgegeben. Die jeweiligen Adressen der einzelnen KERNAL-Routinen finden Sie in den Beschreibungen der KERNAL-Unterprogramme.
Warum benutzt man die Sprungtabelle überhaupt? Warum springt man nicht direkt in das entsprechende KERNAL-Unterprogramm? Das ist eine gute Frage. Die Sprungtabelle wird benutzt, damit Ihre Maschinensprache-Programme auch dann funktionieren, wenn der KERNAL oder der BASIC-Interpreter geändert werden. In künftigen Betriebssystemen können die Speicherplätze der einzelnen Routinen an unterschiedlichen Positionen im Adreßbereich liegen . . . Die Sprungtabelle arbeitet jedoch immer noch richtig!

KERNAL-FUNKTIONEN NACH EINSCHALTEN DER STROMVERSORGUNG

1) Nach Einschalten der Stromversorgung wird durch den KERNAL zunächst der Stapelzeiger rückgesetzt und danach der Dezimalmodus gelöscht.
2) Dann prüft KERNAL, ob in Adresse $8000 HEX (32768 in Dezimaldarstellung) ein ROM mit Automatikstart vorhanden ist. Ist dieses vorhanden, dann wird die normale Initialisierung unterbrochen und die Steuerung dem im ROM abgelegten Code übertragen. Ist ein solches ROM nicht vorhanden, wird die normale Systeminitialisierung fortgesetzt.
3) Als nächstes initialisiert der KERNAL alle Ein-/Ausgabe-Vorrichtungen. Der serielle Bus wird initialisiert. Die beiden Chips 6526 CIA werden für die Tastatur-Abfrage auf die geeigneten Werte gesetzt und der 60-Hz-Timer aktiviert. Der SID wird gelöscht. Die BASIC-Speicherkonfiguration wird gewählt und der Kassettenmotor abgeschaltet.
4) Als nächstes führt der KERNAL einen RAM-Test durch und setzt oben und unten die Speicherzeiger. Auch die Zero-Page wird initialisiert und der Kassettenpuffer eingerichtet.
   Die RAM-Test-Routine ist ein nicht löschender Test, der bei Adresse $0300 beginnt und dann in aufsteigender Reihenfolge arbeitet. Der obere RAM-Zeiger wird gesetzt, wenn der Test auf die erste Nicht-RAM-Adresse trifft. Der untere Speicherzeiger wird stets auf $0800 und der Bildschirm stets auf $0400 gesetzt.
5) Abschließend führt der KERNAL folgende Funktionen aus: Die Ein-/Ausgabe-vektoren werden auf die Standardwerte gesetzt. Der Bildschirm wird dann gelöscht und alle Bildschirm-Editor-Variablen rückgestellt. Für den BASIC-Start wird dann die indirekte Adresse in $A000 benutzt.

ARBEITEN MIT KERNAL

Beim Schreiben von Programmen in Maschinensprache ist es oft empfehlenswert, Betriebssystem-Routinen zu benutzen. Diese umfassen Ein-/Ausgabe, Zugriff auf den System-Taktgeber, Speicherverwaltung und ähnliche Funktionen. Es ist überflüssig, diese Routinen ständig neu zu schreiben. Durch den einfachen Zugriff auf das Betriebssystem wird daher das Programmieren in Maschinensprache beschleunigt.
Wie bereits erwähnt, stellt der KERNAL eine Sprungtabelle dar. Diese ist lediglich eine Ansammlung von JMP-Anweisungen zu zahlreichen Betriebssystem-Routinen.

Um mit einer KERNAL-Routine zu arbeiten, müssen Sie zunächst alle für diese Routine erforderlichen Vorbereitungen treffen. Wenn eine Routine z. B. zunächst den Aufruf einer anderen KERNAL-Routine erfordert, dann müssen Sie diese aufrufen. Setzt die Routine die Eingabe einer Zahl in den Akkumulator voraus, dann muß diese Zahl auch eingegeben sein. Werden die Bedingungen nicht erfüllt, dann können die Routinen natürlich auch nicht richtig arbeiten.
Nach dem Durchführen sämtlicher Vorbereitungen rufen Sie die Routine über die JSR-Anweisung auf. Alle zugänglichen KERNAL-Routinen sind wie Unterprogramme aufgebaut und müssen mit einer RTS-Anweisung enden. Wenn die KERNAL-Routine die entsprechende Aufgabe beendet hat, wird die Steuerung bei der Anweisung nach JSR wieder Ihrem Programm übertragen.
Viele KERNAL-Routinen zeigen bei Störungen Fehler-Codes im Statuswort oder Akkumulator an. Für gutes Programmieren und einen Erfolg der Maschinensprache-Programme dürfen diese Fehlerrückgaben nicht außer acht gelassen werden, da sonst das übrige Programm zerstört werden kann.
Dies ist alles, was Sie beim Arbeiten mit KERNAL zu tun haben. Gehen Sie einfach wie folgt vor:

1) Einrichten
2) Routinenaufruf
3) Fehlerbehandlung

Folgende Konventionen werden bei der Beschreibung von KERNAL-Routinen benutzt:

- **FUNKTIONSNAME:** Bezeichnung der Kernal-Routine.

- **AUFRUFADRESSE:** Dies ist die Aufrufadresse der KERNAL-Routine in Hexadezimaldarstellung.

- **KOMMUNIKATIONS-REGISTER:** Unter dieser Überschrift aufgeführte Register werden zur Übertragung von Parametern zu bzw. von KERNAL-Routinen benutzt.

- **VORBEREITUNGS-ROUTINEN:** Bei bestimmten KERNAL-Routinen ist zuvor eine Dateneingabe erforderlich. Die erforderlichen Routinen werden hier aufgeführt.

- **FEHLERMELDUNGEN:** Ist nach dem Abarbeiten einer KERNAL-Routine das CARRY-Flag gesetzt, so zeigt dies an, daß bei der Verarbeitung ein Fehler festgestellt wurde. Die Fehlerzahl ist im Akkumulator enthalten.

- **STAPELBEDARF:** Dies ist die tatsächliche Anzahl an Stapel-Bytes, die von der KERNAL-Routine benutzt werden.

- **BETROFFENE REGISTER:** Alle von KERNAL-Routinen benutzte Register werden hier aufgeführt.

- **BESCHREIBUNG:** Hier finden Sie eine kurze Funktionsbeschreibung der KERNAL-Routine.

Nachstehend werden die KERNAL-Routinen aufgelistet.

AUFRUFBARE KERNAL-ROUTINEN

NAME	ADRESSE		FUNKTION
	HEXA-DEZIMAL	DEZIMAL	
ACPTR	$FFA5	65445	Byte-Eingabe zum seriellen Port
CHKIN	$FFC6	65478	Kanal für Eingabe öffnen
CHKOUT	$FFC9	65481	Kanal für Ausgabe öffnen
CHRIN	$FFCF	65487	Zeicheneingabe
CHROUT	$FFD2	65490	Zeichenausgabe
CIOUT	$FFA8	65448	Byte-Ausgabe über den seriellen Bus
CINT	$FF81	65409	Bildschirm-Editor-Initialisierung
CLALL	$FFE7	65511	Schließen aller Kanäle und Dateien
CLOSE	$FFC3	65475	Schließen einer bestimmten logischen Datei
CLRCHN	$FFCC	65484	Schließen der Ein- und Ausgabekanäle
GETIN	$FFE4	65508	Zeichen aus Tastaturpuffer lesen
IOBASE	$FFF3	65523	Basisadreß-Rückmeldung der Ein-/Ausgabegeräte
IOINIT	$FF84	65412	Ein-/Ausgabeinitialisierung
LISTEN	$FFB1	65457	LISTEN-Befehl für Geräte am seriellen Bus
LOAD	$FFD5	65493	RAM laden von Peripherie
MEMBOT	$FF9C	65436	Unteren Speicherzeiger lesen/setzen
MEMTOP	$FF99	65433	Oberen Speicherzeiger lesen/setzen
OPEN	$FFC0	65472	Öffnen einer logischen Datei
PLOT	$FFF0	65520	X-, Y-Cursorposition lesen/setzen

NAME	ADRESSE		FUNKTION
	HEXA-DEZIMAL	DEZIMAL	
RAMTAS	$FF87	65415	RAM initialisieren, Kassettenpuffer einrichten, Bildschirm auf $0400 setzen
RDTIM	$FFDE	65502	Uhrzeit lesen
READST	$FFB7	65463	Ein-/Ausgabestatuswort lesen
RESTOR	$FF8A	65418	Standard Ein-/Ausgabevektoren rückstellen
SAVE	$FFD8	65496	RAM-Inhalt auf Peripheriegerät abspeichern
SCNKEY	$FF9F	65439	Tastatur abfragen
SCREEN	$FFED	65517	X-, Y-Bildschirmaufbau ermitteln
SECOND	$FF93	65427	Sekundäradresse nach LISTEN übertragen
SETLFS	$FFBA	65466	Logische, erste und Sekundäradresse setzen
SETMSG	$FF90	65424	KERNAL-Meldungen steuern
SETNAM	$FFBD	65469	Dateinamen festlegen
SETTIM	$FFDB	65499	Uhrzeit setzen
SETTMO	$FFA2	65442	Zeitsperre für seriellen Bus setzen
STOP	$FFE1	65505	Stop-Taste abfragen
TALK	$FFB4	65460	TALK-Befehl für Geräte am seriellen Bus
TKSA	$FF96	65430	Sekundäradresse nach TALK übertragen
UDTIM	$FFEA	65514	Uhrzeit inkrementieren
UNLSN	$FFAE	65454	UNLISTEN-Befehl für seriellen Bus
UNTLK	$FFAB	65451	UNTALK-Befehl für seriellen Bus
VECTOR	$FF8D	65421	Abspeichern von RAM

B-1. Funktionsname: ACPTR

Zweck: Daten vom seriellen Bus lesen
Aufrufadresse: $FFA5 (HEX) 65445 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: TALK, TKSA
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 13
Beeinflußte Register: .A, .X

Beschreibung: Diese Routine benutzen Sie, wenn Sie Informationen von einem Gerät am seriellen Bus, wie z. B. einer Diskette lesen wollen. Diese Routine liest direkt ein Datenbyte vom seriellen Bus. Dieses Datum wird in den Akkumulator übertragen. Als Vorbereitung muß zunächst die TALK-Routine aufgerufen werden. Über diese erhält der serielle Bus den Befehl für die Bus-Datenübertragung.
Wenn die Eingabevorrichtung einen Sekundärbefehl erfordert, muß dieser vor dem Aufruf über die KERNAL-Routine TKSA übertragen werden. Fehler werden im Statuswort rückgemeldet. Zum Lesen des Statusworts wird die READST-Routine benutzt.

Vorgehensweise:

0) Gerät am seriellen Bus für die Datenübertragung zum COMMODORE 64 vorbereiten. (KERNAL-Routinen, TALK und TKSA benutzen.)
1) Diese Routine aufrufen (über JSR).
2) Daten speichern oder benutzen.

BEISPIEL:

```
;GET A BYTE FROM THE BUS
JSR ACPTR
STA DATA
```

B-2. Funktionsname: CHKIN

Zweck: Kanal für Eingabe öffnen
Aufrufadresse: $FFC6 (HEX) 65478 (Dezimal)
Kommunikationsregister: .X
Vorbereitungsroutinen: (OPEN)
Fehlerrückmeldung:
Stapelbedarf: Keiner
Beeinflußte Register: .A, .X

Beschreibung: Jede über die KERNAL-Routine OPEN geöffnete logische Datei kann über diese Routine als Eingabekanal definiert werden. Natürlich muß es sich dabei um ein Eingabegerät handeln, da es sonst zu einem Fehler kommt und die Routine unterbrochen wird.
Werden die Daten nicht über die Tastatur eingegeben, dann muß diese Routine vor dem Arbeiten mit den KERNAL-Routinen CHRIN oder GETIN für die Dateneingabe zuvor aufgerufen werden. Soll die Eingabe über die Tastatur erfolgen und sind keine weiteren Eingabekanäle geöffnet, dann wird diese Routine und die OPEN-Routine nicht benötigt.
Wird diese Routine mit einem Gerät am seriellen Bus benutzt, dann wird über den Bus automatisch die Talk-Adresse (und die Sekundäradresse, wenn eine solche durch die OPEN-Routine festgelegt wurde) übertragen.

Vorgehensweise:

0) Logische Datei öffnen (gegebenenfalls dazugehörige Beschreibung durchlesen).
1) Register .X mit der Nummer der zu verwendenden logischen Datei laden.
2) Diese Routine aufrufen (über JSR).

Mögliche Fehler:

#3: Datei nicht offen.
#5: Gerät nicht vorhanden.
#6: Datei ist keine Eingabedatei.

BEISPIEL:

```
; PREPARE FOR INPUT FROM LOGICAL FILE 2
LDX #2
JSR CHKIN
```

B-3. Funktionsname: CHKOUT

Zweck: Kanal für Ausgabe öffnen
Aufrufadresse: $FFC9 (HEX) 65481 (Dezimal)
Kommunikationsregister: .X
Vorbereitungsroutinen: (OPEN)
Fehlerrückmeldungen: 0,3,5,7 (siehe READST)
Stapelbedarf: 4+
Beeinflußte Register: .A, .X

Beschreibung: Eine über die KERNAL-Routine OPEN erstellte logische Dateinummer kann als Ausgabekanal definiert werden. Natürlich muß es sich hierbei um ein Ausgabegerät handeln, da es sonst zu einem Fehler kommt und die Routine unterbrochen wird.
Ehe Daten zu einem Ausgabegerät übertragen werden, ist ein Aufruf dieser Routine erforderlich. Es sei denn, Sie wollen den Bildschirm des COMMODORE 64 als Ausgabegerät benutzen. Wird eine Bildschirmausgabe gewünscht und ist noch kein anderer Ausgabekanal definiert, dann werden diese Routine und die OPEN-Routine nicht benötigt.
Beim Öffnen des Kanals zum seriellen Bus überträgt diese Routine automatisch die durch die OPEN-Routine festgelegte LISTEN-Adresse (und gegebenenfalls eine Sekundäradresse).

Vorgehensweise:

Denken Sie daran: Diese Routine wird nicht zum Übertragen von Daten auf den Bildschirm benötigt.

0) Eine logische Dateinummer, eine LISTEN-Adresse und eine Sekundäradresse (falls erforderlich) über die KERNAL-Routine OPEN festlegen.
1) Register .X mit der in der OPEN-Anweisung benutzten logischen Dateinummer laden.
2) Diese Routine aufrufen (über JSR).

BEISPIEL:

```
LDX #3          ;DEFINE LOGICAL FILE 3 AS AN OUTPUT CHANNEL
JSR CHKOUT
```

Mögliche Fehler:

#3: Datei nicht offen.
#5: Gerät nicht vorhanden.
#7: Keine Ausgabedatei.

B-4. Funktionsname: CHRIN

Zweck: Zeicheneingabe
Aufrufadresse: $FFCF (HEX) 65487 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: (OPEN, CHKIN)
Fehlerrückmeldungen: 0 (siehe READST)
Stapelbedarf: 7+
Beeinflußte Register: .A, .X

Beschreibung: Über diese Routine wird ein Datenbyte von dem über die KERNAL-Routine CHKIN bereits als Eingabekanal festgelegten Kanal gelesen. Wurde CHKIN nicht zur Definition eines weiteren Eingabekanals benutzt, dann wird davon ausgegangen, daß sämtliche Daten über die Tastatur eingegeben wurden. Das Datenbyte wird in den Akkumulator übertragen. Nach dem Aufruf bleibt der Kanal offen.
Eingaben über die Tastatur werden auf besondere Weise gehandhabt. Zunächst wird der Cursor eingeschaltet und blinkt so lange, bis ein CR (carriage return) eingegeben wird. Alle Zeichen in der Zeile (max. 88 Zeichen) werden im BASIC-Eingabepuffer gespeichert. Diese Zeichen können einzeln aufgerufen werden, indem man diese Routine für jedes einzelne Zeichen aufruft. Nach dem "carriage return" wird die gesamte Zeile verarbeitet. Beim nachfolgenden Aufruf dieser Routine beginnt der gleiche Vorgang wieder von vorn, d. h., mit einem Blinken des Cursors.

Vorgehensweise:

VON DER TASTATUR
1) Ein Datenbyte durch diese Routine aufrufen.
2) Datenbyte speichern.
3) Prüfen, ob es sich um das letzte Datenbyte handelt (ist es ein CR?).
4) Wenn nicht, bei Schritt 1) fortsetzen.

BEISPIEL:

```
    LDY #$00        ;PREPARE THE .Y REGISTER TO STORE THE DATA
RD  JSR CHRIN
    STA DATA,Y      ;STORE THE YTH DATA BYTE IN THE YTH
                    ;LOCATION IN THE DATA AREA.
    INY
    CMP #CR         ;IS IT A CARRIAGE RETURN?
    BNE RD          ;NO, GET ANOTHER DATA BYTE
```

BEISPIEL:

```
JSR CHRIN
STA DATA
```

VON ANDERER PERIPHERIE

0) KERNAL-Routinen OPEN und CHKIN benutzen.
1) Diese Routine aufrufen (über JSR).
2) Daten speichern.

BEISPIEL:

```
JSR CHRIN
STA DATA
```

B-5. Funktionsname: CHROUT

Zweck: Zeichenausgabe
Aufrufadresse: $FFD2 (HEX) 65490 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: (CHKOUT, OPEN)
Fehlerrückmeldungen: 0 (siehe READST)
Stapelbedarf: 8+
Beeinflußte Register: .A

Beschreibung: Über diese Routine wird ein Zeichen zu einem bereits geöffneten Kanal ausgegeben. Vor dem Aufruf dieser Routine den Ausgabekanal durch die KERNAL-Routinen OPEN und CHKOUT bestimmen. Wird dieser Aufruf ausgelassen, dann erfolgt die Datenübertragung zum Standard-Ausgabegerät (Nummer 3, Bildschirm). Das auszugebende Datenbyte wird in den Akkumulator übertragen und diese Routine aufgerufen. Die Daten werden dann zum angegebenen Ausgabegerät übertragen. Nach dem Aufruf bleibt der Kanal geöffnet.

Anmerkung: Besondere Vorsicht ist geboten, wenn diese Routine für die Datenübertragung zu einem speziellen Gerät am seriellen Bus benutzt wird, da alle Daten zu allen offenen Bus-Ausgabekanälen übertragen werden. Ist dies nicht erwünscht, müssen alle Ausgabekanäle des Serienbusses bis auf den gewünschten Kanal durch die KERNAL-Routine CLRCHN geschlossen werden.

Vorgehensweise:

0) Gegebenenfalls KERNAL-Routine CHKOUT benutzen (siehe obige Beschreibung).
1) Auszugebende Daten in den Akkumulator laden.
2) Diese Routine aufrufen.

BEISPIEL:

```
;DUPLICATE THE BASIC INSTRUCTION CMD 4,"A";
LDX #4                          ;LOGICAL FILE #4
JSR CHKOUT                      ;OPEN CHANNEL OUT
LDA #'A'
JSR CHROUT                      ;SEND CHARACTER
```

B-6. Funktionsname: CIOUT

Zweck: Byte-Ausgabe über den seriellen Bus
Aufrufadresse: $FFA8 (HEX) 65448 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: LISTEN, (SECOND)
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 5
Beeinflußte Register: Keine

Beschreibung: Diese Routine wird für die Informationsübertragung zu einem Gerät am seriellen Bus benutzt. Durch Aufruf dieser Routine wird ein Datenbyte auf den seriellen Bus mit "handshake" übertragen. Vor dem Aufruf muß die KERNAL-Routine LISTEN benutzt werden, um das Gerät am seriellen Bus für den Datenempfang vorzubereiten. (Wird eine Sekundäradresse benötigt, dann muß diese über die KERNAL-Routine SECOND übertragen werden.)
Der Akkumulator wird dann mit einem Byte geladen, das als Datum über den seriellen Bus übertragen wird. Eine Vorrichtung muß für den Datenempfang bereit sein, da sonst das Statuswort ein "timeout" meldet. Bei dieser Routine wird stets ein Zeichen zwischengespeichert. Wird die KERNAL-Routine UNLSN zur Beendigung der Datenübertragung aufgerufen, so wird das im Puffer befindliche Zeichen mit einem EOI übertragen. Dann wird der Befehl UNLSN zum Gerät übertragen.

Vorgehensweise:

0) KERNAL-Routine LISTEN (und gegebenenfalls SECOND) benutzen.
1) Ein Datenbyte in den Akkumulator laden.
2) Zur Übertragung des Datenbytes diese Routine aufrufen.

BEISPIEL:

```
LDA #'X'                    ;SEND AN X TO THE SERIAL BUS
JSR CIOUT
```

B-7. Funktionsname: CINT

Zweck: Initialisierung von Bildschirmeditor und Video-Chip 6567
Aufrufadresse: $FF81 (HEX) 65409 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 4
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine wird der Video-Steuerchip 6567 im COMMODORE 64 initialisiert. Auch der KERNAL-Bildschirmeditor wird initialisiert. Diese Routine kann über ein Programm-Modul des COMMODORE 64 aufgerufen werden.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR CINT
JMP RUN                     ;BEGIN EXECUTION
```

B-8. Funktionsname: CLALL

Zweck: Schließen sämtlicher Dateien
Aufrufadresse: $FFE7 (HEX) 65511 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 11
Beeinflußte Register: .A, .X

Beschreibung: Über diese Routine werden alle offenen Dateien geschlossen. Beim Aufruf werden die Zeiger in der Tabelle der offenen Dateien rückgestellt und alle Dateien geschlossen. Die Routine CLRCHN wird automatisch zur Rückstellung der Ein-/Ausgabekanäle aufgerufen.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR CLALL  ;CLOSE ALL FILES AND SELECT DEFAULT I/O CHANNELS
JMP RUN    ;BEGIN EXECUTION
```

B-9. Funktionsname: CLOSE

Zweck: Schließen einer logischen Datei
Aufrufadresse: $FFC3 (HEX) 65475 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: 0,240 (siehe READST)
Stapelbedarf: 2+
Beeinflußte Register: .A, .X, .Y

Beschreibung: Diese Routine wird zum Schließen einer logischen Datei benutzt, nachdem alle Ein-/Ausgaben in bezug auf diese Datei beendet sind. Die Routine wird aufgerufen, nachdem der Akkumulator mit der entsprechenden logischen Dateinummer geladen wurde (gleiche Nummer, die beim Öffnen der Datei über die Routine OPEN benutzt wurde).

Vorgehensweise:

1) Entsprechende logische Dateinummer in den Akkumulator laden.
2) Diese Routine aufrufen.

BEISPIEL:

```
;CLOSE 15
LDA #15
JSR CLOSE
```

B-10. Funktionsname: CLRCHN

Zweck: Löschen von Ein-/Ausgabekanälen
Aufrufadresse: $FFCC (HEX) 65484 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen:
Stapelbedarf: 9
Beeinflußte Register: .A, .X

Beschreibung: Diese Routine wird zum Löschen aller offenen Kanäle und zur Rückstellung der Ein-/Ausgabekanäle auf die Standardwerte aufgerufen. Normalerweise erfolgt der Aufruf nach Öffnen anderer Ein-/Ausgabekanäle (z. B. Kassette oder Diskette) und Beendigung der entsprechenden Ein-/Ausgaben. Die Standard-Eingabegerätenummer ist 0 (Tastatur). Das Standardausgabegerät ist 3 (Bildschirm).
Ist einer der zu schließenden Kanäle der serielle Bus, so wird zunächst zum Löschen des Eingabekanals ein UNTALK-Signal oder zum Löschen des Ausgabekanals ein UNLISTEN-Signal übertragen. Wird diese Routine nicht aufgerufen (und bleiben die Serienbus-Anschlußgeräte empfangsbereit), dann können mehrere Geräte die gleichen Daten vom COMMODORE 64 gleichzeitig empfangen. Hierdurch könnte z. B. der Drucker für die Ausgabe und die Diskette für den Datenempfang eingesetzt werden. Auf diese Weise kann eine Diskettendatei direkt ausgedruckt werden.
Beim Ausführen der KERNAL-Routine CLALL wird diese Routine automatisch aufgerufen.

Vorgehensweise:

1) Diese Routine über die JSR-Anweisung aufrufen.

BEISPIEL:

```
JSR CLRCHN
```

B-11. Funktionsname: GETIN

Zweck: Ein Zeichen lesen
Aufrufadresse: $FFE4 (HEX) 65508 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: CHKIN, OPEN
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 7+
Beeinflußte Register: .A (.X, .Y)

Beschreibung: Handelt es sich bei dem Kanal um die Tastatur, dann nimmt dieses Unterprogramm ein Zeichen aus dem Tastaturpuffer und überträgt es als ACII-Wert in den Akkumulator. Ist der Puffer leer, dann wird der Wert 0 in den Akkumulator übertragen. Zeichen werden automatisch durch eine Tastatur-Abfrageroutine, die die Routine SCNKEY aufruft, in eine "Warteschlange" übertragen. Im Tastaturpuffer können max. 10 Zeichen gespeichert sein. Ist der Puffer gefüllt, dann werden das 11. und alle weiteren Zeichen solange überlesen, bis mind. 1 Zeichen aus der Warteschlange entfernt wurde.
Handelt es sich bei dem Kanal um RS-232, dann wird nur Register .A benutzt und ein einzelnes Zeichen wiedergegeben. Zum Überprüfen siehe READST. Handelt es sich bei dem Kanal um den seriellen Bus, die Kassette oder den Bildschirm, dann rufen Sie die BASIN-Routine auf.

Vorgehensweise:

1) Diese Routine über eine JSR-Anweisung aufrufen.
2) Prüfen, ob im Akkumulator eine 0 gespeichert ist (leerer Puffer).
3) Daten verarbeiten.

BEISPIEL:

```
;WAIT FOR A CHARACTER
WAIT JSR GETIN
CMP #0
BEQ WAIT
```

B-12. Funktionsname: IOBASE

Zweck: Festlegen des Ein-/Ausgabe-Speicherbereichs
Aufrufadresse: $FFF3 (HEX) 65523 (Dezimal)
Kommunikationsregister: .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen:
Stapelbedarf: 2
Beeinflußte Register: .X, .Y

Beschreibung: Über diese Routine werden im X- und Y-Register nieder- und höherwertige Bytes der Adresse des Speicherabschnitts abgespeichert, in dem sich die Ein-/Ausgaberegister befinden. Diese Adresse kann dann zusammen mit relativen Adressen für den Zugriff auf die Ein-/Ausgaberegister des COMMODORE 64 benutzt werden. Register .X enthält das untere Adreßbyte, Register .Y das obere Adreßbyte.
Durch diese Routine wird Kompatibilität zwischen dem COMMODORE 64, VC-20 und künftigen Modellen des COMMODORE 64 gewährleistet. Werden die Ein-/Ausgaberegister für ein Maschinenspracheprogramm durch Aufruf dieser Routine gesetzt, dann sind sie auch mit künftigen Versionen des COMMODORE 64, was KERNAL und BASIC angeht, kompatibel.

Vorgehensweise:

1) Diese Routine über die JSR-Anweisung aufrufen.
2) Das Register .X und .Y in aufeinanderfolgenden Plätzen speichern.
3) Die Verschiebung in Register .Y laden.
4) Auf diesen Ein-/Ausgabeplatz zugreifen.

BEISPIEL:

```
;SET THE DATA DIRECTION REGISTER OF THE USER PORT TO 0 (INPUT)
JSR IOBASE
STX POINT      ;SET BASE REGISTERS
STY POINT+1
LDY #2
LDA #0         ;OFFSET FOR DDR OF THE USER PORT
STA (POINT), Y ;SET DDR TO 0
```

B-13. Funktionsname: IOINIT

Zweck: Initialisieren von Ein-/Ausgabegeräten
Aufrufadresse: $FF84 (HEX) 65412 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen:
Stapelbedarf: Keiner
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine werden alle Ein-/Ausgabevorrichtungen und Routinen initialisiert. Sie wird normalerweise als Teil der Initialisierung eines Programmoduls des COMMODORE 64 aufgerufen.

BEISPIEL:

```
JSR IOINIT
```

B-14. Funktionsname: LISTEN

Zweck: LISTEN-Befehl für ein Gerät am seriellen Bus
Aufrufadresse: $FFB1 (HEX) 65457 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: Keiner
Beeinflußte Register: .A

Beschreibung: Über diese Routine wird Gerät am seriellen Bus für den Datenempfang vorbereitet. Eine Gerätenummer zwischen 0 und 31 wird vor dem Aufruf dieser Routine in den Akkumulator geladen. Über die LISTEN-Anweisung wird die Zahl Bit für Bit ODER-verknüpft und in eine LISTEN-Adresse umgewandelt. Dieses Datum wird dann als Befehl über den seriellen Bus übertragen. Das angegebene Gerät ist dann für den Datenempfang bereit.

Vorgehensweise:

1) Die gewünschte Gerätenummer in den Akkumulator laden.
2) Diese Routine über die JSR-Anweisung aufrufen.

BEISPIEL:

```
;COMMAND DEVICE #8 TO LISTEN
LDA #8
JSR LISTEN
```

B-15. Funktionsname: LOAD

Zweck: RAM laden von Peripherie
Aufrufadresse: $FFD5 (HEX) 65493 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: SETLFS, SETNAM
Fehlerrückmeldungen: 0,4,5,8,5, READST
Stapelbedarf: Keiner
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine werden Datenbytes direkt von einem beliebigen Eingabegerät in den Speicher des COMMODORE 64 geladen. Sie kann auch für einen Vergleich der vom Gerät stammenden Daten mit denen im Speicher benutzt werden, während die im RAM gespeicherten Daten unverändert bleiben (VERIFY).
Zum Laden wird der Akkumulator (.A) auf 0 und für VERIFY auf 1 gesetzt. Wird ein OPEN auf das Eingabegerät mit der Sekundäradresse (SA) 0 eingegeben, dann wird die Ladeadresse ignoriert. In diesem Fall müssen die Register .X und .Y die Startadresse enthalten. Wird die Sekundäradresse 1, 0 oder 2 gewählt, dann werden die Daten ab der durch die Ladeadresse gegebenen Position in den Speicher geladen. Diese Routine ermittelt die Adresse des obersten benutzten RAM-Platzes.
Vor dem Aufruf dieser Routine müssen die KERNAL-Routinen SETLFS und SETNAM aufgerufen werden.

Anmerkung: Ein LOAD über Tastatur (0), RS-232 (2) oder Bildschirm (3) ist nicht möglich.

Vorgehensweise:

0) Routine SETLFS und SETNAM aufrufen. Wird ein verschobenes Laden gewünscht, Routine SETLFS zum Übertragen der Sekundäradresse 0 benutzen.
1) Register .A zum Laden auf 0 und zum Überprüfen auf 1 setzen.
2) Wird Laden an eine bestimmte Adresse gewünscht, müssen Register .X und .Y auf die Lade-Startadresse gesetzt sein.
3) Die Routine über die JSR-Anweisung aufrufen.

BEISPIEL:

```
                ;LOAD A FILE FROM TAPE
                LDA   #DEVICE1          ;SET DEVICE NUMBER
                LDX   #FILENO           ;SET LOGICAL FILE NUMBER
                LDY   CMD1              ;SET SECONDARY ADDRESS
                JSR   SETLFS
                LDA   #NAME1-NAME       ;LOAD .A WITH NUMBER OF
                                        ;CHARACTERS IN FILE NAME
                LDX   #<NAME            ;LOAD .X AND .Y WITH
                                        ;ADDRESS OF
                LDY   #>NAME            ;FILE NAME
                JSR   SETNAM
                LDA   #0                ;SET FLAG FOR A LOAD
                LDX   #$FF              ;ALTERNATE START
                LDY   #$FF
                JSR   LOAD
                STX   VARTAB            ;END OF LOAD
                STY   VARTAB+1
                JMP   START
NAME            .BYT  'FILE NAME'
NAME 1          ;
```

B-16. Funktionsname: MEMBOT

Zweck: Setzen des Zeigers für das untere Speicherende
Aufrufadresse: $FF9C (HEX) 65436 (Dezimal)
Kommunikationsregister: .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: Keiner
Beeinflußte Register: .X, .Y

Beschreibung: Diese Routine wird zum Setzen des unteren Speicherzeigers benutzt. Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gesetzt, dann wird der Zeiger des untersten RAM-Bytes im Register .X und .Y wiedergegeben. Beim nicht erweiterten COMMODORE 64 ist der Zeigeranfangswert $0800 (2048 Dezimal). Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gelöscht (=0), dann werden die Werte des Register .X und .Y zum nieder- bzw. höherwertigen Byte des Zeigers, der den RAM-Anfang festlegt, übertragen.

Vorgehensweise:

LESEN DES RAM-ANFANGS
1) Übertrags-Flag setzen.
2) Diese Routine aufrufen.

SETZEN DES SPEICHERANFANGS
1) Übertrags-Flag löschen.
2) Diese Routine aufrufen.

BEISPIEL:

```
;MOVE BOTTOM OF MEMORY UP 1 PAGE
SEC             ;READ MEMORY BOTTOM
JSR MEMBOT
INY
CLC             ;SET MEMORY BOTTOM TO NEW VALUE
JSR MEMBOT
```

B-17. Funktionsname: MEMTOP

Zweck: Setzen des Zeigers für das obere Speicherende
Aufrufadresse: $FF99 (HEX) 65433 (Dezimal)
Kommunikationsregister: .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .X, .Y

Beschreibung: Über diese Routine wird das RAM-Ende gesetzt. Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gesetzt, dann wird der Zeiger des RAM-Endes in das Register .X und .Y geladen. Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gelöscht, dann werden die Inhalte von Register .X und .Y in den Speicherendezeiger geladen und so die Speicherendposition geändert.

BEISPIEL:

```
;DEALLOCATE THE RS-232 BUFFER
SEC
JSR MEMTOP   ;READ TOP OF MEMORY
DEX
CLC
JSR MEMTOP   ;SET NEW TOP OF MEMORY
```

B-18. Funktionsname: OPEN

Zweck: Öffnen einer logischen Datei
Aufrufadresse: $FFC0 (HEX) 65472 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: SETLFS, SETNAM
Fehlerrückmeldungen: 1,2,4,5,6,240, READST
Stapelbedarf: Keiner
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine kann eine logische Datei geöffnet werden. Nach Einrichten einer logischen Datei kann diese für Ein-/Ausgaben benutzt werden. Bei den meisten KERNAL-Ein-/Ausgaberoutinen wird diese Routine zum Erstellen der entsprechenden logischen Dateien aufgerufen. Für die Verwendung dieser Routine sind keine Parameter erforderlich, es müssen jedoch die KERNAL-Routinen SETLFS und SETNAM zuvor aufgerufen werden.

Vorgehensweise:

0) Routine SETLFS benutzen.
1) Routine SETNAM benutzen.
2) Diese Routine aufrufen.

BEISPIEL:

Dies ist eine Implementierung der BASIC-Anweisung: OPEN 15,8,15,"I/O".

```
        LDA #NAME2-NAME    ;LENGTH OF FILE NAME FOR SETLFS
        LDY #>NAME         ;ADDRESS OF FILE NAME
        LDX #<NAME
        JSR SETNAM
        LDA #15
        LDX #8
        LDY #15
        JSR SETLFS
        JSR OPEN
NAME    .BYT 'I/O'
NAME2
```

B-19. Funktionsname: PLOT

Zweck: Cursorposition lesen/setzen
Aufrufadresse: $FFF0 (HEX) 65520 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A, .X, .Y

Beschreibung: Ist beim Aufruf dieser Routine das Akkumulator-Übertrag-Flag gesetzt, dann wird die derzeitige Cursorposition auf dem Bildschirm (in X-/Y-Koordinaten) in Register .Y und .X geladen. Y ist die Spaltennummer des Cursorplatzes (0–79) und X die Reihennummer (0–24). Ist beim Aufruf das Übertragsbit gelöscht, dann bewegt sich der Cursor in die durch X,Y gegebene Position (entsprechend Register .Y und .X).

Vorgehensweise:

LESEN DER CURSORPOSITION
1) Übertrags-Flag setzen.
2) Diese Routine aufrufen.
3) X- und Y-Position aus Register .X bzw. .Y lesen.

SETZEN DER CURSORPOSITION
1) Übertrags-Flag löschen.
2) In Register .Y und .X die gewünschte Cursorposition schreiben.
3) Diese Routine aufrufen.

BEISPIEL:

```
; MOVE THE CURSOR TO ROW 10, COLUMN 5 (5,10)
LDX #10
LDY #5
CLC
JSR PLOT
```

B-20. Funktionsname: RAMTAS

Zweck: RAM-Test
Aufrufadresse: $FF87 (HEX) 65415 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A, .X., .Y

Beschreibung: Über diese Routine wird der RAM getestet und der obere bzw. untere Speicherzeiger gesetzt. Außerdem werden die Speicherplätze $0000 bis $0101 und $0200 bis $03FF gelöscht. Außerdem wird der Kassettenpuffer initialisiert und der Bildschirmanfang auf $0400 gesetzt. Normalerweise wird diese Routine als Teil der Initialisierung eines Programmoduls des COMMODORE 64 aufgerufen.

BEISPIEL:

```
JSR RAMTAS
```

B-21. Funktionsname: RDTIM

Zweck: Systemtaktgeber lesen
Aufrufadresse: $FFDE (HEX) 65502 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A, .X, .Y

Beschreibung: Diese Routine wird zum Lesen des Systemtaktgebers benutzt. Die Auflösung beträgt hierbei 1/60 s. Durch die Routine werden 3 Bytes ermittelt. Der Akkumulator enthält das signifikanteste (höchste) Byte, das X-Indexregister das nächste signifikante Byte und das Y-Indexregister das am wenigsten signifikante Byte.

BEISPIEL:

```
JSR RDTIM
STY TIME
STX TIME+1
STA TIME+2
. . .
TIME *=*+3
```

B-22. Funktionsname: READST

Zweck: Statuswort lesen
Aufrufadresse: $FFB7 (HEX) 65463 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A

Beschreibung: Über diese Routine wird der derzeitige Status der Ein-/Ausgabegeräte im Akkumulator abgelegt. Diese Routine wird normalerweise nach neuer Kommunikation mit einem Ein-/Ausgabegerät aufgerufen. Sie gibt Ihnen Informationen über den Gerätestatus oder Fehler, die während der Ein-/Ausgabe aufgetreten sind.
Die in den Akkumulator übertragenen Bits enthalten folgende Informationen: (Siehe nachstehende Tabelle).

ST BIT-POSITION	ST NUMERISCHER WERT	LESEN VON KASSETTE	SERIELLEN BUS/ SCHREIBEN/ LESEN	KASSETTE ÜBERPRÜFEN (VERIFY) + LADEN (LOAD)
0	1		Zeitsperre (timeout) Schreiben	
1	2		Zeitsperre (timeout) Lesen	
2	4	Kurzer Satz		Kurzer Satz
2	8	Langer Satz		Langer Satz
4	16	Nicht korrigierbarer Lesefehler		Nicht korrigierbarer Lesefehler
5	32	Prüfsummenfehler		Prüfsummenfehler
6	64	Dateiende	EOI-Leitung	
7	−128	Bandende	Gerät nicht vorhanden	Bandende

Vorgehensweise:

1) Diese Routine aufrufen.
2) Programminformation in Register .A decodieren.

BEISPIEL:

```
;CHECK FOR END OF FILE DURING READ
JSR READST
AND #64                    ;CHECK EOF BIT (EOF=END OF FILE)
BNE EOF                    ;BRANCH ON EOF
```

B-23. Funktionsname: RESTOR

Zweck: Normalzustand des Systems einstellen
Aufrufadresse: $FF8A (HEX) 65418 (Dezimal)
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine werden die Standardwerte sämtlicher vom KERNAL, von BASIC-Routinen und vom Interrupt benutzten Systemvektoren rückgestellt. Über die KERNAL-Routine VECTOR können die einzelnen Systemvektoren gelesen und aufbereitet werden.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR RESTOR
```

B-24. Funktionsname: SAVE

Zweck: Übertragen des Speicherinhalts auf ein entsprechendes Gerät
Aufrufadresse: $FFD8 (HEX) 65496 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: SETLFS, SETNAM
Fehlerrückmeldungen: 5,8,9, READST
Stapelbedarf: Keiner
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine wird ein Speicherbereich auf einem externen Speichermedium abgespeichert. Der Speicher wird ab der durch den Akkumulator festgelegten indirekten Adresse auf Seite 0 bis zu der in den Registern .X und .Y abgelegten Adresse abgespeichert. Er wird zu einer logischen Datei eines Ein-/Ausgabegerätes übertragen. Vor dem Aufruf dieser Routine sind die Routinen SETLFS und SETNAM zu verwenden. Zum Sichern auf Gerät Nr. 1 (Datassette™) ist jedoch kein Dateiname erforderlich. Wird versucht, auf eine andere Vorrichtung ohne Dateiname zu speichern, kommt es zu einem Fehler.

Anmerkung: Es ist nicht möglich, auf Gerät Nr. 0 (Tastatur), 2 (RS-232) und 3 (Bildschirm) zu speichern, da sonst eine Fehlermeldung erfolgt und die Speicherung gestoppt wird.

Vorgehensweise:

0) Routine SETLFS und SETNAM ausführen (wenn nicht auf Band oder Dateiname gespeichert werden soll).
1) Zwei aufeinanderfolgende Plätze der Zero-Page mit dem Zeiger laden, der auf den Anfang des abzuspeichernden Bereichs zeigt (standardmäßig kommt beim 6502 das niederwertige Byte zuerst und danach das höherwertige Byte).
2) Die Adresse des Zeigers in der Zero-Page in den Akkumulator laden.
3) Das niederwertige Byte bzw. das höherwertige Byte der Endadresse des abzuspeichernden Bereiches in Register .X und .Y laden.
4) Diese Routine aufrufen.

BEISPIEL:

```
LDA #1              ;DEVICE=1:CASSETTE
JSR SETLFS
LDA #0              ;NO FILE NAME
JSR SETNAM
LDA PROG            ;LOAD START ADDRESS OF SAVE
STA TXTTAB          ; (LOW BYTE)
LDA PROG+1
STA TXTTAB+1        ; (HIGH BYTE)
LDX VARTAB          ;LOAD .X WITH LOW BYTE OF END OF SAVE
LDY VARTAB+1        ;LOAD .Y WITH HIGH BYTE
LDA #<TXTTAB        ;LOAD ACCUMULATOR WITH PAGE 0 OFFSET
JSR SAVE
```

B-25. Funktionsname: SCNKEY

Zweck: Abfrage der Tastatur
Aufrufadresse: $FF9F (HEX) 65439 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: IOINIT
Fehlerrückmeldungen: Keine
Stapelbedarf: 5
Beeinflußte Register: .A, .X, .Y

Beschreibung: Über diese Routine wird die Tastatur des COMMODORE 64 abgefragt und so festgestellt, ob und wenn ja, welche Tasten gedrückt wurden. Dies ist die gleiche Routine, die bei jedem Interrupt aufgerufen wird. Nach Drücken einer Taste wird der entsprechende ASCII-Wert in den Tastaturpuffer geschrieben. Diese Routine wird nur aufgerufen, wenn die normale IRQ-Unterbrechung übergangen wird.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
GET   JSR SCNKEY     ;SCAN KEYBOARD
      JSR GETIN      ;GET CHARACTER
      CMP #0         ;IS IT NULL?
      BEQ GET        ;YES . . . SCAN AGAIN
      JSR CHROUT     ;PRINT IT
```

B-26. Funktionsname: SCREEN

Zweck: Ermitteln des Bildschirmformats
Aufrufadresse: $FFED (HEX) 65517 (Dezimal)
Kommunikationsregister: .X, .Y
Vorbereitungsroutinen: Keine
Stapelbedarf: 2
Beeinflußte Register: .X, .Y

Beschreibung: Diese Routine gibt das Bildschirmformat wieder, z. B. 40 Spalten in .X und 25 Zeilen in .Y. Sie kann benutzt werden, um zu bestimmen, auf welcher Maschine ein Programm läuft. Diese Funktion wurde für den COMMODORE 64 eingeführt, um Ihre Programme leichter mit anderen Geräten kompatibel machen zu können.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR SCREEN
STX MAXCOL
STY MAXROW
```

B-27. Funktionsname: SECOND

Zweck: Übertragen der Sekundäradresse für LISTEN
Aufrufadresse: $FF93 (HEX) 65427 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: LISTEN
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 8
Beeinflußte Register: .A

Beschreibung: Über diese Routine wird eine Sekundäradresse nach Aufruf der LISTEN-Routine zu einem Ein-/Ausgabegerät übertragen. Das Gerät ist danach empfangsbereit. Diese Routine kann nicht zur Übertragung einer Sekundäradresse nach dem Aufruf der TALK-Routine benutzt werden.
Eine Sekundäradresse wird normalerweise zur Übertragung von zusätzlichen Informationen vor der Ein-/Ausgabe benutzt.

Vorgehensweise:

1) Zu übertragende Sekundäradresse in den Akkumulator laden.
2) Diese Routine aufrufen.

BEISPIEL:

```
;ADDRESS DEVICE #8 WITH COMMAND (SECONDARY ADDRESS) #15
LDA #8
JSR LISTEN
LDA #15
JSR SECOND
```

B-28. Funktionsname: SETLFS

Zweck: Einrichten einer logischen Datei
Aufrufadresse: $FFBA (HEX) 65466 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: Keine

Beschreibung: Über diese Routine wird die logische Dateinummer, Geräteadresse und eine Sekundäradresse (Befehlsnummer) für andere KERNAL-Routinen eingerichtet.
Die logische Dateinummer wird vom System als eine Art Schlüssel für die durch die OPEN-Routinen erstellte Dateitabelle benutzt. Für die Geräteadressen stehen die Zahlen 0 bis 31 zur Verfügung. Die entsprechenden Codes der Peripherie-Geräte beim COMMODORE 64 lauten wie folgt:

ADRESSE	VORRICHTUNG
0	Tastatur
1	Datassette™ #1
2	RS-232C
3	Bildschirmanzeige
4	Drucker am seriellen Bus
8	Diskettenstation am seriellen Bus

Eine Gerätenummer von 4 oder darüber bezieht sich automatisch auf Geräte am seriellen Bus.
Ein Gerätebefehl wird als Sekundäradresse über den seriellen Bus übertragen, nachdem die Gerätenummer während des seriellen Handshakes übertragen wurde. Wird keine Sekundäradresse übertragen, dann muß Indexregister .Y auf 255 gesetzt sein.

Vorgehensweise:

1) Logische Dateinummer in den Akkumulator laden.
2) Gerätenummer in Index .X laden.
3) Befehl in Index .Y laden.

BEISPIEL:

```
FOR LOGICAL FILE 32, DEVICE #4, AND NO COMMAND:
LDA #32
LDX #4
LDY #255
JSR SETLFS
```

B-29. Funktionsname: SETMSG

Zweck: Ausgabe von Systemmeldungen
Aufrufadresse: $FF90 (HEX) 65424 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A

Beschreibung: Durch diese Routine werden Anzeigen von Fehler- und Steuermeldungen über den KERNAL gesteuert. Bei Aufruf der Routine werden je nach Inhalt des Akkumulators Fehler oder Steuermeldungen angezeigt. Eine Fehlermeldung ist z. B. FILE NOT FOUND. PRESS PLAY ON CASSETTE ist z. B. eine Steuermeldung.
Bit 6 und 7 bestimmen, woher die Meldungen kommen. Ist Bit 7 1 gesetzt, dann wird eine der Fehlermeldungen vom KERNAL angezeigt. Ist Bit 6 gesetzt, dann werden Steuermeldungen angezeigt.

Vorgehensweise:

1) Den Akkumulator auf den gewünschten Wert setzen.
2) Diese Routine aufrufen.

BEISPIEL:

```
LDA #$40
JSR SETMSG          ;TURN ON CONTROL MESSAGES
LDA #$80
JSR SETMSG          ;TURN ON ERROR MESSAGES
LDA #0
JSR SETMSG          ;TURN OFF ALL KERNAL MESSAGES
```

B-30. Funktionsname: SETNAM

Zweck: Festlegen des Dateinamens
Aufrufadresse: $FFBD (HEX) 65469 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: Keine
Stapelbedarf: Keiner
Beeinflußte Register: Keine

Beschreibung: Über diese Routine werden die Dateinamen für die Routinen OPEN, SAVE oder LOAD festgelegt. Die Dateinamenlänge wird in den Akkumulator geladen. Die Adresse des Dateinamens wird in die Register .X und .Y geladen. Für 6502 gilt standardmäßig das Format niederwertiges Byte/höherwertiges Byte. Die Adresse kann eine beliebige gültige Systemspeicheradresse sein, ab der der Dateiname als String gespeichert ist. Wird kein Dateiname gewünscht, wird der Akkumulator auf 0 gesetzt (0-Dateilänge). In diesem Fall können die Register .X und .Y auf eine beliebige Speicheradresse gesetzt werden.

Vorgehensweise:

1) Länge des Dateinamens in den Akkumulator laden.
2) Niederwertiges Adreßbyte des Dateinamens in Indexregister .X laden.
3) Höherwertiges Adreßbyte in Indexregister .Y laden.
4) Diese Routine aufrufen.

BEISPIEL:

```
LDA #NAME2-NAME      ;LOAD LENGTH OF FILE NAME
LDX #<NAME           ;LOAD ADDRESS OF FILE NAME
LDY #>NAME
JSR SETNAM
```

B-31. Funktionsname: SETTIM

Zweck: Systemtaktgeber setzen
Aufrufadresse: $FFDB (HEX) 65499 (Dezimal)
Kommunikationsregister: .A, .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: Keine

Beschreibung: Eine Uhr wird alle 1/60 s (=1"jiffy") von einer Interrupt-Routine aktualisiert. Die Uhr ist 3 Bytes "lang", so daß sie bis zu 5184000 "jiffies" (24 Stunden) anzeigen kann. Dann erfolgt die Rückstellung auf 0. Vor dem Aufruf dieser Routine muß in den Akkumulator das signifikanteste Byte, in Indexregister .X das nächste signifikante Byte und in Indexregister .Y das am wenigsten signifikante Byte der Ausgangs-Zeiteinstellung (in jiffies) eingegeben werden.

Vorgehensweise:

1) Das signifikanteste Byte der 3-Byte-Zahl in den Akkumulator laden.
2) Das nächste Byte in Register .X laden.
3) Das am wenigsten signifikante Byte in Register .Y laden.
4) Diese Routine aufrufen.

BEISPIEL:

```
;SET THE CLOCK TO 10 MINUTES = 3600 JIFFIES
LDA #0              ; MOST SIGNIFICANT
LDA #>3600
LDY #<3600          ; LEAST SIGNIFICANT
JSR SETTIM
```

B-32. Funktionsname: SETTMO

Zweck: Setzen des Timeout-Flags für den IEEE-Bus
Aufrufadresse: $FFA2 (HEX) 65442 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: Keine

Anmerkung: Diese Routine wird ausschließlich mit einer zusätzlichen IEEE-Karte benutzt!

Beschreibung: Durch diese Routine wird das Timeout-Flag für den IEEE-Bus gesetzt. Ist dieses Kennzeichen gesetzt, dann wartet der COMMODORE 64 64ms auf die Meldung eines Geräts am IEEE-Port. Antwortet das Gerät nicht auf das DAV-Signal (gültige Datenadresse) des COMMODORE 64 innerhalb dieses Zeitraums, dann erkennt der Computer eine Fehlerbedingung und verläßt die Handshake-Sequenz. Ist beim Aufruf dieser Routine Bit 7 im Akkumulator auf 0 gesetzt, dann sind Timeouts wirksam. Entsprechend sind Timeouts unwirksam, wenn Bit 7 auf 1 gesetzt ist.

Anmerkung: Der COMMODORE 64 benutzt diese Timeout-Routine nur im Zusammenhang mit der IEEE-Karte.

Vorgehensweise:

SETZEN DES TIMEOUT-FLAGS

1) Bit 7 des Akkumulators auf 0 setzen.
2) Diese Routine aufrufen.

RÜCKSTELLEN DES TIMEOUT-FLAGS

1) Bit 7 des Akkumulators auf 1 setzen.
2) Diese Routine aufrufen.

BEISPIEL:

```
;DISABLE TIMEOUT
LDA #0
JSR SETTMO
```

B-33. Funktionsname: STOP

Zweck: Abfrage der STOP -Taste
Aufrufadresse: $FFE1 (HEX) 65505 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: Keiner
Beeinflußte Register: .A, .X.

Beschreibung: Wurde während eines UDTIM-Aufrufs die STOP -Taste gedrückt, dann wird nach Aufruf dieser Routine das Z-Flag gesetzt. Darüber hinaus werden die Kanäle auf die Standardwerte zurückgesetzt. Alle anderen Flags bleiben unverändert. War die STOP -Taste nicht gedrückt, enthält der Akkumulator 1 Byte, das die letzte Reihe der Tastatur-Abfrage wiedergibt. Auf diese Weise kann der Bediener auch prüfen, ob bestimmte andere Tasten gedrückt wurden.

Vorgehensweise:

0) Vor dieser Routine muß UDTIM aufgerufen werden.
1) Diese Routine aufrufen.
2) Auf 0-Flag hin überprüfen.

BEISPIEL:

```
JSR UDTIM    ;SCAN FOR STOP
JSR STOP
BNE *+5      ;KEY NOT DOWN
JMP READY    ;= . . . STOP
```

B-34. Funktionsname: TALK

Zweck: TALK-Befehl für ein Gerät am seriellen Bus
Aufrufadresse: $FFB4 (HEX) 65460 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 8
Beeinflußte Register: .A

Beschreibung: Um mit dieser Routine zu arbeiten, muß zunächst eine Gerätenummer zwischen 0 und 31 in den Akkumulator geladen werden. Nach dem Aufruf wird dann Bit für Bit durch diese Routine ODER-verknüpft, um die Gerätenummer in eine TALK-Adresse umzuwandeln. Dieses Datum wird dann als Befehl über den seriellen Bus übertragen.

Vorgehensweise:

1) Gerätenummer in den Akkumulator laden.
2) Diese Routine aufrufen.

BEISPIEL:

```
;COMMAND DEVICE #4 TO TALK
LDA #4
JSR TALK
```

B-35. Funktionsname: TKSA

Zweck: Übertragen einer Sekundäradresse zu einem Gerät, das den TALK-Befehl erhalten hat
Aufrufadresse: $FF96 (HEX) 65430 (Dezimal)
Kommunikationsregister: .A
Vorbereitungsroutinen: TALK
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 8
Beeinflußte Register: .A

Beschreibung: Diese Routine überträgt eine Sekundäradresse über den Seriebus zu einem TALK-Gerät. Beim Aufruf dieser Routine muß im Akkumulator eine Zahl zwischen 0 und 31 geladen sein. Diese Zahl wird dann als Sekundär-Adreßbefehl über den seriellen Bus gesandt. Zuvor ist unbedingt die TALK-Routine aufzurufen. TKSA ist nicht nach LISTEN wirksam.

Vorgehensweise:

0) TALK-Routine benutzen.
1) Sekundäradresse in den Akkumulator laden.
2) Diese Routine aufrufen.

BEISPIEL:

```
;TELL DEVICE #4 TO TALK WITH COMMAND #7
LDA #4
JSR TALK
LDA #7
JSR TALKSA
```

B-36. Funktionsname: UDTIM

Zweck: Aktualisierung des Systemtaktgebers
Aufrufadresse: $FFEA (HEX) 65514 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A, .X

Beschreibung: Über diese Routine wird die Systemuhr aktualisiert. Normalerweise wird sie alle 1/60 s von der normalen KERNAL-Interrupt-Routine aufgerufen. Arbeitet das Benutzer-Programm mit eigenen Interrupts, dann muß zur Zeitaktualisierung diese Routine aufgerufen werden.
Soll weiterhin die Taste STOP wirksam bleiben, dann muß die STOP-Tastenroutine aufgerufen werden.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR UDTIM
```

B-37. Funktionsname: UNLSN

Zweck: Übertragung eines UNLISTEN-Befehls
Aufrufadresse: $FFAE (HEX) 65454 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 8
Beeinflußte Register: .A

Beschreibung: Über diese Routine erhalten alle Geräte am seriellen Bus den Befehl, den Datenempfang vom COMMODORE 64 zu beenden. Durch Aufruf dieser Routine wird ein UNLISTEN-Befehl über den seriellen Bus übertragen. Hierbei werden nur die Geräte beeinflußt, die zuvor einen LISTEN-Befehl erhalten haben. Normalerweise wird diese Routine benutzt, nachdem der COMMODORE 64 die Datenübertragung zu einem externen Gerät beendet hat. Nach dem UNLISTEN-Befehl sind die Geräte nicht mehr an den seriellen Bus angeschlossen und stehen für andere Zwecke zur Verfügung.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR UNLSN
```

B-38. Funktionsname: UNTLK

Zweck: Übertragung eines UNTALK-Befehls
Aufrufadresse: $FFAB (HEX) 65451 (Dezimal)
Kommunikationsregister: Keine
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Siehe READST
Stapelbedarf: 8
Beeinflußte Register: .A

Beschreibung: Über diese Routine wird ein UNTALK-Befehl über den seriellen Bus übertragen. Alle Befehle, die zuvor einen TALK-Befehl erhalten hatten, beenden dann die Datenübertragung.

Vorgehensweise:

1) Diese Routine aufrufen.

BEISPIEL:

```
JSR UNTLK
```

B-39. Funktionsname: VECTOR

Zweck: Verwaltung der RAM-Vektoren
Aufrufadresse: $FF8D (HEX) 65421 (Dezimal)
Kommunikationsregister: .X, .Y
Vorbereitungsroutinen: Keine
Fehlerrückmeldungen: Keine
Stapelbedarf: 2
Beeinflußte Register: .A, .X, .Y

Beschreibung: Diese Routine verwaltet alle im RAM gespeicherten Sprungvektoren. Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gesetzt, dann wird der derzeitige Inhalt der RAM-Vektoren in einer Liste gespeichert, deren Adresse durch die Inhalte der Register .X und .Y gegeben ist.
Ist beim Aufruf der Routine der Übertrag gelöscht, dann wird die durch Register .X und .Y in ihrer Lage gegebene Liste auf die System-RAM-Vektoren übertragen.

> **Anmerkung:** Beim Arbeiten mit dieser Routine ist äußerste Vorsicht geboten. Zunächst sollte der gesamte Vektor-Inhalt in den Benutzerbereich gelesen, die gewünschten Vektoren geändert und danach dieser Inhalt zurück in die Systemvektoren übertragen werden.

Vorgehensweise:

LESEN DER SYSTEM-RAM-VEKTOREN

1) Übertrag setzen.
2) Register .X und .Y auf die gewünschte Vektor-Adresse zeigen lassen.
3) Diese Routine aufrufen.

LADEN DER SYSTEM-RAM-VEKTOREN

1) Übertragsbit löschen.
2) Register .X und .Y auf die zu ladende RAM-Adreß-Vektorliste zeigen lassen.
3) Diese Routine aufrufen.

BEISPIEL:

```
;CHANGE THE INPUT ROUTINES TO NEW SYSTEM
LDX #<USER
LDY #>USER
SEC
JSR VECTOR      ;READ OLD VECTORS
LDA #<MYINP     ;CHANGE INPUT
STA USER+10
LDA #>MYINP
STA USER+11
LDX #<USER
LDY #>USER
CLC
JSR VECTOR      ;ALTER SYSTEM
...
USER *=*+26
```

FEHLERMELDUNGEN

Nachstehend finden Sie eine Liste der Fehlermeldungen, die beim Arbeiten mit den KERNAL-Routinen auftreten können. Kommt es zu einem dieser Fehler, dann wird das Übertragsbit des Akkumulators gesetzt und die Zahl der Fehlermeldungen in den Akkumulator übertragen.

Anmerkung: Einige KERNAL-Ein-/Ausgaberoutinen arbeiten nicht mit diesen Codes der Fehlermeldungen. Die Fehler werden statt dessen durch die KERNAL-Routine READST identifiziert.

NUMMER	BEDEUTUNG
0	Routine durch STOP -Taste beendet
1	Zu viele offene Dateien
2	Datei bereits offen
3	Datei nicht offen
4	Datei nicht gefunden
5	Gerät nicht vorhanden
6	Keine Eingabe-Datei
7	Keine Ausgabe-Datei
8	Dateiname fehlt
9	Unzulässige Gerätenummer
240	Speicherende verändert (RS-232C)

ARBEITEN MIT MASCHINENSPRACHE UND BASIC

Es gibt verschiedene Arten, wie beim COMMODORE 64 BASIC UND MASCHINENSPRACHE benutzt werden können. Hierzu gehören CBM-BASIC-Sonderanweisungen sowie spezielle Adressen. Um auf der Grundlage von BASIC beim COMMODORE 64 die Maschinensprache-Routinen zu benutzen, gibt es fünf verschiedene Möglichkeiten.

1) BASIC-Anweisung SYS
2) BASIC-Funktion USR
3) Änderung eines RAM-Ein-/Ausgabevektors
4) Änderung eines RAM-Unterbrechungsvektors
5) Änderung der CHRGET-Routine

1) Durch die BASIC-Anweisung **SYS X** erfolgt ein **Sprung** zum Maschinensprache-Unterprogramm bei Adresse X. Diese Routine muß mit einer **RTS**-Anweisung (Rückkehr vom Unterprogramm) enden. Hierdurch wird die Kontrolle wieder an BASIC übertragen. Parameter werden normalerweise zwischen Maschinensprache-Routine und BASIC-Programme über die BASIC-Anweisungen PEEK und POKE sowie die entsprechenden Maschinensprache-Befehle übertragen.
   SYS ist ein sehr wirksamer Befehl, um BASIC mit Maschinensprache zu kombinieren. Die Parameterübertragung wird durch PEEK und POKE erleichtert. Ein Programm kann mehrere SYS-Anweisungen für unterschiedliche (oder für dasselbe) Maschinensprache-Programm enthalten.
2) Durch die BASIC-Funktion **USR(X)** wird die Steuerung zum Maschinensprache-Unterprogramm übertragen, das sich an der in den Speicherplätzen 785 und 786 gespeicherten Adresse befindet. (Die Adresse ist entsprechend dem Standard-Format niederwertiges Byte/höherwertiges Byte gespeichert.) Der Wert X wird über den Gleitpunktakkumulator #1, der bei Adresse $61 beginnt (siehe Speicherbelegung für weitere Einzelheiten), zum Maschinensprache-Unterprogramm übertragen. Durch Eingabe in den Gleitpunkt-Akkumulator kann ein Wert zurück zum BASIC-Programm übergeben werden. Die Maschinensprache-Routine muß für die Rückkehr zu BASIC mit einer RTS-Anweisung enden.
3) Alle Ein-/Ausgaben oder BASIC-Routinen, auf die die Vektor-Tabelle von Seite 3 (siehe **ADRESSIERARTEN, ZERO-PAGE)** zugreift, können durch Benutzer-Code verschoben oder geändert werden. Jeder 2-Byte-Vektor besteht aus einer niederwertigen und einer höherwertigen Byte-Adresse, die vom Betriebssystem benutzt wird.

Vektoren können am zuverlässigsten über die **KERNAL-Vektor**-Routine geändert werden. Einzelne Vektoren lassen sich jedoch auch durch POKE-Anweisungen ändern. Ein neuer Vektor zeigt auf eine vom Benutzer vorbereitete Routine, die die Standardsystem-Routine ersetzen oder erweitern soll. Bei Ausführung des geeigneten BASIC-Befehls wird die Benutzer-Routine ausgeführt. Ist danach eine Ausführung der normalen System-Routine erforderlich, dann muß das Programm zu der Adresse springen (JMP), die zuvor im Vektor enthalten war. Anderenfalls muß die Routine am Ende eine RTS-Anweisung enthalten, um die Steuerung zurück an BASIC zu übertragen.

4) Der **Hardware-Interrupt-Vektor (IRQ)** kann geändert werden. Alle 1/60 s überträgt das Betriebssystem die Steuerung der durch diesen Vektor bestimmten Routine. Der KERNAL benutzt dies normalerweise zur Zeitberechnung, zur Abfrage der Tastatur usw. Wird diese Technik eingesetzt, dann sollten Sie stets die Steuerung der normalen **IRQ**-Routine übertragen, wenn nicht die Austausch-Routine zur Handhabung des CIA-Chips vorbereitet wurde. (Denken Sie daran, daß die Routine mit **RTI** (Rückkehr vom Interrupt) enden muß, wenn das CIA durch diese Routine kontrolliert wird.)
   Diese Methode ist sehr nützlich für Aufgaben, die gleichzeitig mit einem BASIC-Programm ablaufen sollen. Sie ist jedoch die schwierigste.

> **Anmerkung:** VOR ÄNDERUNG DIESES VEKTORS MUSS DER INTERRUPT ABGESCHALTET WERDEN.

5) BASIC benutzt die **CHRGET**-Routine, um jedes einzelne Zeichen oder "tokens" zu lesen.
   Auf diese Weise können leicht neue BASIC-Befehle hinzugefügt werden. Natürlich muß jeder neue Befehl über eine vom Benutzer geschriebene Maschinensprache-Unterroutine ausgeführt werden. Am einfachsten ist die Angabe eines Zeichens (z. B. @), das vor jedem neuen Befehl stehen wird. Die neue Routine CHRGET sucht nach diesem Sonderzeichen. Ist dies nicht vorhanden, dann wird die Steuerung zur normalen BASIC-Routine CHRGET übertragen. Wird das Sonderzeichen jedoch gefunden, dann wird der neue Befehl durch Ihr Maschinensprache-Programm interpretiert und ausgeführt. Hierdurch wird vermieden, daß sich die Ausführungszeit durch die Suche nach zusätzlichen Befehlen verzögert. Diese Technik nennt man häufig auch "wedge".

WO STEHEN MASCHINENSPRACHE-ROUTINEN?

Beim COMMODORE 64 liegt der beste Platz für Maschinensprache-Routinen bei $C000 bis $CFFF, vorausgesetzt, diese Routinen sind kürzer als 4K-Byte. Dieser Speicherbereich wird nicht durch BASIC beeinflußt.
Kann oder soll das Maschinensprache-Programm aus irgendeinem Grund nicht an Platz $C000 stehen (z. B. wenn die Routine länger als 4K-Byte ist), dann muß ein gewisser Bereich am Anfang des BASIC-Speichers für die Routine reserviert werden. Das obere Speicherende liegt normalerweise bei $9FFF. Es kann über die KERNAL-Routine **MEMTOP** oder durch folgende BASIC-Anweisungen geändert werden:

```
10 POKE51,L:POKE52,H:POKE55,L:POKE56,H:CLR
```

Hierbei kennzeichnen H und L den höher- und niederwertigen Byte des neuen Speicherendes. Um z. B. den Bereich von $9000 bis $9FFF für die Maschinensprache zu reservieren, geben Sie folgendes ein:

```
10 POKE51,0:POKE52,144:POKE55,0:POKE56,144:CLR
```

WIE WIRD MASCHINENSPRACHE EINGEGEBEN?

Es gibt drei verschiedene Arten, um Maschinensprache-Programme zu einem BASIC-Programm hinzuzufügen:

1) DATA-ANWEISUNGEN:

Maschinensprache-Routinen lassen sich in DATA-Anweisungen ablegen und können zu Beginn des Programms in den Speicher gePOKEt werden. Dies ist die einfachste Methode. Es brauchen nicht extra Programmteile abgespeichert zu werden. Auch die Fehlersuche ist relativ einfach. Der Nachteil liegt jedoch darin, daß mehr Speicherkapazität benötigt wird und daß das POKEn des Programms einige Zeit dauert. Diese Methode eignet sich daher nur für kleinere Routinen.

BEISPIEL:

```
10 RESTORE:FORX=1TO9:READA:POKE12*4096+X,A:NEXT
.
.
.
BASIC PROGRAM
.
.
.
1000 DATA 161,1,204,204,204,204,204,204,96
```

2) MASCHINENSPRACHE-MONITOR (MONITOR 64):

Über dieses Programm können Sie ein Programm entweder in hexadezimalen oder symbolischen Codes (Mnemonics) eingeben und den Speicherbereich, der das Programm enthält, auf Kassette oder Diskette abspeichern. Vorteile sind hierbei die leichtere Eingabe von Maschinensprache-Programmen, Fehlerbeseitigungs-Funktionen sowie ein bedeutend schnelleres Abspeichern und Laden. Von Nachteil ist jedoch, daß zu Beginn die Maschinensprache-Routine stets über ein BASIC-Programm von Kassette oder von Diskette geladen werden muß. (Bezüglich weiterer Einzelheiten über MONITOR 64 siehe Kapitel "Maschinensprache".)

BEISPIEL:

Nachstehend sehen Sie ein BASIC-Programmbeispiel, das eine durch MONITOR 64 vorbereitete Maschinensprache-Routine benutzt. Die Routine muß dazu auf Kassette gespeichert vorliegen.

```
10 IF FLAG=1 THEN 20
15 FLAG=1:LOAD "MACHINE LANGUAGE ROUTINE NAME",1,1
20
.
.
.
REST OF BASIC PROGRAM
```

3) ASSEMBLER

Die Vorteile sind ähnlich wie bei der Verwendung eines Maschinensprache-Monitors. Die Programme lassen sich jedoch sogar noch leichter eingeben.

SPEICHERBELEGUNG DES COMMODORE 64

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
D6510	0000	0	6510 Datenrichtungsregister
P6510	0001	1	6510 8-Bit-Ein-/Ausgaberegister
	0002	2	Nicht benutzt
ADRAY1	0003–0004	3–4	Sprungvektor: Umwandlung Gleitpunktzahl/Ganze Zahl
ADRAY 2	0005–0006	5–6	Sprungvektor: Umwandlung Ganze Zahl/Gleitpunktzahl
CHARAC	0007	7	Suchzeichen
ENDCHR	0008	8	Flag: Suchen nach einem Anführungszeichen am Ende eines Strings
TRMPOS	0009	9	Bildschirmspalte ab letztem TAB
VERCK	000A	10	0 = LOAD, 1 = VERIFY
COUNT	000B	11	Eingabepufferzeiger, Anzahl der Elemente
DIMFLG	000C	12	Flag: Standard-Felddimensionierung
VALTYP	000D	13	Datentyp: $FF = String, $00 = Numerisch
INTFLG	000E	14	Datentyp: $80 = Ganze Zahl, $00 = Gleitpunktzahl
GARBFL	000F	15	Flag: DATAs lesen/LIST auflisten "garbage collection"
SUBFLG	0010	16	Flag: Benutzerfunktionsaufruf
INPFLG	0011	17	Flag: $00 = INPUT, $40 = GET, $98 = READ
TANSGN	0012	18	Flag: Vorzeichen des TAN/Flag für Gleichheit bei Vergleich
	0013	19	Flag: INPUT-Kommentar
LINNUM	0014–0015	20–21	Ganzzahliger Wert
TEMPPT	0016	22	Zeiger: Temporärer Stringstapel
LASTPT	0017–0018	23–24	Letzte Stringadresse
TEMPST	0019–0021	25–33	Stapel für temporäre Strings
INDEX	0022–0025	34–37	Bereich für Hilfszeiger

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
RESHO	0026–002A	38–42	Gleitpunktergebnis der Multiplikation
TXTTAB	002B–002C	43–44	Zeiger: Anfang BASIC-Text
VARTAB	002D–002E	45–46	Zeiger: Anfang BASIC-Variablen
ARYTAB	002F–0030	47–48	Zeiger: Anfang BASIC-Felder
STREND	0031–0032	49–50	Zeiger: Ende BASIC-Felder (+1)
FRETOP	0033–0034	51–52	Zeiger: Anfang der String-Speicherung
FRESPC	0035–0036	53–54	Hilfszeiger für Strings
MEMSIZ	0037–0038	55–56	Zeiger: Oberste BASIC-Adresse
CURLIN	0039–003A	57–58	Derzeitige BASIC-Zeilennummer
OLDLIN	003B–003C	59–60	Vorherige BASIC-Zeilennummer
OLDTXT	003D–003E	61–62	Zeiger: BASIC-Anweisung für CONT
DATLIN	003F–0040	63–64	Derzeitige DATA-Zeilennummer
DATPTR	0041–0042	65–66	Zeiger: Derzeitige DATA-Adresse
INPPTR	0043–0044	67–68	Vektor: INPUT-Routine
VARNAM	0045–0046	69–70	Derzeitiger BASIC-Variablenname
VARPNT	0047–0048	71–72	Adresse der aktuellen Variablen
FORPNT	0049–004A	73–74	Variablenzeiger für FOR/NEXT
	004B–0060	75–96	Zwischenspeicher für BASIC-Zeiger/Daten
FACEXP	0061	97	Gleitpunktakkumulator #1: Exponent
FACHO	0062–0065	98–101	Gleitpunktakkumulator #1: Mantisse
FACSGN	0066	102	Gleitpunktakkumulator #1: Vorzeichen
SGNFLG	0067	103	Zeiger: Polynomauswertung
BITS	0068	104	Gleitpunktakkumulator #1: Überlauf
ARGEXP	0069	105	Gleitpunktakkumulator #2: Exponent

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
ARGHO	006A–006D	106–109	Gleitpunktakkumulator #2: Mantisse
ARGSGN	006E	110	Gleitpunktakkumulator #2: Vorzeichen
ARISGN	006F	111	Ergebnis des Vorzeichenvergleichs: Akku #1 Akku #2
FACOV	0070	112	Gleitpunktakkumulator #1: Niederwertige Stelle (Rundung)
FBUFPT	0071–0072	113–114	Zeiger: Kassettenpuffer
CHRGET	0073–008A	115–138	Unterroutine: Nächstes Byte vom BASIC-Text lesen
CHRGOT	0079	121	Erneutes Lesen des gleichen Text-Bytes
TXTPTR	007A–007B	122–123	Zeiger: Derzeitiges Byte des BASIC-Textes
RNDX	008B–008F	139–143	Eingangswert der RND-Funktion
STATUS	0090	144	KERNAL-Ein-/Ausgabestatuswort: ST
STKEY	0091	145	Flag: STOP-Taste/RVS-Taste
SVXT	0092	146	Zeit-Konstante für Kassette
VERCK	0093	147	Flag: 0 = LOAD, 1 = VERIFY
C3PO	0094	148	Flag: serieller Bus – Zeichen im Puffer
BSOUR	0095	149	Zeichen im Puffer für seriellen Bus
SYNO	0096	150	Kassetten SYNC.-Nr. (EOT von Kassette empfangen)
	0097	151	Temporäre Datenadresse
LDTND	0098	152	Anzahl der offenen Dateien/ Dateitabellen-Index
DFLTN	0099	153	Standard-Eingabegerät (0)
DFLTO	009A	154	Standard-Ausgabegerät (CMD) (3)
PRTY	009B	155	Paritätsbyte vom Band
DPSW	009C	156	Flag: Byte empfangen

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
MSGFLG	009D	157	Flag: $80 = Direktmodus, $00 = Programm
PTR1	009E	158	Bandfehler/Zeichenpuffer
PTR2	009F	159	Bandfehler korrigiert
TIME	00A0–00A2	160–162	Echtzeituhr (ca.) 1/60 s
	00A3–00A4	163–164	Temporärer Datenbereich
CNTDN	00A5	165	Kassetten Sync.: Abwärtszählung beim Schreiben
BUFPNT	00A6	166	Zeiger: Kassettenpuffer
INBIT	00A7	167	RS-232-Eingabebits/Kassette temp.
BITCI	00A8	168	RS-232-Eingabebit-Zählung/Kassette temp.
RINONE	00A9	169	RS-232 Flag: Startbitüberprüfung
RIDATA	00AA	170	RS-232-Eingabebyte-Puffer/Kassette temp.
RIPRTY	00AB	171	RS-232-Eingabeparität/Kassette, Zählung
SAL	00AC–00AD	172–173	Zeiger: Kassettenpuffer/Bildschirm scrollen
EAL	00AE–00AF	174–175	Kassettenende/Programmende
CMPO	00B0–00B1	176–177	Kassetten-Zeit-Konstante
TAPE1	00B2–00B3	178–179	Zeiger: Anfang des Kassettenpuffers
BITTS	00B4	180	RS-232 nächstes Bit zum Scrollen/Kassette temp.
NXTBIT	00B5	181	RS-232 Nächstes zu übertragendes Bit/Kassettenkennzeichen EOT
RODATA	00B6	182	RS-232 Bytepuffer
FNLEN	00B7	183	Länge der aktuellen Dateinamen
LA	00B8	184	Logische Dateinummer
SA	00B9	185	Aktuelle Sekundäradresse
FA	00BA	186	Aktuelle Gerätenummer
FNADR	00BB–00BC	187–188	Zeiger: Aktueller Dateiname

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
ROPRTY	00BD	189	RS-232 Parität/Kassette, temp.
FSBLK	00BE	190	Anzahl der zum Lesen/ Schreiben verbleibenden Blocks
MYCH	00BF	191	Serieller Puffer
CAS1	00C0	192	Kassettenmotor-Flag
STAL	00C1–00C2	193–194	Ein-/Ausgabestartadresse
MEMUSS	00C3–00C4	195–196	Zeiger auf Vektoradressen des KERNAL
LSTX	00C5	197	Derzeitig gedrückte Taste: CHR$(n); 0 = Keine Taste
NDX	00C6	198	Anzahl der Zeichen im Tastaturpuffer (Warteschlange)
RVS	00C7	199	Flag: Ausdruck negativer Zeichen – 1 = ja, 0 = nein
INDX	00C8	200	Zeiger: Ende der logischen Zeile für Eingabe
LXSP	00C9–00CA	201–202	Cursor X/Y-Position für Eingabe
SFDX	00CB	203	Flag: Gedrückte Taste
BLNSW	00CC	204	Cursor an/aus: (0 = blinkender Cursor)
BLNCT	00CD	205	Zähler für blinkenden Cursor
GDBLN	00CE	206	Zeichen für Cursorposition
BLNON	00CF	207	Flag: Cursor in Blinkphase
CRSW	00D0	208	Flag: INPUT oder GET über Tastatur
PNT	00D1–00D2	209–210	Zeiger: Derzeitige Bildschirmzeile
PNTR	00D3	211	Cursorspalte in derzeitiger Zeile
QTSW	00D4	212	Flag: Editor im Anführungszeichen-Modus, $00 = NEIN
LNMX	00D5	213	Physische Bildschirmzeilenlänge
TBLX	00D6	214	Zeile, in der sich Cursor befindet
	00D7	215	Temporärer Datenbereich
INSRT	00D8	216	Flag: Einfügemodus, >0 = Anzahl der Einfügungen

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
LDTB1	00D9–00F2	217–242	Bildschirmzeilen-Verknüpfungs-tabelle/Editor temp.
USER	00F3–00F4	243–244	Zeiger: Derzeitiger Farb-RAM des Bildschirms
KEYTAB	00F5–00F6	245–246	Vektor: Tastatur Decodiertabelle
RIBUF	00F7–00F8	247–248	RS-232-Eingabepuffer-Zeiger
ROBUF	00F9–00FA	249–250	RS-232-Ausgabepuffer-Zeiger
FREKZP	00FB–00FE	251–254	Freier Platz in der Zero-Page für Betriebssystem
BASZPT	00FF	255	Temp. BASIC-Datenbereich
	0100–01FF	256–511	Stapelspeicher des Mikro-prozessors
	0100–010A	256–266	Arbeitsbereich Umwandlung Gleitpunkt in ASCII
BAD	0100–013E	256–318	Bandfehler
BUF	0200–0258	512–600	System-Eingabepuffer
LAT	0259–0262	601–610	KERNAL-Tabelle: Aktive logische Dateinummern
FAT	0263–026C	611–620	KERNAL-Tabelle: Geräte-Nr. für jede Datei
SAT	026D–0276	621–630	KERNAL-Tabelle: Sekundär-adresse jeder Datei
KEYD	0277–0280	631–640	Tastaturpuffer (Warteschlange) (FIFO)
MEMSTR	0281–0282	641–642	Zeiger: Startadresse des RAM für Betriebssystem
MEMSIZ	0283–0284	643–644	Zeiger: Ende des RAM für Betriebssystem
TIMOUT	0285	645	Flag: Zeitüberschreitung auf IEEE-Bus
COLOR	0286	646	Derzeitiger Zeichenfarbcode
GDCOL	0287	647	Hintergrundfarbe unter Cursor
HIBASE	0288	648	Bildschirmspeicher-Anfang (Page)
XMAX	0289	649	Größe des Tastaturpuffers
RPTFLG	028A	650	Flag: Tastenwiederholung, $80 = Wiederholen

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
KOUNT	028B	651	Zählgeschwindigkeit für Wiederholen
DELAY	028C	652	Zähler für Wiederholungs-verzögerung
SHFLAG	028D	653	Flag: Taste SHIFT/Taste CTRL/ C = Taste
LSTSHF	028E	654	Letztes SHIFT-Muster der Tastatur
KEYLOG	028F–0290	655–656	Zeiger auf Tastatur-Decodier-tabelle
MODE	0291	657	Flag: $80 = SHIFT unwirksam, $00 = wirksam
AUTODN	0292	658	Flag: Automatisches Scrollen (abwärts), 0 = EIN; #0 = AUS
M51CTR	0293	659	RS-232: 6551 Kontrollregister
M51CDR	0294	660	RS-232: 6551 Befehlsregister
M51AJB	0295–0296	661–662	RS-232 nicht Standard (Bit-Zeit)
RSSTAT	0297	663	RS-232: 6551 Statusregister
BITNUM	0298	664	RS-232 Anzahl der noch zu übertragenden Bits
BAUDOF	0299–029A	665–666	RS-232 Baud-Rate: Full Bit Time (µs)
RIDBE	029B	667	RS-232 Eingabepuffer-Ende
RIDBS	029C	668	RS-232 Eingabepuffer-Anfang (Page)
RODBS	029D	669	RS-232 Ausgabepuffer-Anfang (Page)
RODBE	029E	670	RS-232 Ausgabepuffer-Ende
IRQTMP	029F–02A0	671–672	Enthält IRQ-Vektor während Kassetten-Ein-/Ausgabe
ENABL	02A1	673	RS-232
	02A2	674	Temp. Speicherung für Lesen von Kassette
	02A3	675	Temp Storage For Cassette Read
	02A4	676	Temp D1IRQ Indicator For

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
	02A5	677	Cassette Read
	02A6	678	Temp For Line Index
	02A7–02FF	679–767	PAL/NTSC Flag, 0 = NTSC, 1 = PAL
IERROR	0300–0301	768–769	Vektor: BASIC-Fehlermeldung anzeigen
IMAIN	0302–0303	770–771	Vektor: BASIC-Warmstart
ICRNCH	0304–0305	772–773	Vektor: BASIC-Text in Token umwandeln
IQPLOP	0306–0307	774–775	Vektor: BASIC-Text listen
IGONE	0308–0309	776–777	Vektor: BASIC-Befehl ausführen
IEVAL	030A–030B	778–779	Vektor: BASIC-Tokens-Auswertung
SAREG	030C	780	Speicher für 6502 .A-Register
SXREG	030D	781	Speicher für 6502 .X-Register
SYREG	030E	782	Speicher für 6502 .Y-Register
SPREG	030F	783	Speicher für SP6502 SP-Register
USRPOK	0310	784	USR-Sprung
USRADD	0311–0312	785–786	USR-Adresse niederwertiges Byte/höherwertiges Byte
	0313	787	Nicht benutzt
CINV	0314–0315	788–789	Vektor: Hardware Interrupt (IRQ) (EA31)
CBINV	0316–0317	790–791	Vektor: BRK-Interrupt (FE66)
NMINV	0318–0319	792–793	Vektor: Nicht maskierbarer Interrupt (NMI) (FE47)
IOPEN	031A–031B	794–795	KERNAL OPEN-Routine-Vektor
ICLOSE	031C–031D	796–797	KERNAL CLOSE-Routine-Vektor
ICHKIN	031E–031F	798–799	KERNAL CHKIN-Routine-Vektor
ICKOUT	0320–0321	800–801	KERNAL CHKOUT-Routine-Vektor
ICLRCH	0322–0323	802–803	KERNAL CLRCHN-Routine-Vektor
IBASIN	0324–0325	804–805	KERNAL CHRIN-Routine-Vektor

Unused

MARKE (LABEL)	ADRESSE (HEX)	ADRESSE (DEZ)	BESCHREIBUNG
IBSOUT	0326–0327	806–807	KERNAL CHROUT-Routine-Vektor
ISTOP	0328–0329	808–809	KERNAL STOP-Routine-Vektor
IGETIN	032A–032B	810–811	KERNAL GETIN-Routine-Vektor
ICLALL	032C–032D	812–813	KERNAL CLALL-Routine-Vektor
USRCMD	032E–032F	814–815	Benutzer-IRQ
ILOAD	0330–0331	816–817	KERNAL LOAD-Routine-Vektor
ISAVE	0332–0333	818–819	KERNAL SAVE-Routine-Vektor
	0334–033B	820–827	Nicht benutzt
TBUFFR	033C–03FB	828–1019	Kassettenpuffer
	03FC–03FF	1020–1023	Nicht benutzt
VICSCN	0400–07FF	1024–2047	1024 Byte Bildschirmspeicher-Bereich
	0400–07E7	1024–2023	Video-Matrix: 25 Zeilen x 40 Zeichen
	07F8–07FF	2040–2047	Sprite-Datenzeiger
	0800–9FFF	2048–40959	Normaler BASIC-Programm-bereich
	8000–9FFF	32768–40959	VSP-ROM-8192 Bytes (Optional)
	A000–BFFF	40960–49151	BASIC-ROM-8192 Bytes (oder 8K-RAM)
	C000-CFFF	49152–53247	RAM-4096 Bytes
	D000–DFFF	53248–57343	Ein-/Ausgabegerät und Farb-RAM oder Zeichengenerator-ROM oder RAM-4096 Bytes
	E000–FFFF	57344–65535	KERNAL ROM-8192 Bytes (oder 8K-RAM)

EIN-/AUSGABEANORDNUNG BEIM COMMODORE 64

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
0000	0	7–0	MOS 6510 Datenrichtungs-register (xx101111) Bit = 1: Ausgabe, Bit = 0: Eingabe X = Spielt keine Rolle
0001	1		MOS 6510 Mikroprozessor Ein-Chip Ein-/Ausgabeport
		0	/LORAM-Signal (0 = BASIC-ROM ausschalten)
		1	/HIRAM-Signal (0 = KERNAL-ROM ausschalten)
		2	/CHARAM-Signal (0 = Zeichen-ROM ausschalten)
		3	Kassettendaten-Ausgabeleitung
		4	Kassettenschalter 1 = Schalter geschlossen
		5	Kassetten-Motorsteuerung 0 = EIN, 1 = AUS
		6–7	Nicht belegt
D000–D02E	53248–54271		MOS 6566 VIDEO-INTERFACESTEUERUNG (VIC)
D000	53248		Sprite 0, Position X
D001	53249		Sprite 0, Position Y
D002	53250		Sprite 1, Position X
D003	53251		Sprite 1, Position Y
D004	53252		Sprite 2, Position X
D005	53253		Sprite 2, Position Y
D006	53254		Sprite 3, Position X
D007	53255		Sprite 3, Position Y
D008	53256		Sprite 4, Position X
D009	53257		Sprite 4, Position Y
D00A	53258		Sprite 5, Position X
D00B	53259		Sprite 5, Position Y
D00C	53260		Sprite 6, Position X

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
D00D	53261		Sprite 6, Position Y
D00E	53262		Sprite 7, Position X
D00F	53263		Sprite 7, Position Y
D010	53264		Sprites 0–7, Position X (msb der X-Koordinate)
D011	53265		VIC-Steuerregister
		7	Raster-Vergleich: (Bit 8) Siehe 53266
		6	Erweiterter Farbtext-Modus: 1= Einschalten
		5	Bit-Map-Modus: 1 = Einschalten
		4	Bildschirm löschen: 0 = Löschen
		3	Wahl von 24/25 Reihen Text-anzeige: 1 = 25 Reihen
		2–0	Rollen zur Y-Punktposition (0–7)
D012	53266		Leseraster/Schreibraster Wert für Vergleich IRQ
D013	53267		Lichtgriffel, Position X
D014	53268		Lichtgriffel, Position Y
D015	53269		Sprite-Anzeige: 1 = Einschalten
D016	53270		VIC-Steuerregister
		7–6	Nicht benutzt
		5	DIESES BIT STETS AUF 0 SETZEN!
		4	Mehrfarbenmodus: 1 = Einschalten (Text oder Bit-Mappe)
		3	Wahl von 38/40 Spalten Text-anzeige: 1 = 40 Zeichen
		2–0	Rollen zu Position X
D017	53271		Sprites 0–7 vergrößern 2 x vertikal (Y)
D018	53272		VIC-Speicher-Steuerregister
		7–4	Video-Matrix-Basisadresse

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
		3–1	Zeichengenerator-Basisadresse
D019	53273		VIC-Interrupt-Flag (Bit = 1: Einschalten des IRQ)
		7	Beliebige VIC-IRQ-Bedingung setzen
		3	IRQ-Flag wird durch Lichtgriffel getriggert
		2	IRQ-Flag für Sprite-Kollision
		1	IRQ-Flag für Sprite-/Hintergrundkollision
		0	IRQ-Flag für Rastervergleich
D01A	53274		IRQ-Maskenregister: 1 = Interrupt einschalten
D01B	53275		Sprite-/Hintergrund-Anzeigepriorität: 1 = Sprite
D01C	53276		Sprites 0–7 Mehrfarbenmodus gewählt: 1 = Mehrfarbenmodus
D01D	53277		Sprites 0–7, vergrößern 2 x horizontal (X)
D01E	53278		Sprite-Kollisionserkennung
D01F	53279		Sprite-/Hintergrundkollisions-Erkennung
D020	53280		Rahmenfarbe
D021	53281		Hintergrundfarbe 0
D022	53282		Hintergrundfarbe 1
D023	53283		Hintergrundfarbe 2
D024	53284		Hintergrundfarbe 3
D025	53285		Sprite-Mehrfarbenregister 0
D026	53286		Sprite-Mehrfarbenregister 1
D027	53287		Farbe von Sprite 0
D028	53288		Farbe von Sprite 1
D029	53289		Farbe von Sprite 2
D02A	53290		Farbe von Sprite 3
D02B	53291		Farbe von Sprite 4
D02C	53292		Farbe von Sprite 5
D02D	53293		Farbe von Sprite 6

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
D02E	53294		Farbe von Sprite 7
D400–D7FF	54272–55295		MOS 6581 SOUND-INTERFACE-DEVICE (SID)
D400	54272		Stimme 1: Frequenzsteuerung – Unteres Byte
D401	54273		Stimme 1: Frequenzsteuerung – Oberes Byte
D402	54274		Stimme 1: Pulswellen-Breite – Unteres Byte
D403	54275	7–4	Nicht benutzt
		3–0	Stimme 1: Pulswellen-Breite – Oberes Nybble
D404	54276		Stimme 1: Steuerregister
		7	Geräuschwellenform wählen, 1 = Ein
		6	Pulswellenform wählen, 1 = Ein
		5	Sägezahnwellenform wählen, 1 = Ein
		4	Dreieckswellenform wählen, 1 = Ein
		3	Testbit: 1 = Oszillator 1 abschalten
		2	Oszillator 1 mit Oszillator-ausgabe 3 ringmodulieren, 1 = Ein
		1	Oszillator 1 mit Oszillator 3 synchronisieren, 1 = Ein
		0	GATE-Bit: 1 = Beginn von ATTACK/DECAY/SUSTAIN, 0 = Start des RELEASE-Abschnitts
D405	54277		Hüllkurvengeber 1: Steuerung des ATTACK-/DECAY-Zyklus
		7–4	Wahl der ATTACK-Zyklusdauer: 0–15
		3–0	Wahl der DECAY-Zyklusdauer: 0–15

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
D406	54278		Hüllkurvengeber 1: Steuerung des SUSTAIN-/RELEASE-Zyklus
		7–4	Wahl des SUSTAIN-Pegels: 0–15
		3–0	Wahl der RELEASE-Dauer: 0–15
D407	54279		Stimme 2: Frequenzsteuerung – Unteres Byte
D408	54280		Stimme 2: Frequenzsteuerung – Oberes Byte
D409	54281		Stimme 2: Pulswellen-Breite – Unteres Byte
D40A	54282	7–4	Nicht benutzt
		3–0	Stimme 2: Pulswellen-Breite – Oberes Nybble
D40B	54283		Stimme 2: Steuerregister
		7	Wahl der Geräuschwellenform, 1 = Ein
		6	Wahl der Pulswellenform, 1 = Ein
		5	Wahl der Sägezahnwellenform, 1 = Ein
		4	Wahl der Dreieckswellenform, 1 = Ein
		3	Testbit: 1 = Oszillator 2 ausschalten
		2	Oszillator 2 mit Oszillator-ausgabe 1 ringmodulieren, 1 = Ein
		1	Oszillator 2 mit Oszillator-frequenz 1 synchronisieren, 1 = Ein
		0	GATE-Bit: 1 = Beginn von ATTACK/DECAY/SUSTAIN, 0 = Start des RELEASE-Abschnitts

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
D40C	54284		Hüllkurvengeber 2: Steuerung des ATTACK-/DECAY-Zyklus
		7–4	Wahl der ATTACK-Dauer: 0–15
		3–0	Wahl der DECAY-Dauer: 0–15
D40D	54285		Hüllkurvengeber 2: Steuerung SUSTAIN-/RELEASE-Zyklus
		7–4	Wahl des SUSTAIN-Pegels: 0–15
		3–0	Wahl der RELEASE-Dauer: 0–15
D40E	54286		Stimme 3: Frequenzsteuerung – Unteres Byte
D40F	54287		Stimme 3: Frequenzsteuerung – Oberes Byte
D410	54288		Stimme 3: Pulswellen-Breite – Unteres Byte
D411	54289	7–4	Nicht benutzt
		3–0	Stimme 3: Pulswellen-Breite – Oberes Nybble
D412	54290		Stimme 3: Steuerregister
		7	Wahl der Geräuschwellenform, 1 = Ein
		6	Wahl der Impulswellenform, 1 = Ein
		5	Wahl der Sägezahnwellenform, 1 = Ein
		4	Wahl der Dreieckswellenform, 1 = Ein
		3	Testbit: 1 = Oszillator 3 ausschalten
		2	Oszillator 3 mit Oszillator-ausgabe2 ringmodulieren, 1 = Ein
		1	Oszillator 3 mit Oszillator-frequenz 2 synchronisieren, 1 = Ein

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
		0	GATE-Bit: 1 = Beginn von ATTACK/DECAY/SUSTAIN, 0 = Start des RELEASE-Abschnitts
D413	54291		Hüllkurvengeber 3: Steuerung des ATTACK-/DECAY-Zyklus
		7–4	Wahl der ATTACK-Dauer: 0–15
		3–0	Wahl der DECAY-Dauer: 0–15
D414	54292		Hüllkurvengeber 3: Steuerung des SUSTAIN-/RELEASE-Zyklus
		7–4	Wahl des SUSTAIN-Pegels: 0–15
		3–0	Wahl der RELEASE-Dauer: 0–15
D415	54293		Filtergrenzfrequenz: Unteres Nybble (Bits 2–0)
D416	54294		Filtergrenzfrequenz: Oberes Byte
D417	54295		Filterresonanz-Steuerung/ Stimmeneingabe-Steuerung
		7–4	Wahl der Filterresonanz: 0–15
		3	Externe Filtereingabe: 1 = Ja, 0 = Nein
		2	Ausgabe von Stimme 3 filtern: 1 = Ja, 0 = Nein
		1	Ausgabe von Stimme 2 filtern: 1 = Ja, 0 = Nein
		0	Ausgabe von Stimme 1 filtern: 1 = Ja, 0 = Nein
D418	54296		Filtermodus und Lautstärke wählen
		7	Ausgabe von Stimme 3 abschalten: 1 = AUS, 0 = EIN
		6	Hochpaßfiltermodus wählen: 1 = Ein

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
		5	Wahl des Bandfiltermodus: 1 = Ein
		4	Wahl des Tiefpaßfiltermodus: 1 = Ein
		3–0	Wahl der Lautstärke: 0–15
D419	54297		Analog-/Digitalwandler: Drehregler 1 (0–255)
D41A	54298		Analog-/Digitalwandler: Drehregler 2 (0–255)
D41B	54299		Oszillator 3, Zufallszahlen-Generator
D41C	54230		Ausgabe von Hüllkurvengeber 3
D500-D7FF	54528–55295		SID-Images
D800–DBFF	55296–56319		Farb-RAM (Nybbles)
DC00–DCFF	56320–56575		MOS 6562 Komplexes Interfaceadapter (CIA) # 1
DC00	56320		Datenport A (Tastatur, Steuerknüppel, Drehregler, Lichtgriffel)
		7–0	Nummer der Tastaturspalte für Tastatur-Abfrage
		7–6	Drehregler Port A/B (01 = Port A, 10 = Port B)
		4	Steuerknüppel A Feuerknopf: 1 = Feuer
		3–2	Drehregler-Feuerknöpfe
		3–0	Steuerknüppel-Richtung (0–15)
DC01	56321		Daten-Port B (Tastatur, Steuerknüppel, Drehregler): Spielport 1
		7–0	Nummer der Tastatur-Reihe für Tastaturabfrage
		7	Timer B: Impulsausgabe
		6	Timer A: Impulsausgabe
		4	Steuerknüppel Feuerknopf 1: 1 = Feuer
		3–2	Drehregler-Feuerknopf

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
		3–0	Steuerknüppel-Richtung
DC02	56322		Datenrichtungsregister – Port A (56320)
DC03	56323		Datenrichtungsregister – Port A (56321)
DC04	56324		Timer A: Unteres Byte
DC05	56325		Timer A: Oberes Byte
DC06	56326		Timer B: Unteres Byte
DC07	56327		Timer B: Oberes Byte
DC08	56328		Tageszeituhr: 1/10 s
DC09	56329		Tageszeituhr: Sekunden
DC0A	56330		Tageszeituhr: Minuten
DC0B	56331		Tageszeituhr: Stunden + Flag AM/PM (Bit 7)
DC0C	56332		Serieller Bus Ein-/Ausgabe-datenpuffer
DC0D	56333		CIA-Interrupt-Steuerregister
		7	IRQ-Flag (1 = Auftreten von IRQ)/Löschflag setzen
		4	Flag 1 IRQ (Lesen von Kassette/serieller Bus SRQ-Eingabe)
		3	Serieller Bus (Interrupt)
		2	Tageszeituhr-Interrupt
		1	Timer B-Interrupt
		0	Timer A-Interrupt
DC0E	56334		CIA-Steuerregister A
		7	Tageszeituhr-Frequenz: 1 = 50 Hz, 0 = 60 Hz
		6	Serieller Bus Ein-/Ausgabe-modus: 1 = Ausgabe, 0 = Eingabe
		5	Timer A: 1 = CNT-Signale, 0 = System-Uhr 02
		4	Force Load Timer A: 1 = Ja
		3	Modus von Timer A: 1 = one-shot, 0 = kontinuierlich

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
		2	Ausgabemodus von Timer A zu PB6: 1 = Toggle, 0 = Impuls
		1	Ausgabe von Timer A an PB6: 1 = Ja, 0 = Nein
		0	Start/Stop von Timer A: 1 = Start, 0 = Stop
DC0F	56335		CIA-Steuerregister B
		7	Alarm/TOD-Uhr: 1 = Alarm, 0 = Takt
		6–5	Wahl des Modus von Timer B: 00 = Taktimpuls von System 02 zählen 01 = Positive CNT-Übergänge zählen 10 = Underflow-Impulse von Timer A zählen 11 = Underflows von Timer A zählen, wenn CNT positiv
		4–0	Entspricht CIA-Steuerregister A – für Timer B
DD00–DDFF	56576–56831		MOS 6526 Komplexes Interfaceadapter (CIA) # 2
DD00	56576		Datenport A (serieller Bus, RS-232, VIC-Speichersteuerung)
		7	Serieller Bus-Dateneingabe
		6	Serieller Bus-Impulseingabe
		5	Serieller Bus-Datenausgabe
		4	Serieller Bus-Impulsausgabe
		3	Serieller Bus-ATN-Signalausgabe
		2	RS-232-Datenausgabe (User-Port)
		1–0	VIC-Chip Bank-select (Standard = 11)

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
DD01	56577		Datenport B (User-Port, RS-232)
		7	RS-232 Datensatz bereit
		6	RS-232 Clear to send
		5	User
		4	RS-232 Carrier Detect
		3	RS-232 Ring Indicator
		2	RS-232 Daten-Terminal
		1	RS-232 Request to send
		0	RS-232 Received
DD02	56578		Datenrichtungs-Register – Port A
DD03	56579		Datenrichtungs-Register – Port B
DD04	56580		Timer A: Unteres Byte
DD05	56581		Timer A: Oberes Byte
DD06	56582		Timer B: Unteres Byte
DD07	56583		Timer B: Oberes Byte
DD08	56584		Tageszeituhr: 1/10 s
DD09	56585		Tageszeituhr: Sekunden
DD0A	56586		Tageszeituhr: Minuten
DD0B	56587		Tageszeituhr: Stunden + Flag AM/PM (Bit 7)
DD0C	56588		Serieller Bus Ein-/Ausgabe-datenpuffer
DD0D	56589		CIA-Interruptsteuerregister
		7	NMI-Flag (1 = Auftreten eines NMI)/ Löschflag setzen
		4	Flag 1 NMI (RS-232 Received Data Input)
		3	Interrupt-Serieller Bus
		1	Timer B-Interrupt
		0	Timer A-Interrupt
DD0E	56590		CIA-Steuerregister A
		7	TOD-F: 1 = 50 Hz, 0 = 60 Hz

HEXA-DEZIMAL	DEZIMAL	BITS	BESCHREIBUNG
		6	Serieller Bus Ein-/Ausgabe-modus: 1 = Ausgabe, 0 = Eingabe
		5	Timer A: 1 = CNT-Signale, 0 = System-Uhr 02
		4	Force Load Timer A: A = Ja
		3	Modus von Timer A: 1 = one-shot, 0 = kontinuierlich
		2	Ausgabemodus von Timer A zu PB6: 1 = Toggle, 0 = Impuls
		1	Ausgabe von Timer A an PB6: 1 = Ja, 0 = Nein
		0	Start/Stop von Timer A: 1 = Start, 0 = Stop
DD0F	56591		CIA-Steuerregister B
		7	Alarm/TOD-Clock: 1 = Alarm, 0 = Clock
		6–5	Wahl von Timermodus B: 00 = Impulse von System 02 zählen 01 = Positive CNT-Übergänge zählen 10 = Underflowimpulse von Timer A zählen 11 = Underflows von Timer A zählen, wenn CNT positiv
		4–0	Entspricht CIA-Steuerregister A – für Timer B
DE00–DEFF	56832–57087		Reserviert für künftige Ein-/Ausgabeerweiterungen
DF00–DFFF	57088–57343		Reserviert für künftige Ein-/Ausgabeerweiterungen

KAPITEL 6

EIN-/AUSGABE-ANLEITUNG

- Einführung
- Ausgabe auf den Bildschirm
- Ausgabe auf andere Geräte
- Spiele-Ports
- RS-232-Interface-Beschreibung
- User-Port
- Der serielle Bus
- Erweiterungsport
- Z-80-Mikroprozessor-Modul

EINFÜHRUNG

Computer zeichnen sich durch drei Hauptfunktionen aus: Sie können rechnen, Entscheidungen treffen und kommunizieren. Die Rechenfunktion läßt sich hierbei wahrscheinlich am einfachsten programmieren. Wir sind mit den meisten mathematischen Regeln vertraut. Entscheidungen zu treffen, ist auch nicht allzu schwierig, da es nur wenige logische Regeln gibt.
Die komplexeste Funktion ist die Kommunikation, da diese am wenigsten genauen Regeln unterliegt. Dies ist nicht etwa auf ein Versehen bei der Konstruktion des Computers zurückzuführen. Die Regeln sind vielmehr so flexibel, daß praktisch auf viele Arten kommuniziert werden kann. Die einzige Regel heißt: Die Informationen müssen stets so übertragen werden, daß sie der Empfänger auch verstehen kann.

AUSGABE AUF DEN BILDSCHIRM

Die einfachste Form der Ausgabe ist die BASIC-Anweisung PRINT. Durch PRINT wird als Ausgabegerät der Bildschirm benutzt. Das "Eingabegerät" sind Ihre Augen, da sie die Bildschirminformation lesen.
Bei der Anzeige auf dem Bildschirm besteht die Hauptaufgabe darin, die Informationen so zu formatieren, daß sie leicht lesbar sind. Spielen Sie hier ein bißchen den Graphiker oder Designer, benutzen Sie Farben, überdenken Sie die Anordnung der einzelnen Buchstaben, wählen Sie Groß- und Kleinbuchstaben oder auch Graphikzeichen, um die Information am besten darzustellen. Denken Sie daran: auch bei einem noch so raffinierten Programm müssen Sie doch in der Lage sein, das Ergebnis zu verstehen.
Die PRINT-Anweisung benutzt bestimmte Zeichen-Codes als "Befehle", mit denen der Cursor gesteuert wird. Über die Taste CRSR kann eigentlich nichts auf dem Bildschirm angezeigt werden. Sie ändert lediglich die Cursorposition. Über weitere Befehle wird die Farbe geändert, der Bildschirm gelöscht und Leerstellen eingefügt bzw. gelöscht. Die Taste RETURN hat die Zeichen-Code-Nr. 13 (CHR$). Diese Codes sind in einer Tabelle in Anhang C dargestellt.
Es gibt noch zwei weitere Funktionen in BASIC, die zusammen mit der PRINT-Anweisung verwendet werden. Durch TAB wird der Cursor in eine Position gebracht, die einen vorgegebenen Abstand vom linken Bildschirmrand hat, SPC bewegt den Cursor von der derzeitigen Position um eine gegebene Anzahl Leerstellen nach rechts.
Interpunktionszeichen in der PRINT-Anweisung trennen und formatieren die Information. Durch das Semikolon (;) werden zwei Ausdrücke ohne Leerzeichen voneinander getrennt. Ist das Semikolon das letzte Zeichen in einer Zeile, so bleibt der Cursor hinter dem zuletzt angezeigten Zeichen und geht nicht in die nächste Zeile

über. Es unterdrückt das RETURN, das normalerweise am Ende einer Zeile steht. Durch das Komma (,) werden die Daten in Spalten dargestellt. Der Bildschirm des COMMODORE 64 hat vier Spalten mit je 10 Zeichen. Wenn der Computer ein Komma "PRINTED", bewegt sich der Cursor nach rechts zum Anfang der nächsten Spalte. Nach der letzten Spalte der Zeile geht der Cursor in die nächste Zeile. Auch hier wird, wenn es sich um das letzte Zeichen in einer Zeile handelt, RETURN unterdrückt.
Anführungszeichen (" ") trennen Text von Variablen. Das erste Anführungszeichen in einer Zeile kennzeichnet den Anfang des Textbereichs und das nächste Anführungszeichen das entsprechende Ende. Übrigens ist am Zeilenende kein abschließendes Anführungszeichen erforderlich.
Durch den RETURN-Code (CHR$ (13)) wird der Cursor in die nächste *logische* Bildschirmzeile bewegt. Es muß nicht immer unbedingt die nächste Zeile sein. Wird über das Zeilenende hinaus geschrieben, so wird die Zeile mit der nächsten Zeile verbunden. Der Computer weiß, daß beide Zeilen in Wirklichkeit eine einzelne, lange Zeile sind. Diese Verbindungen werden in der *"line-link-Tabelle"* festgehalten (bezügl. Einzelheiten siehe Speicherbelegung).
Eine logische Zeile kann bis zu zwei Bildschirmzeilen lang sein, je nachdem, was eingegeben wurde. Die logische Zeile am Bildschirmanfang bestimmt, ob der Bildschirm um ein oder zwei Zeilen "gescrollt" wird.
Es gibt noch andere Möglichkeiten, um den Bildschirm als Ausgabegerät zu benutzen. In dem Kapitel über Graphiken wird beschrieben, wie man Graphiken erzeugen und über den Bildschirm bewegen kann. Im Abschnitt über den VIC-Chip wird beschrieben, wie man die Bildschirm- und Rahmenfarben und Größen ändern kann. Das Kapitel über den Sound-Synthesizer zeigt Ihnen schließlich, wie man mit dem TV-Lautsprecher Klangeffekte und Musik erzeugt.

AUSGABE AUF ANDERE GERÄTE

Oft ist es erforderlich, daß Ausgaben nicht auf den Bildschirm, sondern auf andere Geräte wie z. B. Kassettendecks, Drucker, Diskettenstationen oder Modems erfolgen. Über die BASIC-Anweisung OPEN wird ein "Kanal" für die Kommunikation mit diesen Geräten erstellt. Ist dieser Kanal geöffnet, dann werden über die Anweisung PRINT# Zeichen zu diesem Gerät übertragen.

BEISPIEL FÜR DIE ANWEISUNGEN OPEN UND PRINT#:

```
100 OPEN 4, 4: PRINT# 4, "WRITING ON PRINTER"
110 OPEN 3, 8, 3, "0:DIKS-FILE,S,W": PRINT# 3, "SEND TO DISK"
120 OPEN 1, 1, 1, "TAPE-FILE": PRINT# 1, "WRITE ON TAPE"
130 OPEN 2, 2, 0, CHR$(10): PRINT# 2, "SEND TO MODEM"
```

Die OPEN-Anweisung ist für die einzelnen Geräte leicht unterschiedlich. Die Parameter dieser Anweisung für das jeweilige Gerät werden in der nachstehenden Tabelle gegeben.

Tabelle für die OPEN-Anweisung:

FORMAT: OPEN Dateinummer, Geräteadresse, Sekundäradresse, String

GERÄT	GERÄTE-ADRESSE	SEKUNDÄRADRESSE	STRING
CASSETTE	1	0 = Eingabe 1 = Ausgabe 2 = Ausgabe mit EOT	Dateiname
MODEM	2	0	Steuerregister
SCREEN	3	0,1	
PRINTER	4 oder 5	0 = Großschrift/ Graphikzeichen 7 = Groß-/ Kleinschrift	Text wird angezeigt
DISK	8 bis 11	2–14 = Datenkanal 15 = Befehlskanal	Laufwerknummer, Dateiname, Dateityp, Befehl lesen/ schreiben

AUSGABE ZUM DRUCKER

Der Drucker ist eine ähnliche Ausgabevorrichtung wie der Bildschirm. Die Hauptaufgabe für Sie besteht hierbei darin, ein leicht lesbares Format zu erstellen. Hierbei stehen Ihnen negative (weiß auf schwarz darstellbare) Zeichen, Zeichen in doppelter Breite, Groß- und Kleinbuchstaben sowie programmierbare Graphikzeichen zur Verfügung.

Durch die OPEN-Anweisung wird der erforderliche Kanal zum Drucker geöffnet. Außerdem wird durch diese Anweisung angegeben, welcher Zeichensatz benutzt wird: entweder Großbuchstaben und Graphikzeichen oder Zeichen Groß- und Kleinbuchstaben.

BEISPIELE FÜR DIE OPEN-ANWEISUNG BEI DER AUSGABE AUF EINEN DRUCKER:

```
OPEN 1, 4: REM UPPER CASE/GRAPHICS
OPEN 1, 4, 7: REM UPPER AND LOWER CASE
```

Beim Ausdruck mit einem Zeichensatz können einzelne Zeilen mit dem anderen Zeichensatz erstellt werden. Wurden Großbuchstaben und Graphikzeichen benutzt, dann erfolgt die Umschaltung auf den anderen Zeichensatz, d. h. Groß- und Kleinbuchstaben durch (CHR$(17)). In der umgekehrten Richtung erfolgt die Umschaltung durch (CHR$(145)).
Andere Drucker-Sonderfunktionen werden über Zeichencodes gesteuert. Diese Codes werden genau wie andere Zeichen durch PRINT# übermittelt.

Tabelle der Drucker-Steuerzeichencodes:

CODE CHR$	ZWECK
10	Zeilenvorschub
13	RETURN ZEILENSCHALTUNG (automatischer Zeilenvorschub bei CBM-Druckern)
14	Beginn des Zeichenausdrucks in doppelter Breite
15	Ende des Zeichenausdrucks in doppelter Breite
18	Beginn des Ausdrucks negativ dargestellter Zeichen
146	Ende des Ausdrucks negativ dargestellter Zeichen
17	Umschalten auf Groß- und Kleinschrift
145	Umschalten auf Großschrift/Graphik
27	Bewegung zur angegebenen Punktposition
8	Beginn des Graphikmodus
26	Wiederholung der Graphikdaten

Bezüglich weiterer Einzelheiten über die Befehls-Codes siehe Bedienungsanleitung des jeweiligen COMMODORE-Druckers.

ARBEITEN MIT MAGNETBANDKASSETTEN

Kassetten haben eine fast unbegrenzte Daten-Speicherkapazität. Je länger hierbei das Kassettenband ist, desto mehr Informationen können gespeichert werden. Kassetten sind jedoch ziemlich langsam. Je mehr Daten abgespeichert sind, desto länger braucht man, um eine Information zu finden.

Dieser Zeitfaktor muß daher bei der Kassettenspeicherung vom Programmierer auf ein Mindestmaß beschränkt werden. Im allgemeinen wird die gesamte Kassetten-Datendatei in den RAM gelesen, dann verarbeitet und danach wieder zurück auf Kassette geschrieben. Auf diese Weise können die Daten sortiert, aufbereitet und überprüft werden. Allerdings wird hierdurch die Dateigröße durch die verfügbare RAM-Kapazität begrenzt.
Ist die Datendatei länger als der verfügbare RAM-Bereich, dann sollten Sie mit einer Disketten-Station arbeiten. Hierbei können Daten in jeder beliebigen Position auf der Diskette gelesen werden, ohne daß zuvor ein Lesen der übrigen Daten erforderlich ist. Alte Daten lassen sich ohne Beeinträchtigung der restlichen Datei überschreiben. Aus diesem Grund werden Disketten im Geschäftsbereich z. B. für Buchführungen und Adreßkarteien benutzt.
Durch die Anweisung PRINT# werden Daten genau wie durch PRINT formatiert. Auch die Interpunktion ist hierbei gleich. Denken Sie jedoch daran, daß Sie nun nicht mehr mit dem Bildschirm arbeiten. Beim Formatieren müssen Sie also stets an die Anweisung INPUT# denken.
Nehmen wir die Anweisung PRINT# 1, A$, B$, C$. Beim Arbeiten mit dem Bildschirm wird durch die Kommata zwischen den Variablen genug Platz geschaffen, um diese in Spalten mit je 10 Zeichen zu unterteilen. Bei einer Kassette werden 1 bis 10 Leerzeichen eingefügt, je nach Länge der Zeichenkette. Hierdurch wird Speicherkapazität verschwendet.
Wesentlich schlimmer wirkt sich dies jedoch aus, wenn die Zeichenketten durch die Anweisung INPUT# gelesen werden. Die Anweisung INPUT 1#, A$, B$, C$ findet keine Daten für B$ und C$. A$ enthält alle drei Variablen und die Leerzeichen dazwischen ab. Was passiert? Folgendermaßen sieht die Datei auf der Kassette aus:

```
A$="DOG":B$="CAT":C$="TREE"
PRINT# 1, A$, B$, C$
```

Ein geeignetes Begrenzungszeichen auf der Kassette wäre z. B. ein Komma (,) oder RETURN. Der RETURN-Code wird automatisch ans Ende einer PRINT-Anweisung oder von PRINT# gesetzt. Um diesen Code zwischen die einzelnen Punkte zu setzen, kann man z. B. nur ein Datum PRINT#-Anweisung benutzen. Besser ist jedoch, dem CHR$(13) oder dem Komma eine Variable zuzuordnen. Die Anweisung für letztere Möglichkeit ist R$=",":PRINT# 1, A$ R$ B$ R$ C$. Zwischen den Variablennamen dürfen keine Kommata oder andere Interpunktionszeichen verwendet werden; da der COMMODORE 64 sie auch so unterscheidet, kann auf diese Weise nur Kapazität verschwendet werden.

Eine richtige Kassetten-Datei sieht z. B. wie folgt aus:

```
1 2 3 4 5 6 7 8 9 10 11 12 13
DOG , CAT , T R E E RETURN
```

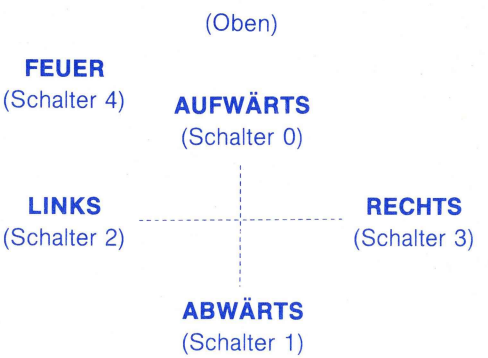
Durch die Anweisung GET# wird jeweils ein Zeichen der auf Kassette gespeicherten Daten gelesen. Jedes Zeichen, einschließlich RETURN-Code und anderen Interpunktionszeichen wird empfangen. Der Code CHR$(0) wird als Leerstring und nicht als Zeichenstring mit dem Code 0 empfangen.
Wird versucht, die ASC-Funktion bei einem Leerstring anzuwenden, dann erscheint die Fehlermeldung **ILLEGAL QUANTITY ERROR.**
Zur Überprüfung der Kassettendaten wird normalerweise die Zeile GET# 1, A$: A= ASC(A$) in Programmen benutzt. Zur Vermeidung von Fehlermeldungen sollte die Zeile wie folgt geändert werden: GET#1, A$: A= ASC(A$+ CHR$(0)). CHR$(0) am Ende macht Leerstrings "unschädlich", beeinflußt jedoch nicht die ASC-Funktion, wenn A$ andere Zeichen enthält.

DATENSPEICHERUNG AUF DISKETTEN

Auf Disketten sind drei verschiedene Arten der Datenspeicherung möglich. Sequentielle Dateien entsprechen denen auf Kassette, es können jedoch mehrere gleichzeitig benutzt werden. Relative Dateien ermöglichen ein Organisieren der Daten in Sätzen (Records) und ein Lesen und Ändern der einzelnen Sätze innerhalb der Datei. Bei Random-Dateien schließlich ist ein Arbeiten mit an beliebiger Diskettenstelle gespeicherten Daten möglich. Diese Daten sind in Abschnitten mit je 256 Bytes zusammengefaßt, die man Blöcke nennt.
Die Einschränkungen beim Arbeiten mit der Anweisung PRINT# sind in dem Abschnitt "Arbeiten mit Kassetten" beschrieben. Die gleichen Überlegungen treffen auch bei Disketten zu. Zum Abtrennen der einzelnen Daten wird RETURN oder ein Komma benötigt. Durch die Anweisung GET# wird CHR$(0) auch hier als leere Zeichenkette gelesen.
Relative und Random-Dateien arbeiten beide mit getrennten Daten und "Befehlskanälen". Die auf Diskette geschriebenen Daten gehen durch den Datenkanal und werden in den temporären Pufferspeicher des Disketten-RAMs geschrieben. Wenn ein Satz oder Block komplett ist, wird über den Befehlskanal ein Befehl übertragen, der angibt, wohin die Daten geschrieben werden sollen. Dann wird der gesamte Puffer geschrieben.
Bei Anwendungen, die die Verarbeitung großer Datenmengen erfordern, werden relative Diskettendateien verwendet. Dies erfordert am wenigsten Zeit und läßt dem Programmierer ein Höchstmaß an Flexibilität. Eine vollständige Programmieranleitung für Diskettendateien finden Sie im Handbuch der Diskettenstation.

SPIELE-PORTS

Der COMMODORE 64 hat zwei 9-Pin-Spiele-Ports, die die Verwendung von Steuerknüppeln, Drehreglern oder Lichtgriffeln ermöglichen. Jeder Port ist entweder für einen Steuerknüppel oder zwei Drehregler geeignet. Für spezielle Graphiksteuerungen usw. kann (nur in Port A) ein Lichtgriffel verwendet werden. In diesem Kapitel werden wir Ihnen Beispiele dafür zeigen, wie Sie sowohl Steuerknüppel als auch Drehregler über BASIC und Maschinensprache benutzen können.
Der Steuerknüppel wird an CIA #1 angeschlossen (MOS 6526). Dieser Ein-/Ausgabechip ist auch für die Feuerknöpfe an den Drehreglern und die Tastatur-Abfrage verantwortlich. Der 6526 CIA-Chip hat 16 Register in den Speicherplätzen 56320 bis einschließlich 56335 ($DC00 bis $DC0F). Die Daten von Port A finden Sie in Adresse 56320 (DC00) und von Port B in 56321 ($CD01).
Ein Steuerknüppel hat fünf unterschiedliche Schalter, von denen vier für die Richtung und einer als Feuerknopf benutzt wird. Die Steuerknüppel-Schalter sind wie folgt angeordnet:



Diese Schalter entsprechen den unteren 5 Bits des Inhalts der Adresse 56320 oder 56321. Ein Bit ist auf 1 gesetzt, wenn eine Richtung nicht gewählt oder der Feuerknopf nicht gedrückt wurde.
Wird der Feuerknopf gedrückt, dann wird das Bit (in diesem Fall Bit 4) auf 0 gesetzt. Um den Steuerknüppel von BASIC abzufragen, wird folgendes Unterprogramm benutzt:

```
10 FORK=0TO10:REM SET UP DIRECTION STRING
20 READDR$(K):NEXT
30 DATA"","N","S","","W","NW"
40 DATA"SW","","E","NE","SE"
50 PRINT"GOING...";
60 GOSUB100:REM READ THE JOYSTICK
65 IFDR$(JV)=""THEN80:REM CHECK IF A DIRECTION WAS
CHOSEN
70 PRINTDR$(JV);" ";:REM OUTPUT WHICH DIRECTION
80 IFFR=16THEN60:REM CHECK IF FIRE BUTTON WAS
PUSHED
90 PRINT"-----F-----I-----R-----E-----!!!":GOTO60
100 JV=PEEK(56320):REM GET JOYSTICK VALUE
110 FR=JVAND16:REM FORM FIRE BUTTON STATUS
120 JV=15-(JVAND15):REM FORM DIRECTION VALUE
130 RETURN
```

Anmerkung: Für den zweiten Steuerknüppel JV = PEEK (56321) setzen.

Die Werte für JV entsprechen diesen Richtungen:

JV ENTSPRICHT	RICHTUNG
0	KEINE
1	AUFWÄRTS
2	ABWÄRTS
3	–
4	LINKS
5	AUFWÄRTS & LINKS
6	ABWÄRTS & LINKS
7	–
8	RECHTS
9	AUFWÄRTS & RECHTS
10	ABWÄRTS & RECHTS

Folgendes kurzes Maschinen-Code-Programm erfüllt die gleiche Aufgabe:

```
1000 .PAGE (JOYSTICK.8/5)  JOYSTICK - BUTTON READ
ROUTINE
1010 ;
1020 ;AUTHOR - BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR    LDA $DC00      ;    (GET INPUT FROM PORT
A ONLY)
1080 DJRRB   LDY #0         ;THIS ROUTINE READS AND
DECODES THE
1090         LDX #0         ;JOYSTICK/FIREBUTTON
INPUT DATA IN
1100         LSR A          ;THE ACCUMULATOR. THIS
LEAST SIGNIFICANT
1110         BCS DJR0       ;5 BITS CONTAIN THE
SWITCH CLOSURE
1120         DEY            ;INFORMATION. IF A SWITCH
IS CLOSED THEN IT
1130 DJR0    LSR A          ;PRODUCES A ZERO BIT. IF
A SWITCH IS OPEN THEN
1140         BCS DJR1       ;IT PRODUCES A ONE BIT.
THE JOYSTICK DIR-
1150         INY            ;ECTIONS ARE RIGHT, LEFT,
FORWARD, BACKWARD
1160 DJR1    LSR A          ;BIT3=RIGHT, BIT2=LEFT,
BIT1=BACKWARD,
1170         BCS DJR2       ;BIT0=FORWARD AND
BIT4=FIRE BUTTON.
1180         DEX            ;AT RTS TIME DX AND DY
CONTAIN 2'S COMPLIMENT
1190 DJR2    LSR A          ;DIRECTION NUMBERS I.E.
$FF=-1, $00=0, $01=1.
1200         BCS DJR3       ;DX=1 (MOVE RIGHT), DX=-1
(MOVE LEFT),
1210         INX            ;DX=0 (NO X CHANGE).
DY=-1 (MOVE UP SCREEN),
1220 DJR3    LSR A          ;DY=1 (MOVE DOWN SCREEN),
DY=0 (NO Y CHANGE).
1230         STX DX         ;THE FORWARD JOYSTICK
POSITION CORRESPONDS
1240         STY DY         ;TO MOVE UP THE SCREEN
AND THE BACKWARD
1250         RTS            ;POSITION TO MOVE DOWN
SCREEN.
1260 ;
1270 ;AT RTS TIME THE CARRY FLAG CONTAINS THE FIRE
BUTTON STATE.
1280 ;IF C=1 THEN BUTTON NOT PRESSED. IF C=0 THEN
PRESSED.
1290 ;
1300 .END
```

DREHREGLER

Ein Paar Drehregler wird am Chip CIA #1 und SID (MOS 6581 Sound-Interface-Vorrichtung) über ein Spiele-Port angeschlossen. Der Drehregler-Wert wird über die SID-Register 54297 ($D419) und 54298 ($D41A) gelesen. DREHREGLER SIND NICHT ABSOLUT ZUVERLÄSSIG, WENN SIE NUR VON BASIC ABGEFRAGT WERDEN!!! Am besten benutzt man die Drehregler mit folgendem Maschinensprache-Programm . . . (SYS von BASIC, danach PEEK der von dem Unterprogramm benutzten Speicherplätze).

```
1000
;*************************************************************
1010 ;* FOUR PADDLE READ ROUTINE (CAN ALSO BE USED
FOR TWO)
1020
;*************************************************************
1030 ;AUTHOR - BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 PDLX *=*+2
1100 PDLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 PDLRD
1150     LDX #1                    ;FOR FOUR PADDLES
OR TWO ANALOG JOYSTICKS
1160 PDLRD0                        ;ENTRY POINT FOR
ONE PAIR (CONDITION X 1ST)
1170     SEI
1180     LDA CIDDRA                ;GET CURRENT VALUE
OF DDR
1190     STA BUFFER                ;SAVE IT AWAY
1200     LDA #$C0
1210     STA CIDDRA                ;SET PORT A FOR
INPUT
1220     LDA #$80
1230 PDLRD1
1240     STA PORTA                 ;ADDRESS A PAIR OF
PADDLES
1250     LDY #$80                  ;WAIT A WHILE
1260 PDLRD2
1270     NOP
1280     DEY
1290     BPL PDLRD2
1300     LDA SID+25                ;GET X VALUE
1310     STA PDLX,X
1320     LDA SID+26                ;GET Y VALUE
1330     STA PDLY,X
```

```
1340     LDA PORTA                    ;TIME TO READ
PADDLE FIRE BUTTONS
1350     ORA #$80                     ;MAKE IT THE SAME
AS OTHER PAIR
1360     STA BTNA                     ;BIT 2 IS PDL X,
BIT 3 IS PDL Y
1370     LDA #$40
1380     DEX                          ;ALL PAIRS DONE?
1390     BPL PDLRD1                   ;NO
1400     LDA BUFFER
1410     STA CIDDRA                   ;RESTORE PREVIOUS
VALUE OF DDR
1420     LDA PORTA+1                  ;FOR 2ND PAIR -
1430     STA BTNB                     ;BIT 2 IS PDL X,
BIT 3 IS PDL Y
1440     CLI
1450     RTS
1460 .END
```

Die Drehregler können notfalls durch folgendes BASIC-Programm abgefragt werden:

```
10 C=12*4096:REM SET PADDLE ROUTINE START
11 REM POKE IN THE PADDLE READING ROUTINE
15 FORI=0TO63:READA:POKEC+I,A:NEXT
20 SYSC:REM CALL THE PADDLE ROUTINE
30 P1=PEEK(C+257):REM SET PADDLE ONE VALUE
40 P2=PEEK(C+258):REM  "     "     TWO    "
50 P3=PEEK(C+259):REM  "     "    THREE   "
60 P4=PEEK(C+260):REM  "     "     FOUR   "
61 REM READ FIRE BUTTON STATUS
62 S1=PEEK(C+261):S2=PEEK(C+262)
70 PRINTP1,P2,P3,P4:REM PRINT PADDLE VALUES
72 REM PRINT FIRE BUTTON STATUS
75 PRINT:PRINT"FIRE A ";S1,"FIRE B ";S2
80 FORW=1TO50:NEXT:REM WAIT A WHILE
```

SHIFT CLR/HOME

```
90 PRINT"□":PRINT:GOTO 20:REM CLEAR SCREEN AND DO
AGAIN
95 REM DATA FOR MACHINE CODE ROUTINE
100 DATA162,1,120,173,2,220,141,0,193,169,192,141,
2,220,169
110 DATA128,141,0,220,160,128,234,136,16,252,173,
25,212,157
120 DATA1,193,173,26,212,157,3,193,173,0,220,9,128,
141,5,193
130 DATA169,64,202,16,222,173,0,193,141,2,220,173,
1,220,141
140 DATA6,193,88,96
```

LICHTGRIFFEL

Der Lichtgriffeleingang LEGT die derzeitige Bildschirmposition in einem Registerpaar (LPX, LPY) ab. Das X-Positionsregister 19 (#13) enthält die 8 MSB der X-Position zum Übergangszeitpunkt. Da die X-Position durch einen 512-Statuszähler (9 Bits) definiert wird, ist eine Auflösung von zwei Punkten in horizontaler Richtung möglich. Ähnlich wird die Y-Position in Register 20 ($14) abgelegt. 8 Bits erlauben hier jedoch eine Einzel-Rasterauflösung innerhalb der sichtbaren Anzeige. Der Lichtgriffel kann nur einmal pro Einzel-Bild ausgelassen werden, alle nachfolgenden Abfragen innerhalb des gleichen Bildes bleiben unberücksichtigt.

RS-232-INTERFACE-BESCHREIBUNG

ALLGEMEINER ÜBERBLICK

Der COMMODORE 64 hat ein eingebautes RS-232-Interface zum Anschluß an ein beliebiges RS-232-Modem, einen Drucker oder eine andere Vorrichtung. Um ein solches Gerät an den COMMODORE 64 anzuschließen, brauchen Sie lediglich ein entsprechendes Kabel und ein klein wenig Programmierung.

RS-232 vom COMMODORE 64 ist entsprechend dem Standard-RS-232-Format eingerichtet. Die Spannungen haben jedoch TTL-Pegel (0 bis 5V) und liegen nicht, wie normalerweise, im −12V bis +12V-Bereich. Im Bedarfsfall muß ein Interface zwischen dem COMMODORE 64 und dem RS-232-Gerät die Spannungen umwandeln. Dies leistet z. B. das COMMODORE-RS-232-Interface-Modul.

Auf die RS-232-Interface-Software kann über BASIC oder den KERNAL (für Maschinensprache-Programmierung) zugegriffen werden.

RS-232 arbeitet mit normalen BASIC-Befehlen: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT# und die reservierte Variable ST. INPUT# und GET# lesen Daten vom Empfangspuffer, PRINT# und CMD geben die Daten hingegen in den Übertragungspuffer.

Die Anwendung dieser Befehle wird später in diesem Kapitel noch anhand von Beispielen beschrieben.

Die RS-232-KERNAL-Routinen werden durch die 6526 CIA #2-Timer und Interrupts gesteuert. Der Chip 6526 erzeugt NMI-Anforderungen (nicht maskierbarer Interrupt) für die RS-232-Verarbeitung.

Hierdurch wird eine RS-232-Hintergrundverarbeitung während BASIC und Maschinensprache-Programmen möglich. Routinen des KERNAL, der Kassette und des seriellen Busses sind so abgesichert, daß keine Störungen während der Datenspeicherung oder Übertragung durch NMIs möglich sind, die durch die RS-232-Routine erzeugt wurden. Wenn der Kassettenport oder der serielle Bus aktiv sind, ist kein Datenempfang durch RS-232-Vorrichtungen möglich.
Das RS-232-Interface vom COMMODORE 64 hat zwei Puffer, damit beim Empfang oder der Übertragung von RS-232-Informationen keine Daten verlorengehen.
Die RS-232-KERNAL-Puffer bestehen aus zwei FIFO-Puffern (first in/first out-Puffer), die jeweils 256 Bytes lang sind und sich am oberen Speicherende befinden. Durch das Öffnen eines RS-232-Kanals werden automatisch 512 Bytes des Speichers für diese Puffer reserviert. Sollte nicht genug Platz hinter dem Ende des BASIC-Programms vorhanden sein, wird keine Fehlermeldung angezeigt und das Programmende daher zerstört. **SEIEN SIE DAHER VORSICHTIG!**
Diese Puffer werden automatisch durch die CLOSE-Befehle gelöscht.

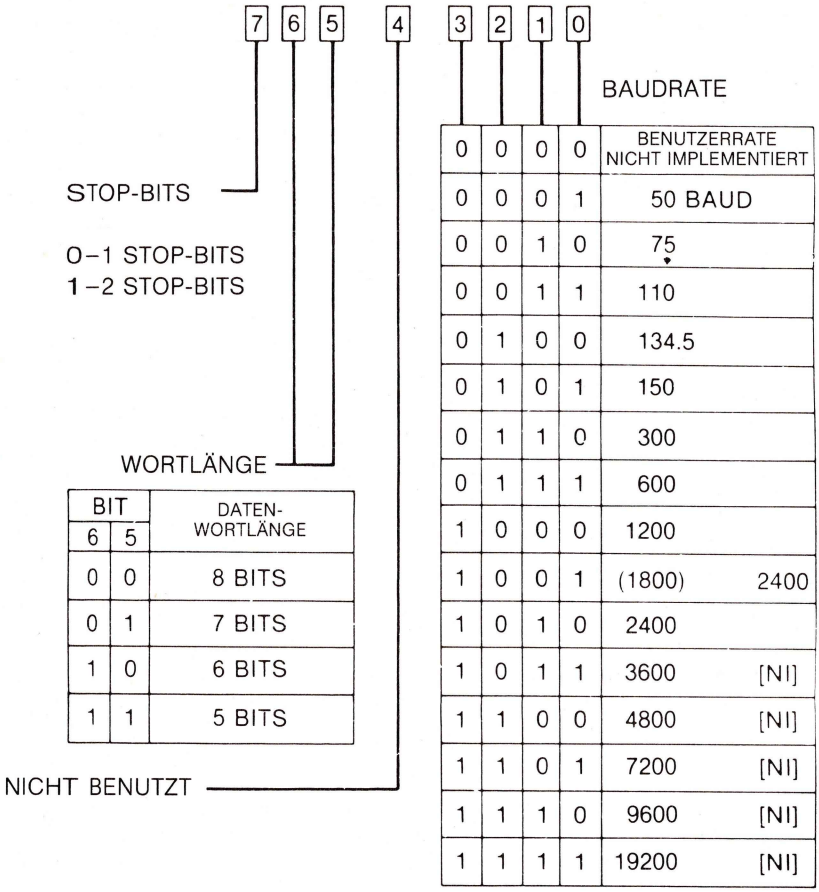
ÖFFNEN EINES RS-232-KANALS

Es darf nur stets ein RS-232-Kanal offen sein; durch eine zweite OPEN-Anweisung werden die Puffer-Zeiger rückgestellt. Alle Zeichen, die entweder im Ausgangs- oder im Eingangspuffer sind, werden gelöscht.
Das Dateiname-Feld kann maximal vier Zeichen enthalten. Die ersten beiden sind Steuer- und Befehlsregisterzeichen, die nächsten zwei sind für künftige Systemoptionen reserviert. Auf diese Weise kann man Baud-Rate, Parität und andere Optionen wählen.
Die Eingabe in das Steuerregister wird nicht auf eine nicht-implementierte Baud-Rate überprüft. Durch eine unzulässige Eingabe ergibt sich für die Systemausgabe eine extrem lange Rate (unter 50 Baud).

BASIC-SYNTAX:

OPEN lfn,2,0,"<Steuerregister><Befehlsregister><Option, niedrige Baud-Rate><Option, hohe Baud-Rate>"

Lfn – Für die logische Dateinummer (lfn) kann eine beliebige Zahl zwischen 1 und 255 gewählt werden. Beachten Sie jedoch, daß bei einer logischen Dateinummer über 127 nach einer Zeilenschaltung auch ein Zeilenvorschub erfolgt.



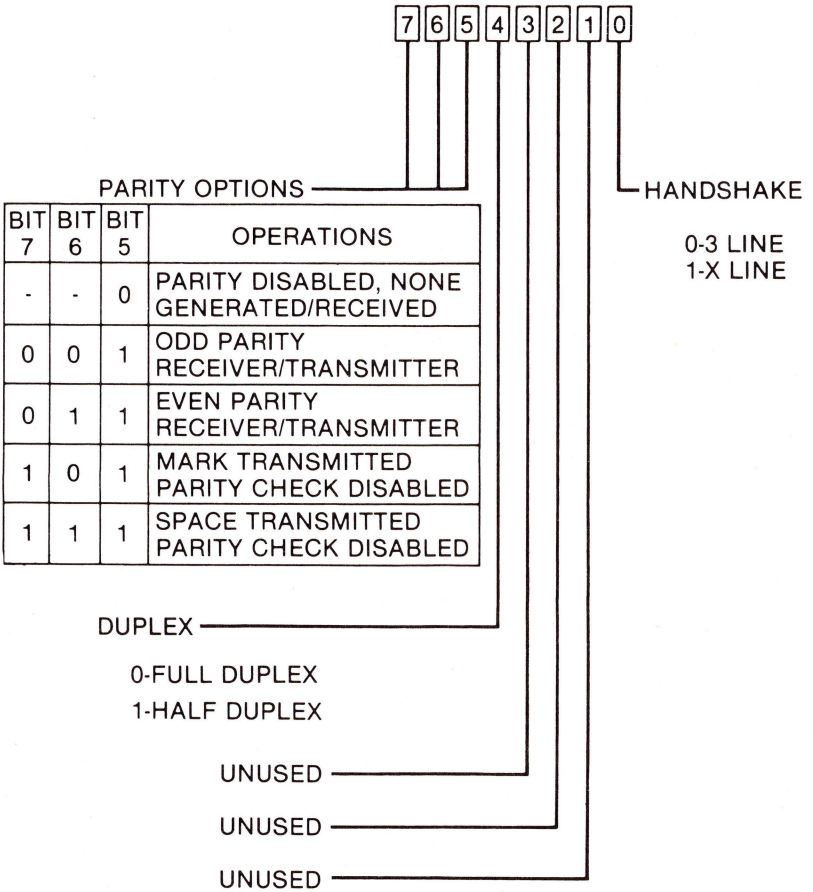
BIT 6	BIT 5	DATEN-WORTLÄNGE
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

3	2	1	0	BENUTZERRATE NICHT IMPLEMENTIERT
0	0	0	0	
0	0	0	1	50 BAUD
0	0	1	0	75
0	0	1	1	110
0	1	0	0	134.5
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	(1800) 2400
1	0	1	0	2400
1	0	1	1	3600 [NI]
1	1	0	0	4800 [NI]
1	1	0	1	7200 [NI]
1	1	1	0	9600 [NI]
1	1	1	1	19200 [NI]

Abb. 6.1. Steuerregisterbelegung

<Steuerregister> – Dies ist ein Ein-Byte-Zeichen (siehe Abb. 6.1. Steuerregisterbelegung), durch das die Eingabe der Baud-Rate festgelegt wird. Sind die unteren vier Bits der Baud-Rate null (0), dann werden durch <Option, Baud-low><Option, Baud-high> folgende Raten angegeben:

<Option, Baud-low>=<Systemfrequenz/Rate/2–100–
<Option, Baud-high>*256
<Option, Baud-high>=INT((Systemfrequenz/Rate/2–100)/256



BIT 7	BIT 6	BIT 5	OPERATIONS
-	-	0	PARITY DISABLED, NONE GENERATED/RECEIVED
0	0	1	ODD PARITY RECEIVER/TRANSMITTER
0	1	1	EVEN PARITY RECEIVER/TRANSMITTER
1	0	1	MARK TRANSMITTED PARITY CHECK DISABLED
1	1	1	SPACE TRANSMITTED PARITY CHECK DISABLED

Abb. 6.2. Befehlsregisterbelegung

Obige Formeln basieren auf folgender Grundlage:

Systemfrequenz = 1,02273E6 NTSC (North American TV standard)
= 0,98525E6 PAL (Britische und Europäische TV-Norm)

<Befehlsregister> – Dies ist ein Ein-Byte-Zeichen (siehe Abb. 6.2., Befehlsregisterbelegung), das weitere Terminal-Parameter festlegt. Dieses Zeichen ist nicht erforderlich.

KERNALEINGABE:

OPEN ($FFC0) (Siehe KERNAL-Spezifikation bezüglich weiterer Einzelheiten über Eingabebedingungen und -anweisungen.)

Wichtiger Hinweis: In einem Basic-Programm muß der OPEN-Befehl RS-232 vor der Erstellung von Variablen oder Feldern ausgeführt werden, da nach dem Öffnen eines RS-232-Kanals automatisch ein CLR ausgeführt wird (dies liegt an der Reservierung von 512 Bytes am oberen Speicherende). Denken Sie auch daran, daß das Programm zerstört wird, wenn diese 512 Bytes bei der OPEN-Anweisung nicht zur Verfügung stehen.

LESEN DER DATEN VON EINEM RS-232-KANAL

Beim Lesen von Daten von diesem Kanal speichert der Empfangspuffer des COMMODORE 64 255 Zeichen, ehe es zu einem Puffer-Überlauf kommt. Dies wird im RS-232-Statuswort (ST in BASIC oder RSSTAT in Maschinensprache) angezeigt. Bei einem Überlauf gehen alles überzähligen Zeichen verloren. Der Puffer sollte daher stets so frei wie möglich gehalten werden.
Ist ein schneller Empfang von RS-232-Daten gewünscht (dies ist mit BASIC nur begrenzt möglich, besonders bei der "Garbage collection" kann es zu einem Überlauf des Eingangspuffers kommen), dann müssen hierzu Maschinensprache-Routinen benutzt werden.

BASIC-SYNTAX:

Empfohlen: GET#lfn, <String>
NICHT empfohlen: INPUT#lfn, <Variablenliste>

KERNAL-EINGABEN:

CHKIN ($FFC6) – Bezüglich weiterer Einzelheiten über Ein- und Ausgabebedingungen siehe Speicherbelegung.
GETIN ($FFE4) – Bezüglich weiterer Einzelheiten über Ein- und Ausgabebedingung siehe Speicherbelegung.
CHRIN ($FFCF) – Bezüglich weiterer Einzelheiten über Ein- und Ausgabebedingung siehe Speicherbelegung.

Anmerkungen: Ist ein Wort kürzer als 8 Bit, dann wird allen nicht benutzten Bits der Wert 0 zugeordnet. Findet GET# keine Daten im Puffer, dann wird das Zeichen "" (eine Null) ausgegeben. Wird INPUT# benutzt, dann wartet das System so lange, bis ein Nicht-Nullzeichen und danach ein CR empfangen wird. Aus diesem Grund werden die Routinen INPUT# und CHRIN NICHT empfohlen. Routine CHKIN verwaltet das X-Draht-Handshake, das dem EIA-Standard (August 1979) für RS-232-C-Interfaces entspricht. (Die Leitungen für RTS, CTS und DCD sind beim COMMODORE 64 wie bei einem Datenterminal angeordnet.)

ÜBERTRAGEN VON DATEN ÜBER EINEN RS-232-KANAL

Beim Übertragen von Daten kann der Ausgabepuffer maximal 255 Zeichen speichern. Das System wartet in der Routine CHROUT so lange, bis die Übertragung ermöglicht oder die Tasten RUN/STOP und RESTORE gedrückt werden, um das System über einen WARMSTART zurückzusetzen.

BASIC-SYNTAX:

CMD lfn – entspricht den BASIC-Spezifikationen
PRINT#lfn, <Variablenliste>

KERNAL-EINGABEN:

CHKOUT ($FFC9) – Bezüglich weiterer Einzelheiten über Ein- und Ausgabebedingungen siehe Speicherbelegung.
CHROUT ($FFD2) – Bezüglich weiterer Einzelheiten über Eingabebedingungen siehe Speicherbelegung.

Wichtige Hinweise: Der Ausgabekanal enthält keine Verzögerung für CR. Dies bedeutet, daß ein normaler RS-232-Drucker nicht richtig ausdrucken kann, wenn nicht eine Verzögerung (die den COMMODORE 64 warten läßt) oder ein interner Puffer implementiert sind. Dies kann leicht per Programm erfolgen. Bei der Implementierung eines CTS-Kontakts (X-Draht-Handshake) wird der Puffer des COMMODORE 64 gefüllt und stoppt dann weitere Ausgaben, bis die Übertragung durch das RS-232-Gerät ermöglicht wird. X-Draht-Handshake ist eine Handshake-Routine, die für das Übertragen und Empfangen von Daten mehrere Leitungen benutzt.
Die Routine CHKOUT regelt das X-Draht-Handshake, das dem EIA-Standard (August 1979) für RS-232-C-Interfaces entspricht. Die Leitungen RTS, CTS und DCD sind beim COMMODORE wie bei einem Datenterminal implementiert.

SCHLIESSEN EINES RS-232-DATENKANALS

Nach dem Schließen einer RS-232-Datei werden alle Daten im Puffer gelöscht (unabhängig davon, ob sie übertragen oder ausgedruckt wurden), der gesamte RS-232-Übertragungs- und Empfangsbetrieb gestoppt, RTS und Übertragungsdatenleitungen (S_{out}) auf H-Pegel gesetzt und beide RS-232-Puffer gelöscht.

BASIC-SYNTAX:

CLOSE lfn

KERNAL-EINGABE:

CLOSE ($FFC3) – bezüglich weiterer Einzelheiten über Ein- und Ausgabebedingung siehe Speicherbelegung.

Anmerkung: Vor dem Schließen eines Kanals stets sicherstellen, daß alle Daten übertragen wurden. Hierzu gilt folgende BASIC-Anweisung:

```
100 SS=ST: IF(SS=0 OR SS=8) THEN 100
110 CLOSE lfn
```

Tabelle 6.1. User-Port-Lines

(6526 DEVICE #2 Loc. $DD00–$DD0F)						
PIN ID	6526 ID	DESCRIPTION	EIA	ABV	IN/ OUT	MODES
C	PB0	RECEIVED DATA	(BB)	S_{in}	IN	1 2
D	PB1	REQUEST TO SEND	(CA)	RTS	OUT	1*2
E	PB2	DATA TERMINAL READY	(CD)	DTR	OUT	1*2
F	PB3	RING INDICATOR	(CE)	RI	IN	3
H	PB4	RECEIVED LINE SIGNAL	(CF)	DCD	IN	2
J	PB5	UNASSIGNED	( )	XXX	IN	3
K	PB6	CLEAR TO SEND	(CB)	CTS	IN	2
L	PB7	DATA SET READY	(CC)	DSR	IN	2
B	FLAG2	RECEIVED DATA	(BB)	S_{in}	IN	1 2
M	PA2	TRANSMITTED DATA	(BA)	S_{out}	OUT	1 2
A	GND	PROTECTIVE GROUND	(AA)	GND		1 2
N	GND	SIGNAL GROUND	(AB)	GND		1 2 3

Erklärung:
1) 3-LEITUNGS-INTERFACE (S_{in}, S_{out}, GND)
2) X-LEITUNGS-INTERFACE
3) NUR FÜR BENUTZER (nicht benutzt/nicht implementiert)
* Diese Leitungen werden während des 3-Drahtmodus auf "High" gelegt.

```
[7] [6] [5] [4] [3] [2] [1] [0] (Machine Lang.—RSSTAT
 :   :   :   :   :   :   :   :_PARITY ERROR BIT
 :   :   :   :   :   :   :_____FRAMING ERROR BIT
 :   :   :   :   :   :_________RECEIVER BUFFER OVERRUN BIT
 :   :   :   :   :_____________RECEIVER BUFFER—EMPTY
                               (USE TO TEST AFTER A GET#)
 :   :   :   :_________________CTS SIGNAL MISSING BIT
 :   :   :_____________________UNUSED BIT
 :   :_________________________DSR SIGNAL MISSING BIT
 :_____________________________BREAK DETECTED BIT
```

Abb. 6.3. RS-232-Statusregister

Anmerkungen: Ist Bit = 0, dann wurde kein Fehler erkannt.
Das RS-232-Statusregister kann von BASIC über die Variable ST gelesen werden.
Wird ST über BASIC oder die KERNAL-Routine READST gelesen, dann wird das RS-232-Statuswort beim Programmende gelöscht. Soll das Statuswort mehrfach benutzt werden, dann ist ST einer anderen Variable zuzuordnen. Z. B.:

```
SR=ST: REM ASSIGNS ST TO SR
```

Der RS-232-Status wird nur gelesen (und gelöscht), wenn der RS-232-Kanal die zuletzt benutzte externe Ein-/Ausgabe war.

BASIC-PROGRAMMBEISPIEL

```
10 REM THIS PROGRAM SENDS AND RECEIVES DATA
TO/FROM A SILENT 700
11 REM TERMINAL MODIFIED FOR PET ASCII
20 REM TI SILENT 700 SET-UP: 300 BAUD, 7-BIT ASCII,
 MARK PARITY,
21 REM FULL DUPLEX
30 REM SAME SET-UP AT COMPUTER USING 3-LINE
INTERFACE
100 OPEN 2,2,3,CHR$(6+32)+CHR$(32+128):REM OPEN
THE CHANNEL
110 GET#2,A$:REM TURN ON THE RECEIVER CHANNEL
(TOSS A NULL)
200 REM MAIN LOOP
210 GET B$:REM GET FROM COMPUTER KEYBOARD
220 IF B$<>"" THEN PRINT#2,B$;:REM IF A KEY
PRESSED, SEND TO TERMINAL
230 GET#2,C$:REM GET A KEY FROM THE TERMINAL
240 PRINT B$;C$;:REM PRINT ALL INPUTS TO COMPUTER
SCREEN
250 SR=ST: IF SR=0 OR SR=8 THEN 200:REM CHECK
STATUS, IF GOOD THEN CONTINUE
300 REM ERROR REPORTING
310 PRINT "ERROR: ";
320 IF SR AND 1 THEN PRINT "PARITY"
330 IF SR AND 2 THEN PRINT "FRAME"
340 IF SR AND 4 THEN PRINT "RECEIVER BUFFER FULL"
350 IF SR AND 128 THEN PRINT "BREAK"
360 IF (PEEK(673) AND 1) THEN 360:REM WAIT UNTIL
ALL CHARS TRANSMITTED
370 CLOSE 2: END
```

```
10 REM THIS PROGRAM SENDS AND RECEIVES TRUE ASCII
DATA
100 OPEN 5,2,3,CHR$(6)
110 DIM F%(255),T%(255)
200 FOR J=32 TO 64:T%(J)=J:NEXT
210 T%(13)=13:T%(20)=8:RV=18:CT=0
220 FOR J=65 TO 90:K=J+32:T%(J)=K:NEXT
230 FOR J=91 TO 95:T%(J)=J:NEXT
240 FOR J=193 TO 218:K=J-128:T%(J)=K:NEXT
250 T%(146)=16:T%(133)=16
260 FOR J=0 TO 255
270 K=T%(J)
280 IF K<>0THEN F%(K)=J:F%(K+128)=J
290 NEXT
300 PRINT " "CHR$(147)
310 GET#5,A$
320 IF A$=""OR ST<>0 THEN 360
330 PRINT " "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34 THEN POKE212,0
350 GOTO 310
360 PRINTCHR$(RV)" "CHR$(157);CHR$(146);:GET A$
370 IF A$<>""THENPRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THENCT=0:RV=164-RV
410 GOTO310
```

ZEIGER FÜR EMPFANGS-/ÜBERTRAGUNGSPUFFER

$00F7—RIBUF – Ein Zwei-Byte-Zeiger zur Empfangspuffer-Basisadresse.
$00F9—ROBUF – Ein Zwei-Byte-Zeiger zur Übertragungspuffer-Basisadresse.

Die beiden obigen Adressen werden durch die KERNAL-Routine OPEN bereitgestellt, wobei jede auf einen anderen 256-Byte-Puffer zeigt. Die Zuordnung wird annulliert, indem man in die höherwertigen Bytes ($00F8 und $00FA) über die KERNAL-Eingabe CLOSE eine Null schreibt. Die Zuordnung bzw. Annullierung kann auch durch Maschinensprache-Programme erfolgen, wobei der/die erforderlichen Puffer erstellt oder gelöscht werden. Beim Arbeiten mit einem Maschinensprache-Programm, das diese Puffer zuordnet, stets darauf achten, daß die Zeiger auf dem oberen Speicherende stehen. Dies gilt besonders dann, wenn gleichzeitig BASIC-Programme abgearbeitet werden sollen.

ZERO-PAGE-ADRESSEN UND IHRE ANWENDUNG FÜR DAS SYSTEM-INTERFACE RS-232

$00A7—INBIT — Empfänger Temp. Speicherung des Eingangsbits
$00A8—BITCI — Empfänger-Bitzählung EIN
$00A9—RINONE — Empfänger-Flag Startbit-Prüfung
$00AA—RIDATA — Empfänger-Bytepuffer/Assemblierplatz
$00AB—RIPRTY — Empfänger-Paritätsbit-Speicherung
$00B4—BITTS — Übertrager-Bit-Zählung AUS
$00B5—NXTBIT — Übertrager, nächstes zu übertragendes Bit
$00B6—RODATA — Übertrager-Byte-Puffer/Disassemblierplatz

Die obigen Zero-Page-Adressen sind lediglich als Hilfsmittel für die Erklärung der zugehörigen ROUTINEN gedacht. Sie können nicht direkt über BASIC- oder KERNAL-Programme benutzt werden, um RS-232-Funktionen auszuführen. Hierzu sind die System-Routinen RS-232 einzusetzen.

ADRESSEN, DIE NICHT IN DER ZERO-PAGE ENTHALTEN SIND UND IHRE ANWENDUNG FÜR DAS SYSTEM-INTERFACE RS-232

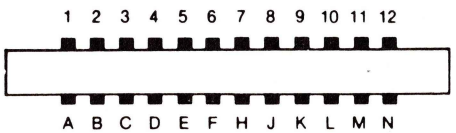
Allgemeine RS-232-Speicherung:

$0293—M51CTR — Pseudo-Steuerregister 6551 (siehe Abb. 6.1.)
$0294—M51COR — Pseudo-Befehlsregister 6551 (siehe Abb. 6.2.)
$0295—M51AJB — Zwei Bytes nach dem Steuer- und Befehlsregister im Dateinamenfeld. Diese Plätze enthalten die Baud-Rate für den Anfang des Bit-Tests während des Interface-Betriebs, in dem wiederum die Baud-Rate berechnet wird.
$0297—RSSTAT — Statusregister RS-232 (siehe Abb. 6.3.)
$0298—BITNUM — Anzahl der zu übertragenden/empfangenden Bits.
$0299—BAUDOF — Zwei Bytes, die der Zeit einer Bitzelle entsprechen. (Basierend auf Systemuhr/Baud-Rate.)
$029B—RIDBE — Byteindex zum Ende des Empfänger-FIFO-Puffers.
$029C—RIDBS — Byteindex zum Anfang des Empfänger-FIFO-Puffers.
$029D—RODBS — Byteindex zum Anfang des Übertragungs-FIFO-Puffers.
$029E—RODBE — Byteindex zum Ende des Übertragungs-FIFO-Puffers.
$02A1—ENABL — Verzögert derzeitig aktive Interrupts im CIA #2 ICR. Ist Bit 4 eingeschaltet, wartet das System auf das "Receiver Edge". Ist Bit 1 eingeschaltet, dann empfängt das System Daten. Ist Bit 0 eingeschaltet, überträgt das System Daten.

USERPORT

Über den Userport kann der COMMODORE 64 an die Außenwelt angeschlossen werden. Durch Verwendung der über diesen Port zur Verfügung stehenden Leitungen können Sie den COMMODORE 64 an einen Drucker, ein Modem und sogar an einen anderen Computer anschließen.
Der Port des COMMODORE 64 wird direkt an einen der Chips 6526 CIA angeschlossen. Durch Programmierung kann der CIA an zahlreiche andere Geräte angeschlossen werden.

PORT PIN DESCRIPTION



PORT-PIN-BESCHREIBUNG

PIN Oberseite	Beschreibung	Anmerkungen
1	GROUND	
2	+ 5 V	(Max. 100 mA)
3	RESET	Durch Erdung dieses Pins führt der COMMODORE 64 einen Kaltstart aus. Auch die Zeiger auf ein BASIC-Programm werden zurückgestellt, der Speicher jedoch nicht gelöscht. Gleichzeitig wird ein RESET-Signal an die Peripherie-Geräte gegeben.
4	CNT1	Zählereingang des seriellen Ports vom CIA #1 (CIA 6526-Datenblatt)
5	SP1	Serieller Port vom CIA #1 (siehe CIA 6526-Datenblatt)

PIN	Beschreibung	Anmerkungen
Oberseite		
8	PC2	Handshake-Leitung vom CIA –2 (siehe CIA 6526-Datenblatt)
9	SERIAL ATN	Dieser Anschluß ist mit der ATN-Leitung des seriellen Busses verbunden.
10	9 VAC+phase	Direkt an den Transformator des COMMODORE 64 angeschlossen (max. 50 mA).
11	9 VAC–phase	
12	GND	
Unterseite		
A	GND	Beim COMMODORE 64 ist der Port B des CIA #1-Chips frei verfügbar. Neben Ein-/Ausgabeleitungen stehen zwei Handshake-Leitungen zur Verfügung. Die Ein-/Ausgabeleitung von Port B wird über zwei Adressen gesteuert. Die eine Adresse ist der Port selbst und liegt bei 56577 ($DD01 in HEX). Auf diese Adresse können Sie die Befehle PEEK (Eingabe) und POKE (Ausgabe) anwenden. Jede der 8 Ein-/Ausgabeleitungen kann entweder als Eingabe- oder Ausgabeleitung definiert werden. Hierzu wird das Datenrichtungsregister entsprechend eingestellt.
B	FLAG 2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	

Das **DATENRICHTUNGS-REGISTER** liegt bei Adresse 56579 ($DD03 in HEX). Jede der acht Port-Leitungen hat ein Bit im 8-Bit-Datenrichtungs-Register (Data Direction Register = DDR), über das gesteuert wird, ob es sich um eine Eingabe- oder Ausgabeleitung handelt. Ist das Bit im DDR eine 1, dann ist die entsprechende Port-Leitung eine Ausgabeleitung. Ist das Bit auf 0 gesetzt, dann handelt es sich um eine Eingabeleitung. Ist z. B. Bit 3 des DDR auf 1 gesetzt, dann ist Port-Leitung 3 eine Ausgabeleitung. Ein weiteres Beispiel:

Das DDR ist wie folgt eingestellt:

BIT #: 7 6 5 4 3 2 1 0
WERT: 0 0 1 1 1 0 0 0

Die Leitungen 5, 4 und 3 sind Ausgabeleitungen, da diese Bits auf 1 gesetzt sind. Bei den restlichen Leitungen handelt es sich um Eingabeleitungen, da deren Bits auf 0 gesetzt sind.
Zum PEEKen oder POKEn des Userports muß sowohl das Datenrichtungs-Register als auch das Port-Register selbst benutzt werden. Die in dem Beispiel gegebenen Zahlen müssen vor der Verwendung in Dezimalzahlen umgewandelt werden.

$$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$$

(16 = 2↑4=2×2×2×2, 8 = 2↑3=2×2×2)

Die übrigen zwei Leitungen, FLAG1 und PA2, unterscheiden sich von den restlichen Benutzer-Port-Leitungen. Diese zwei Leitungen werden hauptsächlich für das "Handshaking" eingesetzt und müssen anders programmiert werden, als die Leitungen des Port B. Bei der Kommunikation zwischen zwei Geräten ist ein Handshaking-Betrieb erforderlich. Da der Datenaustausch bei den beiden Geräten verschieden lang dauern kann, ist es erforderlich, daß ein Gerät weiß, in welchem Zustand sich das andere gerade befindet. Auch wenn beide Übertragungsrichtungen gleich schnell sind, ist ein Handshake-Betrieb erforderlich, um anzuzeigen, wann Daten übertragen werden sollen und ob sie empfangen wurden. Leitung **FLAG2** hat besondere Eigenschaften, durch die sie sich besonders für diesen Zweck eignet.
FLAG2 ist ein für negative Flanken sensitiver Eingang, der als allgemeiner Interrupteingang benutzt werden kann. Jede negative Flanke auf der FLAG-Leitung setzt das FLAG-Interruptbit. Ist der FLAG-Interrupt zugelassen, dann führt dies zu einer INTERRUPT REQUEST (IRQ).

Ist der Flag-Interrupt nicht zugelassen, so kann eine eingetroffene negative Flanke durch Abfrage des Interrupt-Flag-Registers erkannt werden.
PA2 ist Bit 2 von Port A des CIA. Es wird genau wie andere Bits dieses Ports gesteuert. Der Port befindet sich in Adresse 65576 ($DD00). Das Datenrichtungs-Register befindet sich in Adresse 56578 ($DD02).
Weitere Einzelheiten über den 6526 entnehmen Sie bitte Anhang M.

DER SERIELLE BUS

Über den seriellen Bus kann der COMMODORE 64 mit anderen Geräten wie z. B. der VC-1541-Disketten-Station und dem VC-1525-Graphikdrucker kommunizieren. Der Vorteil dieses seriellen Busses liegt darin, daß bis zu 5 Geräte angeschlossen werden können. Es gibt verschiedene Funktionen, die am seriellen Bus möglich sind – "control", "talk" und "listen".
Ein als "CONTROLLER" fungierendes Gerät steuert die Kommunikation am seriellen Bus. Ein TALKER sendet Daten auf den Bus. Ein LISTENER empfängt Daten vom Bus.
Der COMMODORE 64 ist der CONTROLLER. Er kann auch TALKER sein, z. B. bei der Übertragung von Daten zum Drucker oder LISTENER (z. B. beim Laden eines Programms von Diskette). Andere Geräte können entweder LISTENER (der DRUKKER), TALKER oder beides (die DISKETTENSTATION) sein. Ausschließlich der COMMODORE 64 ist CONTROLLER.
Alle an den seriellen Bus angeschlossenen Geräte empfangen sämtliche über den Bus übertragenen Daten. Damit der COMMODORE 64 die Daten zum gewünschten Ziel übertragen kann, hat jedes Gerät eine Bus-Adresse. Durch Verwendung dieser Geräte-Adresse kann der COMMODORE 64 den Zugriff auf den Bus steuern. Die Adressen 4 bis 31 stehen zur Verfügung.
Der COMMODORE 64 kann ein bestimmtes Gerät anweisen, zu senden oder zu empfangen. Wenn ein Gerät diesen TALK-Befehl vom COMMODORE 64 erhält, beginnt es mit der Datenausgabe über den Serienbus. Empfängt ein Gerät den LISTEN-Befehl vom COMMODORE 64, dann bereitet sich dieses Gerät auf den Datenempfang vor (vom COMMODORE 64 oder einem anderen Gerät, das an den Bus angeschlossen ist). Gleichzeitig kann jeweils nur ein Gerät über den Bus übertragen, da es sonst zu einer Datenkollision kommt und das System zusammenbricht. Eine beliebige Anzahl an Geräten kann jedoch gleichzeitig die Daten empfangen.

STANDARD-ADRESSEN AUF DEM SERIELLEN BUS

NUMMER	GERÄT
4 oder 5	VC-1525 GRAPHIKDRUCKER
8	VC-1541 DISKETTENSTATION

Andere Geräteadressen sind möglich. Jedes Gerät hat seine eigene Adresse. Bestimmte Geräte (wie z. B. der COMMODORE 64-Drucker) bieten dem Anwender die Möglichkeit, zwischen zwei Adressen zu wählen.
Über die SEKUNDÄRADRESSE kann der COMMODORE 64 Betriebsinformationen zu einem Gerät übertragen. Z. B. um einen Kanal zum Drucker zu öffnen und einen Text in Groß-/Kleinschrift auszudrucken, benutzen Sie folgende Anweisung:

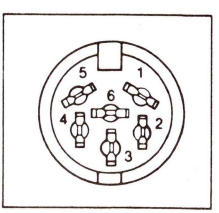
```
OPEN 1,4,7
```

wobei
1 logische Dateinummer (Nummer, zu der die Ausgabe PRINT# erfolgt)
4 Druckeradresse
7 gleich SEKUNDÄRADRESSE, die dem Drucker mitteilt, daß der Groß-Kleinschrift-Modus gewählt ist.

Der serielle Bus verwendet 6 Leitungen – 3 Eingabe- und 3 Ausgabeleitungen. Die 3 Eingabeleitungen übertragen Daten-, Steuer- und Timing-Signale zum COMMODORE 64. Die 3 Ausgabeleitungen übertragen Daten-, Steuer- und Timingsignale zu externen Geräten, die an den seriellen Bus angeschlossen sind.

ANSCHLÜSSE DES SERIELLEN BUSSES

PIN	BEZEICHNUNG
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NO CONNECTION



SERIAL SRQ IN: (SERIAL SERVICE REQUEST IN)

Jedes an den seriellen Bus angeschlossene Gerät kann dieses Signal auf Low ziehen, um den COMMODORE 64 auf sich aufmerksam zu machen (siehe Abb. 6.4.).

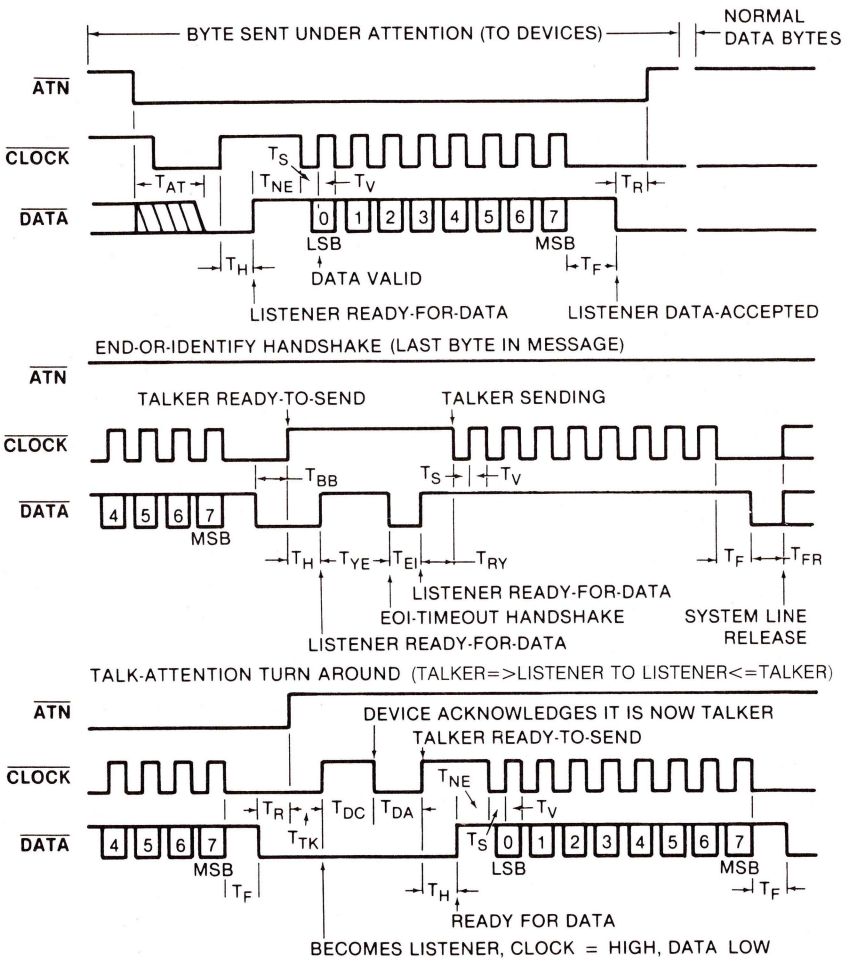
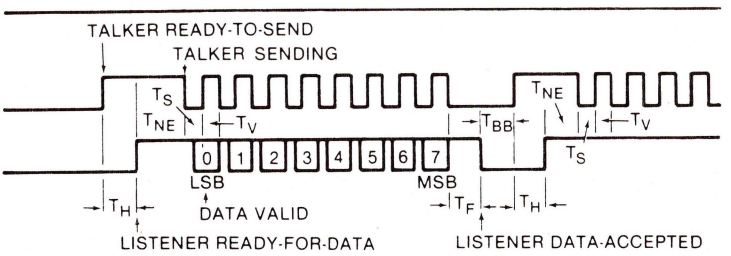


Abb. 6.4.

SERIAL ATN IN/OUT: (SERIAL ATTENTION IN/OUT)

Der COMMODORE 64 benutzt dieses Signal, um eine Befehlsfolge für ein an den seriellen Bus angeschlossenes Gerät zu beginnen. Setzt der COMMODORE 64 dieses Signal auf Low, dann warten alle an den Bus angeschlossenen Geräte auf eine vom COMMODORE 64 zu sendende Adresse. Das adressierte Gerät antwortet innerhalb eines festgelegten Zeitraums – anderenfalls nimmt der COMMODORE 64 an, daß das Gerät mit der speziellen Adresse nicht an den Bus angeschlossen ist, und gibt die Fehlermeldung im Statuswort aus (siehe Abb. 6.4.).



SERIENBUS TIMING

Description	Symbol	Min.	Typ.	Max.
ATN RESPONSE (REQUIRED)[1]	T_{AT}	—	—	1000µs
LISTENER HOLD OFF	T_H	0	—	∞
NON-EOI RESPONSE TO RFD[2]	T_{NE}	—	40µs	200µs
BIT SET-UP TALKER[4]	T_S	20µs	70µs	—
DATA VALID	T_V	20µs	20µs	—
FRAME HANDSHAKE[3]	T_F	0	20µs	1000µs
FRAME TO RELEASE OF ATN	T_R	20µs	—	—
BETWEEN BYTES TIME	T_{BB}	—	—	—
EOI RESPONSE TIME	T_{YE}	200µs	250µs	—
EOI RESPONSE HOLD TIME	T_{EI}	60µs	—	—
TALKER RESPONSE LIMIT	T_{RY}	0	30µs	60µs
BYTE-ACKNOWLEDGE[4]	T_{PR}	20µs	30µs	—

Notes:
1. If maximum time exceeded, device not present error.
2. If maximum time exceeded, EOI response required.
3. If maximum time exceeded, frame error.
4. T_V and T_{PR} minimum must be 60µs for external device to be a talker.

SERIAL CLK IN/OUT: (SERIAL CLOCK IN/OUT)

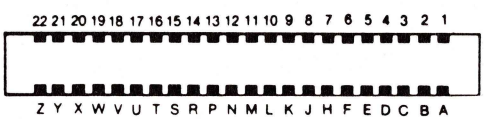
Dieses Signal wird für das "timing" der Datenübertragung über den seriellen Bus benutzt (siehe Abb. 6.4.).

SERIAL DATA IN/OUT:

Die Datenübertragung über diese Leitung des seriellen Busses geschieht Bit-seriell (siehe Abb. 6.4.).

ERWEITERUNGSANSCHLUSS

Der Erweiterungsanschluß ist als 44-Pin-Steckverbinder (22/22) ausgebildet. Wenn Sie vor dem COMMODORE 64 stehen, liegt der Erweiterungsanschluß ganz rechts auf der Computer-Rückseite. Um diesen Anschluß zu benutzen, ist ein entsprechender 44-Pin-Stecker erforderlich.
Dieser Anschluß wird für Erweiterungen des COMMODORE 64 benötigt, die den Zugriff auf den Adreß- oder Datenbus des Computers erfordern. Bei der Verwendung des Erweiterungsbusses ist vorsichtig vorzugehen, da der COMMODORE 64 sonst beschädigt werden kann.
Der Erweiterungsanschluß ist wie folgt belegt:



Folgende Signale sind an diesem Anschluß verfügbar:

NAME	PIN	BEZEICHNUNG
GND	1	Erdung
+ 5 VDC	2	(User-Port und Steckmodule dürfen zusammen nicht
+ 5 VDC	3	mehr als 450 mA verbrauchen.)
$\overline{\text{IRQ}}$	4	Interrupt-Request-Leitung zum 6510 (Aktiv-L-Pegel)
R/$\overline{\text{W}}$	5	Lesen/$\overline{\text{Schreiben}}$
DOT CLOCK	6	8,18 MHz Video-Dot-Clock
$\overline{\text{I/01}}$	7	Ein-/Ausgabe-Bereich 1 bei $DE00–$DFFF (Aktiv-L-Pegel LS TTL-Ausgang)
$\overline{\text{GAME}}$	8	Aktiv-L-Pegel LS TTL-Eingang
$\overline{\text{EXROM}}$	9	Aktiv-L-Pegel LS TTL-Eingang
$\overline{\text{I/02}}$	10	Ein-/Ausgabesatz 2 bei $DF00–$DFFF (Aktiv-L-Pegel LS TTL-Ausgang)
$\overline{\text{ROML}}$	11	Ausdekodierter 8K-RAM/ROM-Bereich bei $8000 (Aktiv-L-Pegel LS TTL-Ausgang)
BA	12	Bus-Available Signal vom VIC-Chip 6569 (nicht gepuffert, max. 1 LS TTL-Last)
$\overline{\text{DMA}}$	13	Direct-Memory-Access-Request-Leitung (Aktiv-L-Eingang, LS TTL)
D7	14	Datenbus-Bit 7
D6	15	Datenbus-Bit 6
D5	16	Datenbus-Bit 5
D4	17	Datenbus-Bit 4
D3	18	Datenbus-Bit 3
D2	19	Datenbus-Bit 2
D1	20	Datenbus-Bit 1
D0	21	Datenbus-Bit 0
		nicht gepuffert, max. 1 LS TTL-Last
GND	22	Erdung
GND	A	
$\overline{\text{ROMH}}$	B	Ausdekodierter 8K-RAM/ROM-Bereich bei $E000 (Aktiv-L-Pegel LS TTL-Ausgang)
$\overline{\text{RESET}}$	C	6510 RESET-Anschluß (Aktiv-L)
$\overline{\text{NMI}}$	D	6510 not Maskable Interrupt (Aktiv-L)
Ø2	E	Phase 2 Systemclock

NAME	PIN	BEZEICHNUNG	
A15	F	Adreßbus Bit 15	nicht gepuffert, max. 1 LS TTL-Last
A14	H	Adreßbus Bit 14	
A13	J	Adreßbus Bit 13	
A12	K	Adreßbus Bit 12	
A11	L	Adreßbus Bit 11	
A10	M	Adreßbus Bit 10	
A9	N	Adreßbus Bit 9	
A8	P	Adreßbus Bit 8	
A7	R	Adreßbus Bit 7	
A6	S	Adreßbus Bit 6	
A5	T	Adreßbus Bit 5	
A4	U	Adreßbus Bit 4	
A3	V	Adreßbus Bit 3	
A2	W	Adreßbus Bit 2	
A1	X	Adreßbus Bit 1	
A0	Y	Adreßbus Bit 0	
GND	Z	Erdung	
Ein Strich über dem Signalnamen bedeutet Aktiv-L			

Einige der wichtigsten Leitungen des Erweiterungsanschlusses werden nachfolgend beschrieben:

Die **Pins 1, 22, A** und **Z** sind geerdet.
An **Pin 6** liegt das Signal DOT CLOCK an. Über dieses Signal von 7,88 MHz erfolgt die gesamte Systemzeitsteuerung.
Pin 12 ist das BA-Signal (Bus-available) des VIC-Chip. Diese Leitung geht 3 Zyklen des Systemtaktes (Ø2) bevor der VIC-Chip den System-Bus vollständig übernimmt, auf Low. Dies gilt so lange, wie Anzeigeinformationen vom VIC-Chip abgerufen werden.
Pin 13 ist die DMA-Leitung (DIRECT MEMORY ACCESS-Leitung). Ist diese Leitung auf Low, so befinden sich Adreßbus, Datenbus und Read-/Writeleitung des Prozessors 6510 im hochohmigen Zustand. Auf diese Weise kann ein externer Prozessor die Steuerung des Systembusses übernehmen. Diese Leitung sollte nur auf Low gelegt werden, während der Ø2-Taktgeber L-Pegel hat. Da der VIC-Chip weiterhin Anzeige-DMA ausführt, muß der externe Prozessor außerdem mit der Zeitsteuerung des VIC-Chips übereinstimmen. (Siehe Timing-Diagramm des VIC-Chip.) Diese Leitung liegt beim COMMODORE 64 auf H-Pegel.

Z-80 MIKROPROZESSOR-MODUL

Beim Lesen dieses Buches und Arbeiten mit Ihrem Computer werden Sie festgestellt haben, wie vielseitig Ihr COMMODORE 64 wirklich ist. Noch wirkungsvoller zeigt er sich jedoch bei der Kombination mit Peripheriegeräten. Peripheriegeräte sind z. B. Datassette, Diskettenstationen, Drucker und Modems. Diese Geräte lassen sich über die verschiedenen Ports auf der Rückseite des COMMODORE 64 anschließen.
COMMODORE-Peripheriegeräte zeichnen sich besonders dadurch aus, daß sie "intelligent" sind, d. h., sie benötigen keinerlei RAM-Speicherkapazität. Sie können also den 64K-Speicher des COMMODORE 64 voll ausnutzen.
Ein weiterer Vorteil des COMMODORE 64 besteht darin, daß die meisten Programme, die Sie heute schreiben, auch noch mit den Geräten von morgen kompatibel sind. Dies liegt zum Teil am sinnvollen Aufbau des Betriebssystems (OS).
Eins kann das Betriebssystem des COMMODORE jedoch nicht: Ihre Programme für die Computer einer anderen Herstellerfirma kompatibel machen.
Da der COMMODORE 64 jedoch so einfach zu handhaben ist, werden Sie erst gar nicht daran denken, ein anderes Gerät zu benutzen. Für die Fälle, in denen ein Anwender jedoch Software benutzen möchte, die nicht im Format des COMMODORE 64 zur Verfügung steht, haben wir ein COMMODORE-CP/M®-Modul entwikkelt.
CP/M® ist kein "computerabhängiges" Betriebssystem. Vielmehr wird für das Betriebssystem Speicherplatz verwendet, der normalerweise für die Programmierung benutzt wird. Dies hat Vor- und Nachteile. Die Nachteile liegen darin, daß die Programme kürzer als bei Verwendung des eingebauten Betriebssystems sein müssen. Darüber hinaus kann nicht mit den Bildschirm-Editierfunktionen des COMMODORE 64 gearbeitet werden. Von Vorteil ist jedoch, daß sie nun wesentlich mehr Software speziell für CP/M® und den Mikroprozessor Z-80 benutzen können und daß die über dieses CP/M®-Betriebssystem geschriebenen Programme auf beliebige andere Computer übertragen und dort ausgeführt werden können, die mit CP/M® und Z-80-Karte ausgerüstet sind.
Bei den meisten Computern mit Z-80-Mikroprozessor muß die Z-80-Karte übrigens im Gerät eingebaut werden. Hierbei ist besonders vorsichtig vorzugehen, da leicht die empfindliche Schaltung beschädigt wird. Beim COMMODORE-Steckmodul CP/M® ist dies nicht erforderlich, da es schnell und einfach an der Rückseite Ihres COMMODORE 64 aufgesteckt wird.

ARBEITEN MIT COMMODORE CP/M®

Mit dem COMMODORE-Modul Z-80 können Sie für einen Z-80-Mikroprozessor entworfene Programme auf Ihrem COMMODORE 64 laufen lassen. Zur Z-80-Karte gehört auch eine Diskette mit dem COMMODORE-CP/M®-Betriebssystem.

AUSFÜHRUNG

Zur Ausführung von CP/M®:

1) CP/M®-Programm von der Diskette laden.
2) Über die Tastatur RUN eingeben.
3) RETURN -Taste drücken.

Beim COMMODORE 64 sind 64K-Byte RAM durch den Prozessor 6510 oder 48K-Byte RAM durch den Prozessor Z-80 erreichbar. Ein Schalten zwischen diesen beiden Prozessoren ist möglich, sie können jedoch nicht gleichzeitig in ein und demselben Programm benutzt werden. Das Umschalten wird durch die raffinierte Timing-Technik des COMMODORE 64 möglich.
Nachfolgend sehen Sie die Speicheradressen-Verschiebung für das Z-80-Modul. Bitte beachten Sie, daß durch Hinzufügen von 4096 Bytes zu den von CP/M® benutzten Speicherplätzen sich die Speicheradressen des normalen Betriebssystems des COMMODORE 64 ergeben. Die Speicheradressen von Z-80 und 6510 stehen in folgendem Zusammenhang:

ADRESSEN, Z-80		ADRESSEN, 6510	
DEZIMAL	HEXADEZIMAL	DEZIMAL	HEXADEZIMAL
0000–4095	0000–0FFF	4096–8191	1000–1FFF
4096–8191	1000–1FFF	8192–12287	2000–2FFF
8192–12287	2000–2FFF	12288–16383	3000–3FFF
12288–16383	3000–3FFF	16384–20479	4000–4FFF
16384–20479	4000–4FFF	20480–24575	5000–5FFF
20480–24575	5000–5FFF	24576–28671	6000–6FFF
24576–28671	6000–6FFF	28672–32767	7000–7FFF
28672–32767	7000–7FFF	32768–36863	8000–8FFF
32768–36863	8000–8FFF	36864–40959	9000–9FFF
36864–40959	9000–9FFF	40960–45055	A000–AFFF
40960–45055	A000–AFFF	45056–49151	B000–BFFF
45056–49151	B000–BFFF	49152–53247	C000–CFFF
49152–53247	C000–CFFF	53248–57343	D000–DFFF
53248–57343	D000–DFFF	57344–61439	E000–EFFF
57344–61439	E000–EFFF	61440–65535	F000–FFFF
61440–65535	F000–FFFF	0000–4095	0000–0FFF

Um den Z-80 einzuschalten und den 6510 auszuschalten, geben Sie folgendes Programm ein:

```
10 REM THIS PROGRAM IS TO BE USED WITH THE Z80 CARD
20 REM IT FIRST STORES Z80 DATA AT $1000
(Z80=$0000)
30 REM THEN IT TURNS OFF THE 6510 IRQ'S AND ENABLES
40 REM THE Z80 CARD.  THE Z80 CARD MUST BE TURNED
OFF
50 REM TO REENABLE THE 6510 SYSTEM.
100 REM STORE Z80 DATA
110 READ B: REM GET SIZE OF Z80 CODE TO BE MOVED
120 FOR I=4096 TO 4096+B-1:REM MOVE CODE
130 READ A:POKE I,A
140 NEXT I
200 REM RUN Z80 CODE
210 POKE 56333,127 : REM TURN OF 6510 IRQ'S
220 POKE 56832,00  : REM TURN ON Z80 CARD
230 POKE 56333,129 : REM TURN ON 6510 IRQ'S WHEN
Z80 DONE
240 END
1000 REM Z80 MACHINE LANGUAGE CODE DATA SECTION
1010 DATA 18 : REM SIZE OF DATA TO BE PASSED
1100 REM Z80 TURN ON CODE
1110 DATA 00,00,00 : REM OUR Z80 CARD REQUIRES
TURN ON TIME AT $0000
1200 REM Z80 TASK DATA HERE
1210 DATA 33,02,245 : REM LD HL,NN (LOCATION ON
SCREEN)
1220 DATA 52 : REM INC HL (INCREMENT THAT LOCATION)
1300 REM Z80 SELF-TURN OFF DATA HERE
1310 DATA 62,01 : REM LD A,N
1320 DATA 50,00,206 : REM LD (NN),A :I/O LOCATION
1330 DATA 00,00,00 : REM NOP: NOP: NOP
1340 DATA 195,00,00 : REM JMP $0000
```

Bezüglich weiterer Einzelheiten über COMMODORE CP/M® und den Mikroprozessor Z-80 fragen Sie bitte Ihren COMMODORE-Händler.

ANHANG

ANHANG A

ABKÜRZUNGEN DER BASIC-SCHLÜSSELWÖRTER

Zur Zeitersparnis können bei der Eingabe von Programmen und Befehlen beim COMMODORE 64 die meisten BASIC-Schlüsselwörter abgekürzt werden. Die Abkürzung für PRINT ist z. B. ein Fragezeichen. Die übrigen Wörter werden wie folgt abgekürzt: Eingabe des ersten bzw. der ersten zwei Buchstaben, danach der nächste Buchstabe mit SHIFT-Taste. Werden Abkürzungen in Programmzeilen benutzt, dann erscheint das Schlüsselwort bei der Auflistung in ausgeschriebener Form.

Befehl	Abkürzung	Bildschirmanzeige	Befehl	Abkürzung	Bildschirmanzeige
ABS	A SHIFT B	A	END	E SHIFT N	E
AND	A SHIFT N	A	EXP	E SHIFT X	E
ASC	A SHIFT S	A	FN	NONE	FN
ATN	A SHIFT T	A	FOR	F SHIFT O	F
CHR$	C SHIFT H	C	FRE	F SHIFT R	F
CLOSE	CL SHIFT O	CL	GET	G SHIFT E	G
CLR	C SHIFT L	C	GET#	NONE	GET#
CMD	C SHIFT M	C	GOSUB	GO SHIFT S	GO
CONT	C SHIFT O	C	GOTO	G SHIFT O	G
COS	NONE	COS	IF	NONE	IF
DATA	D SHIFT A	D	INPUT	NONE	INPUT
DEF	D SHIFT E	D	INPUT#	I SHIFT N	I
DIM	D SHIFT I	D	INT	NONE	INT

Befehl	Abkürzung	Bildschirmanzeige
LEFT$	LE SHIFT F	LE
LEN	NONE	LEN
LET	L SHIFT E	L
LIST	L SHIFT I	L
LOAD	L SHIFT O	L
LOG	NONE	LOG
MID$	M SHIFT I	M
NEW	NONE	NEW
NEXT	N SHIFT E	N
NOT	N SHIFT O	N
ON	NONE	ON
OPEN	O SHIFT P	O
OR	NONE	OR
PEEK	P SHIFT E	P
POKE	P SHIFT O	P
POS	NONE	POS
PRINT	?	?
PRINT#	P SHIFT R	P
READ	R SHIFT E	R
REM	NONE	REM
RESTORE	RE SHIFT S	RE
RETURN	RE SHIFT T	RE

Befehl	Abkürzung	Bildschirmanzeige
RIGHT$	R SHIFT I	R
RND	R SHIFT N	R
RUN	R SHIFT U	R
SAVE	S SHIFT A	S
SGN	S SHIFT G	S
SIN	S SHIFT I	S
SPC(	S SHIFT P	S
SQR	S SHIFT Q	S
STATUS	ST	ST
STEP	ST SHIFT E	ST
STOP	S SHIFT T	S
STR$	ST SHIFT R	ST
SYS	S SHIFT Y	S
TAB(	T SHIFT A	T
TAN	NONE	TAN
THEN	T SHIFT H	T
TIME	TI	TI
TIME$	TI$	TI$
USR	U SHIFT S	U
VAL	V SHIFT A	V
VERIFY	V SHIFT E	V
WAIT	W SHIFT A	W

ANHANG B

BILDSCHIRM-ANZEIGE-CODES

Nachfolgend werden sämtliche Zeichen aufgelistet, die mit den Zeichensätzen des COMMODORE 64 möglich sind. In der Tabelle wird gezeigt, welche Zahlen für ein gewünschtes Zeichen in den Bildschirmspeicher (Plätze 1024–2023) gePOKEt werden müssen. Außerdem sehen Sie, welches Zeichen einer vom Bildschirm gePEEKten Zahl entspricht.

Es stehen zwei Zeichensätze zur Verfügung, von denen jeweils einer gewählt werden kann. D. h., bei der Anzeige von Zeichen eines Satzes ist der andere Satz nicht wirksam. Zum Umschalten der Zeichensätze werden gleichzeitig die Tasten SHIFT und C= gedrückt.

Bei BASIC wird durch POKE 53272,21 in den Großschrift-Graphik-Modus und durch POKE 53272,23 in den Klein-Großschrift-Modus umgeschaltet.

Alle in der Tabelle gezeigten Zahlen können negativ dargestellt werden. Den entsprechenden Adreß-Code erhält man, indem zu den gezeigten Werten 128 addiert wird. Soll ein ausgefüllter Kreis in Bildschirmadresse 1504 dargestellt werden, dann POKEn Sie den Code für den Kreis (81) in Adresse 1504: POKE 1504,81.

Für die Farbsteuerung der einzelnen auf dem Bildschirm angezeigten Zeichen existiert ein entsprechender Speicherbereich (Adressen 55296–56295). Um die Farbe des Kreises z. B. in Gelb zu ändern (Farb-Code 7), wird der Farb-Code in den entsprechenden Speicherplatz (55776) gePOKEt: POKE 55776,7.

Die vollständigen Bildschirm- und Farbspeicherbelegungen sowie die Farb-Codes finden Sie in Anhang D.

BILDSCHIRM-CODES

SATZ 1	SATZ 2	POKE	SATZ 1	SATZ 2	POKE	SATZ 1	SATZ 2	POKE
@		0	C	c	3	F	f	6
A	a	1	D	d	4	G	g	7
B	b	2	E	e	5	H	h	8

SATZ 1	SATZ 2	POKE	SATZ 1	SATZ 2	POKE	SATZ 1	SATZ 2	POKE
I	i	9	%		37		A	65
J	j	10	&		38		B	66
K	k	11	'		39		C	67
L	l	12	(		40		D	68
M	m	13	)		41		E	69
N	n	14	*		42		F	70
O	o	15	+		43		G	71
P	p	16	,		44		H	72
Q	q	17	–		45		I	73
R	r	18	.		46		J	74
S	s	19	/		47		K	75
T	t	20	0		48		L	76
U	u	21	1		49		M	77
V	v	22	2		50		N	78
W	w	23	3		51		O	79
X	x	24	4		52		P	80
Y	y	25	5		53		Q	81
Z	z	26	6		54		R	82
[		27	7		55		S	83
£		28	8		56		T	84
]		29	9		57		U	85
↑		30	:		58		V	86
←		31	;		59		W	87
SPACE		32	<		60		X	88
!		33	=		61		Y	89
"		34	>		62		Z	90
#		35	?		63			91
$		36			64			92

SATZ 1	SATZ 2	POKE	SATZ 1	SATZ 2	POKE	SATZ 1	SATZ 2	POKE
		93			105			117
		94			106			118
		95			107			119
SPACE		96			108			120
		97			109			121
		98			110			122
		99			111			123
		100			112			124
		101			113			125
		102			114			126
		103			115			127
		104			116			

Die Codes 128–258 sind die umgekehrten Bilder von 0–127.

ANHANG C

ASCII- UND CHR$-CODES

In diesem Anhang werden für alle X-Werte gezeigt, welche Zeichen bei der Anzeige PRINT CHR$ (X) erscheinen. Außerdem finden Sie hier die über PRINT ASC ("X") möglichen Werte, wobei X ein beliebiges einzugebendes Zeichen ist. Dies ist besonders nützlich bei der Auswertung von in einer GET-Anweisung empfangenen Zeichen, der Umwandlung von Zeichen der oberen/unteren Umschaltstellung sowie der Anzeige von Zeichen-Befehlen (z. B. Umschaltung zu Zeichen der oberen/unteren Umschaltstellung), die nicht in Anführungszeichen stehen können.

ANZEIGE	CHR$	ANZEIGE	CHR$	ANZEIGE	CHR$	ANZEIGE	CHR$
	0	CRSR ↑	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
un-wirksam	SHIFT + C= 8		25	*	42	;	59
wirksam	SHIFT + C= 9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	–	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
SWITCH TO LOWER CASE	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

ANZEIGE	CHR$	ANZEIGE	CHR$	ANZEIGE	CHR$	ANZEIGE	CHR$
D	68		97		126		155
E	69		98		127		156
F	70		99		128		157
G	71		100	Orange	129		158
H	72		101		130		159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	SWITCH TO UPPER CASE	142		171
U	85		114		143		172
V	86		115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120		149		178
£	92		121		150		179
]	93		122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183

ANZEIGE	CHR$	ANZEIGE	CHR$	ANZEIGE	CHR$	ANZEIGE	CHR$
	184		186		188		190
	185		187		189		191

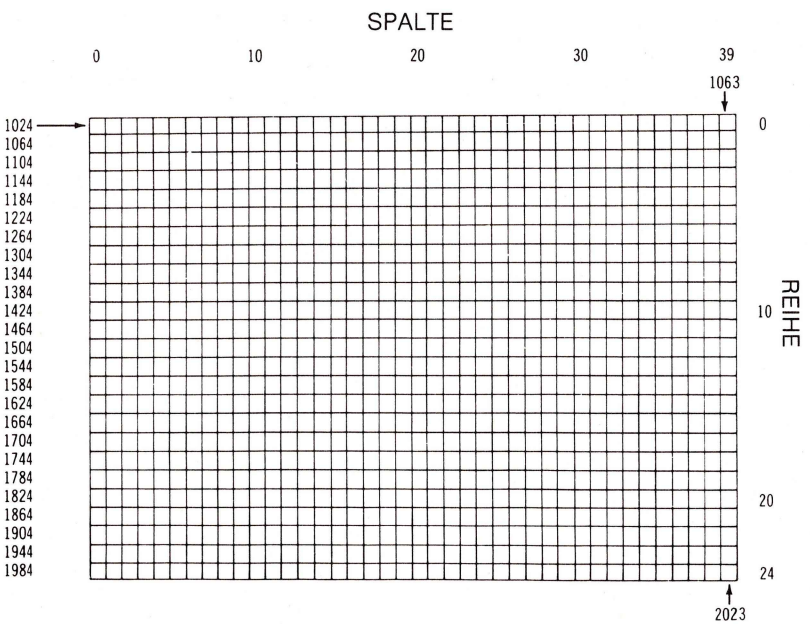
CODES 192-223 SAME AS 96-127
CODES 224-254 SAME AS 160-190
CODE 255 SAME AS 126

ANHANG D

BILDSCHIRM- UND FARBSPEICHERMAPPEN

In der nachstehenden Tabelle finden Sie die für die Bildschirm-Zeichensteuerung verantwortlichen Speicherplätze, die Plätze zur Änderung einzelner Zeichen-Farben sowie Zeichen-Farb-Codes.

BILDSCHIRM-SPEICHERBELEGUNG

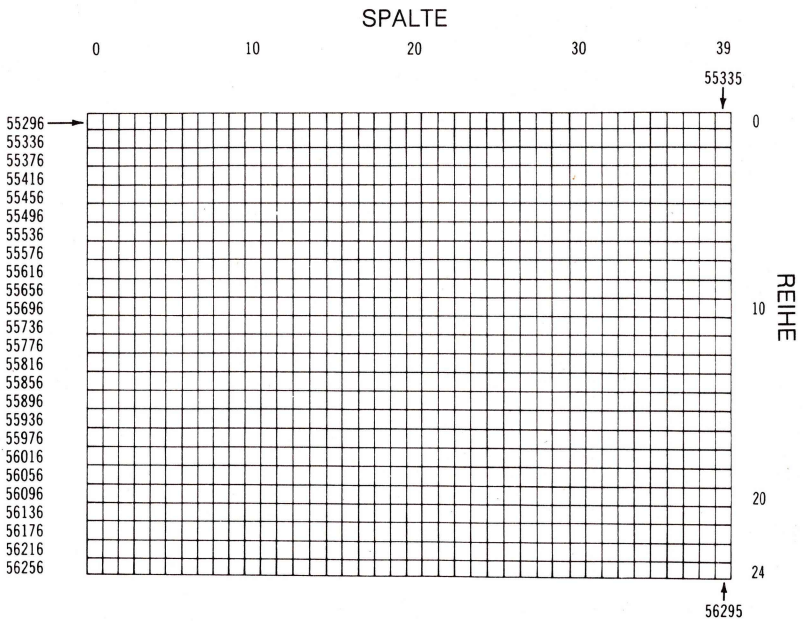


Zum Ändern einer Zeichenfarbe müssen folgende Werte in einen Farbspeicher-Platz gePOKEt werden:

0 SCHWARZ	8 ORANGE
1 WEISS	9 BRAUN
2 ROT	10 HELLROT
3 ZYAN	11 GRAU 1
4 PURPUR	12 GRAU 2
5 GRÜN	13 HELLGRÜN
6 BLAU	14 HELLBLAU
7 GELB	15 GRAU 3

Um z. B. die Farbe eines Zeichens oben links auf dem Bildschirm in Rot umzuändern, geben Sie folgendes ein: POKE 55296,2.

FARBSPEICHERBELEGUNG



ANHANG E

MUSIKNOTENWERTE

In diesem Anhang finden Sie eine vollständige Liste der Noten, zugehörigen Frequenzen und Frequenzparameter und der Werte, die in die Register FREQ HI und FREQ LO des Klangchips gePOKEt werden müssen, um den gewünschten Ton zu erzeugen.

NOTE	OKTAVE	DEZIMAL	HI	LOW
0	C–0	278	1	22
1	C#–0	295	1	39
2	D–0	313	1	57
3	D#–0	331	1	75
4	E–0	351	1	95
5	F–0	372	1	116
6	F#–0	394	1	138
7	G–0	417	1	161
8	G#–0	442	1	186
9	A–0	468	1	212
10	A#–0	496	1	240
11	H–0	526	2	14
12	C–1	557	2	45
13	C#–1	590	2	78
14	D–1	625	2	113
15	D#–1	662	2	150
16	E–1	702	2	190
17	F–1	743	2	231
18	F#–1	788	3	20
19	G–1	834	3	66
20	G#–1	884	3	116
21	A–1	937	3	169
22	A#–1	992	3	224
23	H–1	1051	4	27
24	C–2	1114	4	90

NOTE	OKTAVE	DEZIMAL	HI	LOW
25	C#−2	1180	4	156
26	D−2	1250	4	226
27	D#−2	1325	5	45
28	E−2	1403	5	123
29	F−2	1487	5	207
30	F#−2	1575	6	39
31	G−2	1669	6	133
32	G#−2	1768	6	232
33	A−2	1873	7	81
34	A#−2	1985	7	193
35	H−2	2103	8	55
36	C−3	2228	8	180
37	C#−3	2360	9	56
38	D−3	2500	9	196
39	D#−3	2649	10	89
40	E−3	2807	10	247
41	F−3	2974	11	158
42	F#−3	3150	12	78
43	G−3	3338	13	10
44	G#−3	3536	13	208
45	A−3	3746	14	162
46	A#−3	3969	15	129
47	H−3	4205	16	109
48	C−4	4455	17	103
49	C#−4	4720	18	112
50	D−4	5001	19	137
51	D#−4	5298	20	178
52	E−4	5613	21	237
53	F−4	5947	23	59
54	F#−4	6301	24	157
55	G−4	6676	26	20
56	G#−4	7072	27	160
57	A−4	7493	29	69
58	A#−4	7939	31	3
59	H−4	8411	32	219
60	C−5	8911	34	207
61	C#−5	9441	36	225
62	D−5	10002	39	18
63	D#−5	10597	41	101

NOTE	OKTAVE	DEZIMAL	HI	LOW
64	E–5	11227	43	219
65	F–5	11894	46	118
66	F#–5	12602	49	58
67	G–5	13351	52	39
68	G#–5	14145	55	65
69	A–5	14986	58	138
70	A#–5	15877	62	5
71	H–5	16821	65	181
72	C–6	17821	69	157
73	C#–6	18881	73	193
74	D–6	20004	78	36
75	D#–6	21193	82	201
76	E–6	22454	87	182
77	F–6	23789	92	237
78	F#–6	25203	98	115
79	G–6	26702	104	78
80	G#–6	28290	110	130
81	A–6	29972	117	20
82	A#–6	31754	124	10
83	H–6	33642	131	106
84	C–7	35643	139	59
85	C#–7	37762	147	130
86	D–7	40008	156	72
87	D#–7	42387	165	147
88	E–7	44907	175	107
89	F–7	47578	185	218
90	F#–7	50407	196	231
91	G–7	53404	208	156
92	G#–7	56580	221	4
93	A–7	59944	234	40
94	A#–7	63508	248	20

Sie sind nicht an die Werte dieser Tabelle gebunden! Wenn Sie mehrere Stimmen benutzen, sollten Sie sogar bewußt die zweite und dritte Stimme etwas „verstimmen", d. h. das Lo-Byte aus der Tabelle leicht (!) abändern. Sie bekommen so einen volleren Klang.

FILTEREINSTELLUNGEN

Adresse	Inhalt
54293	Grenzfrequenz, Low Byte (0–7)
54294	Grenzfrequenz, High Byte (0–255)
54295	Resonanz (Bits 4–7)
	Filter, Stimme 3 (Bit 2)
	Filter, Stimme 2 (Bit 1)
	Filter, Stimme 1 (Bit 0)
54296	Hochpaß (Bit 6)
	Bandpaß (Bit 5)
	Tiefpaß (Bit 4)
	Lautstärke (Bits 0–3)

ANHANG F

LITERATURVERZEICHNIS

Addison-Wesley	"BASIC and the Personal Computer", Dwyer and Critchfield
Compute	"Compute's First Book of PET/CBM"
Cowbay Computing	"Feed me, I'm Your PET Computer", Carol Alexander "Looking good with your PET", Carol Alexander "Teacher's PET – Plans, Quizzes, and Answers"
Creative Computing	"Getting Acquainted With Your VIC 20", T. Hartnell
Dilithium Press	"BASIC Basic-English Dictionary for the PET", Larry Noonan "PET NASIC", Tom Rugg and Phil Feldman
Faulk Baker Associates	"MOS Programming Manual", MOS Technology
Hayden Book Co.	"BASIC From the Ground UP", David E. Simon "I Speak BASIC to My PET", Aubrey Jones, jr. "Library of PET Subroutines", Nick Hampshire "PET Graphics", Nick Hampshire "BASIC conversions Handbook, Apple, TRS-80, and PET", David A. Brain, Phillip R. Oviatt, Paul J. Paquin, and Chandler P. Stone
Howard W. Sams	"The Howard W. Sams Crash Course in Microcomputers", Louis E. Frenzel, jr. "Mostly BASIC: Application for your PET", Howard Berenbon "PET Interfacing", James N. Downey and Steven M. Rogers "VIC 20 Programmer's Reference Guide", A. Finkel, P. Higginbottom, N. Harris, and M. Tomczyk

Little, Brown & Co.	"Computer Games for Business, Schools, and Homes", J. Victor Nagigian, and William S. Hodges "The Computer Tutor: Learning Activities for Homes and Schools", Gary W. Orwig, University of Central Florida, and William S. Hodges
McGraw-Hill	"Hands-On BASIC With a PET", Herbert D. Peckman "Home and Office Use of VisiCalc", D. Castlewitz, and L. Chisauki
Osborne/McGraw-Hill	"PET/CBM Personal Computer Guide", Carroll S. Donahue "PET Fun and Games", R. Jeffries and G. Fisher "PET and the IEEE", A. Osborne and C. Donahue "Some Common BASIC Programms for the OET", L. Poole, M. Borchers, and C. Donahue "Osborne CP/M User Guide", Thom Hogan "CBM Professional Computer Guide" "The PET Personal Guide" "The 8086 Book", Russell Rector and George Alexy
P. C. Publications	"Beginning Self-Teaching Computer Lessons"
Prentice-Hall	"The PET Personal Computer for Beginners", S. Dunn and V. Morgan
Reston Publishing Co.	"PET and the IEEE 488 Bus (GPIB)", Eugene Fisher and C. W. Jensen "PET BASIC – Training Your PET Computer", Ramon Zamora, Wm. F. Carrie, and B. Allbrecht "PET Games and Recreation", M. Ogelsby, L. Lindsey, and D. Kunkin "PET BASIC", Richard Huskell "VIC Games and Recreation"

Telmas Courseware	"BASIC and the Personal Computer", T. A. Dwyer, and M. Critchfield
Total Information Services	"Understanding Your PET/CBM, Vol. 1, BASIC Programming" "Understanding Your VIC", David Schultz

In den COMMODORE-Zeitschriften finden Sie die aktuellsten Informationen über Ihren COMMODORE 64. Hiervon möchten wir die folgenden zwei besonders empfehlen:

COMMODORE – Die Mikrocomputer-Zeitschrift erscheint zweimal monatlich und kann abonniert werden ($25,00 für ein Jahr).

POWER/PLAY – Diese vierteljährlich erscheinende Home-Computer-Zeitschrift kann ebenfalls abonniert werden ($15,00 pro Jahr).

ANHANG G

VIC-CHIP REGISTERBELEGUNG

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7							S0X0	SPRITE 0 X Component
1	1	S0Y7							S0Y0	SPRITE 0 Y Component
2	2	S1X7							S1X0	SPRITE 1 X
3	3	S1Y7							S1Y0	SPRITE 1 Y
4	4	S2X7							S2X0	SPRITE 2 X
5	5	S2Y7							S2Y0	SPRITE 2 Y
6	6	S3X7							S3X0	SPRITE 3 X
7	7	S3Y7							S3Y0	SPRITE 3 Y
8	8	S4X7							S4X0	SPRITE 4 X
9	9	S4Y7							S4Y0	SPRITE 4 Y
10	A	S5X7							S5X0	SPRITE 5 X
11	B	S5Y7							S5Y0	SPRITE 5 Y
12	C	S6X7							S6X0	SPRITE 6 X
13	D	S6Y7							S6Y0	SPRITE 6 Y
14	E	S7X7							S7X0	SPRITE 7 X Component
15	F	S7Y7							S7Y0	SPRITE 7 Y Component
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	MSB of X COORD.
17	11	RC8	ECM	BMM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	Y SCROLL MODE
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASTER
19	13	LPX7							LPX0	LIGHT PEN X
20	14	LPY7							LPY0	LIGHT PEN Y

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
21	15	SE7							SE0	SPRITE ENABLE (ON/OFF)
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCL2	XSCL1	XSCL0	X SCROLL MODE
23	17	SEXY7							SEXY0	SPRITE EXPAND Y
24	18	VS13	VS12	VS11	VS10	CB13	CB12	CB11	N.C.	SCREEN and Character Memory Base Address
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Interrupt Request's
26	1A	N.C.	N.C.	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Interrupt Request MASKS
27	1B	BSP7							BSP0	Background-Sprite PRIORITY
28	1C	SCM7							SCM0	MULTICOLOR SPRITE SELECT
29	1D	SEXX7							SEXX0	SPRITE EXPAND X
30	1E	SSC7							SSC0	Sprite-Sprite COLLISION
31	1F	SBC7							SBC0	Sprite-Background COLLISION

COLOR CODES					DEC				HEX	COLOR
32	20	0	0	BLACK	EXT 1					EXTERIOR COL
33	21	1	1	WHITE	BKGD0					
34	22	2	2	RED	BKGD1					
35	23	3	3	CYAN	BKGD2					
36	24	4	4	PURPLE	BKGD3					
37	25	5	5	GREEN	SMC 0					SPRITE MULTICOLOR 0
38	26	6	6	BLUE	SMC 1					1
39	27	7	7	YELLOW	S0COL					SPRITE 0 COLOR
40	28	8	8	ORANGE	S1COL					1
41	29	9	9	BROWN	S2COL					2
42	2A	10	A	LT RED	S3COL					3
43	2B	11	B	GRAY 1	S4COL					4
44	2C	12	C	GRAY 2	S5COL					5
45	2D	13	D	LT GREEN	S6COL					6
46	2E	14	E	LT BLUE	S7COL					7
		15	F	GRAY 3						

Anmerkung: Im Mehrfarben-Zeichenmodus können nur die Farben 0–7 benutzt werden.

ANHANG H

ABGELEITETE MATHEMATISCHE FUNKTIONEN

Funktionen, die in Commodore-64-Basic nicht vordefiniert sind, können mit Hilfe der folgenden Formel berechnet werden:

FUNKTION	BASIC-ENTSPRECHUNG
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(−X*X+1))
INVERSE COSINE	ARCCOS(X)=−ATN(X/SQR (−X*X +1)) +π/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X−1))
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X−1)) +(SGN(X)−1*π/2
INVERSE COTANGENT	ARCOT(X)=ATN(X)+π/2
HYPERBOLIC SINE	SINH(X)=(EXP(X)−EXP(−X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(−X))/2
HYPERBOLIC TANGENT	TANH(X)=EXP(−X)/(EXP(x)+EXP (−X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(−X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)−EXP(−X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(−X)/(EXP(X) −EXP(−X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X−1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1−X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR (−X*X+1)+1/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR (X*X+1/x)
INVERSE HYPERBOLIC COTAN-GENT	ARCCOTH(X)=LOG((X+1)/(X−1))/2

ANHANG I

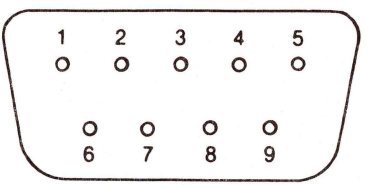
STECKERBELEGUNG DER ANSCHLÜSSE FÜR PERIPHERIEGERÄTE

Dieser Anhang soll Ihnen zeigen, wie welches Gerät wo an den COMMODORE 64 angeschlossen werden kann.

1) Steuereingänge für Spiele
2) Modul-Steckplatz
3) Audio/Video
4) Serieller E/A (Disk/Drucker)
5) Kassette
6) User Port

Control Port 1

Pin	Signal	Bemerkung
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY**	
6	BUTTON A/LP*	
7	± 5 V	MAX. 100 mA
8	GND	
9	POT AX**	



Control Port 2

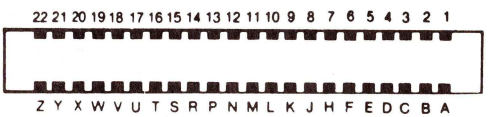
Pin	Signal	Bemerkung
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY**	
6	BUTTON B	
7	+ 5 V	MAX. 100 mA
8	GND	
9	POT BX**	

*) Button = Feuerknopf am LP = Light pen.
**) POT = Paddle Potentiometer

Modul-Steckplatz

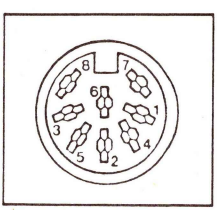
Pin	Signal
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	DMA
12	BA
11	ROML
10	I/O2
9	EXROM
8	GAME
7	I/O1
6	Dot Clock
5	CR/W
4	IRQ
3	+ 5 V
2	+ 5 V
1	GND

Pin	Signal
Z	GND
Y	CA0
X	CA1
W	CA2
V	CA3
U	CA4
T	DA5
S	CA6
R	CA7
P	CA8
N	CA9
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	Ø2
D	NMI
C	RESET
B	ROMH
A	GND



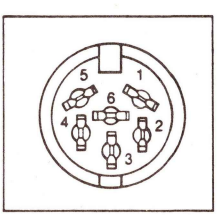
Audio/Video

Pin	Signal
1	LUMINANCE
2	GND
3	AUDIO OUT
4	VIDEO OUT
5	AUDIO IN
6	CHROMINANCE
7	NICHT ANGESCHLOSSEN
8	NICHT ANGESCHLOSSEN



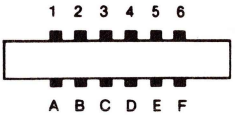
Serielle E/A

Pin	Signal
1	SERIAL SRQIN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET



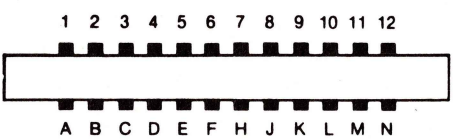
Cassette

Pin	Signal
A-1	GND
B-2	+ 5 V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE



User Port

Pin	Signal	Bemerkung
1	GND	
2	+ 5 V	MAX. 100 mA
3	RESET	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	
A	GND	
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	



ANHANG J

ÜBERTRAGUNG VON FREMDEN BASIC-PROGRAMMEN AUF COMMODORE-64-BASIC

Besitzen Sie Programme, die in einer anderen BASIC-Version als COMMODORE-BASIC geschrieben wurden, werden einige kleinere Anpassungen nötig sein, bevor sie auf dem COMMODORE 64 laufen können. Wir geben Ihnen nun einige Tips, die die Anpassung leichter machen.

Dimensionen von Strings

Entfernen Sie alle Statements, die die Länge eines Strings festlegen. Ein Befehl wie etwa DIM A$(I,J), der ein Stringarray für J Elemente der Länge I dimensioniert, muß in das COMMODORE-BASIC-Statement DIM A$(J) abgeändert werden.
Einige BASIC-Versionen benutzen ein Komma (,) oder Kaufmanns-Und (&) zur Verknüpfung von Strings. Diese müssen in ein Plus-Zeichen (+) geändert werden, das in COMMODORE-BASIC der entsprechende Operator zur Stringverknüpfung ist.
Im BASIC des COMMODORE 64 dienen die Funktionen MID$, RIGHT$ und LEFT$ der Erzeugung von Teilstrings. Formen wie A$(I) zur Ansprache des I-ten Zeichens in String A$ oder A$(I,J) zur Gewinnung des Teilstrings von A$ von Position I bis J müssen wie folgt geändert werden:

sonstiges BASIC	COMMODORE-64-BASIC
A$(I) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,I+1)
A$(I,J) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,J+1)

Mehrfache Zuweisungen

Um die Variablen B und C gleichzeitig auf Null zu setzen, erlauben einige BASIC-Versionen Statements der Form:

```
10 LET B=C=0
```

COMMODORE-64-BASIC würde das zweite Gleichheitszeichen als logischen Operator interpretieren. Falls dann C=0 wäre, würde B=−1. Schreiben Sie statt dessen zwei Befehle:

```
10 B=0 : C=0
```

Mehrfache Anweisungen

Einige BASIC-Versionen benutzen den Schrägstrich rückwärts (\) um mehrere Statements in einer Zeile voneinander zu trennen. In COMMODORE-64-BASIC werden alle Anweisungen durch einen Doppelpunkt (:) voneinander getrennt.

MAT-Funktionen

Programme, die die in einigen BASIC-Versionen vorrätigen MAT-Funktionen für Matrizenoperationen verwenden, müssen umgeschrieben werden, indem diese Funktionen mit Hilfe von FOR . . . NEXT-Schleifen nachgebildet werden.

ANHANG K

FEHLERMELDUNGEN

Dieser Anhang enthält eine vollständige Liste der Fehlermeldungen des COMMODORE 64 zusammen mit einer Beschreibung der Ursachen.

BAD DATA Von einem File wurden String-Daten gelesen, das Programm erwartete jedoch numerische Daten.

BAD SUBSCRIPT Das Programm versuchte, ein Element des Arrays anzusprechen, dessen Nummer außerhalb des in der DIM-Anweisung vorgegebenen Bereichs liegt.

CAN'T CONTINUE Der Befehl CONT arbeitet nicht, wenn ein Programm nicht vorher mit RUN gestartet war, ein Fehler auftrat oder eine Zeile geändert wurde.

DEVICE NOT PRESENT Das angesprochene E/A-Gerät war nicht verfügbar bei OPEN, CLOSE, CMD, PRINT#, INPUT# oder GET#.

DIVISION BY ZERO Division durch Null ist mathematisch undefiniert und nicht erlaubt.

EXTRA IGNORED Nach Aufforderung durch INPUT wurden zu viele Daten eingegeben. Nur die ersten wurden berücksichtigt.

FILE NOT FOUND Suchen Sie ein File auf Band, dann wurde eine END-OF-TAPE-Markierung gefunden. Suchen Sie ein File auf der Diskette, dann existiert ein File dieses Namens nicht.

FILE NOT OPEN Das mit CMD, PRINT#, INPUT#, GET# angesprochene File muß zuerst mit OPEN geöffnet werden.

FILE OPEN Sie versuchten ein File zu öffnen und benutzten dazu eine logische Filenummer, die bereits vergeben war.

FORMULA TOO COMPLEX Der Stringausdruck sollte in wenigstens zwei Teile aufgespalten werden, damit das System ihn bearbeiten kann.

ILLEGAL DIRECT INPUT kann nur innerhalb eines Programms benutzt werden und nicht im Direktmodus.

ILLEGAL QUANTITY Eine Zahl, die als Argument einer Funktion oder einer Anweisung benutzt wurde, liegt außerhalb des erlaubten Bereichs.

LOAD Es gibt ein Problem mit dem Programm auf der Kassette.

NEXT WITHOUT FOR Entweder wurden einige Schleifen nicht korrekt verschachtelt oder eine bei NEXT angegebene Variable entspricht nicht der bei FOR verwendeten.

NOT INPUT FILE Es wurde versucht, mit INPUT# oder GET# Daten von einem File zu lesen, das nur zur Ausgabe bestimmt ist.

NOT OUTPUT FILE Sie versuchten Daten durch PRINT# an ein File zu senden, das nur zum Lesen geöffnet wurde.

OUT OF DATA Eine READ-Anweisung wurde ausgeführt, es gibt aber keine Daten in einer DATA-Zeile, die noch nicht mit READ gelesen wurden.

OUT OF MEMORY Es ist kein RAM-Bereich mehr für Programm oder Variablen verfügbar. Dieser Fehler kann auch auftreten, wenn zu viele FOR . . . NEXT-Schleifen oder Unterprogramme ineinander geschachtelt oder zu viele Klammern geöffnet wurden.

OVERFLOW Das Ergebnis einer Rechnung ist größer als die größte erlaubte Zahl (1.70141183 E + 38).

REDIM'D ARRAY Ein Array kann nur einmal DIMensioniert werden. Wird eine Array-Variable aufgerufen, bevor sie DIMensioniert wurde, führt der Rechner eine automatische DIM-Operation aus, wobei die Dimension auf zehn gesetzt wird. Jede folgende DIM-Anweisung wird dann diesen Fehler verursachen.

REDO FROM START String-Zeichen wurden eingegeben, während ein INPUT-Statement numerische Eingabe erwartete. Tippen Sie einfach die korrekten Eingaben noch einmal, und das Programm wird von selbst fortfahren.

RETURN WITHOUT GOSUB Eine RETURN-Anweisung wurde entdeckt, aber kein GOSUB-Befehl wurde vorher gegeben.

STRING TOO LONG Ein String kann höchstens 255 Zeichen enthalten.

SYNTAX Eine Anweisung kann vom COMMODORE 64 nicht erkannt werden. Sie haben eine Klammer vergessen oder zuviel angegeben, ein Schlüsselwort falsch eingetippt usw.

TYPE MISMATCH Dieser Fehler tritt auf, wenn Sie eine Zahl statt eines Strings verwenden und umgekehrt.

UNDEF'D FUNCTION Sie nehmen Bezug auf eine selbst definierte Funktion, die noch nicht im DEF FN angelegt wurde oder deren Definitions-Zeile vom Programm noch nicht durchlaufen wurde.

UNDEF'D STATEMENT Eine nicht existente Zeilennummer wurde mit GOTO, GOSUB oder RUN angesprochen.

VERIFY Das Programm auf Band oder Diskette stimmt nicht mit dem Programm im Speicher überein.

ANHANG L

DATENBLATT MIKROPROZESSOR 6510

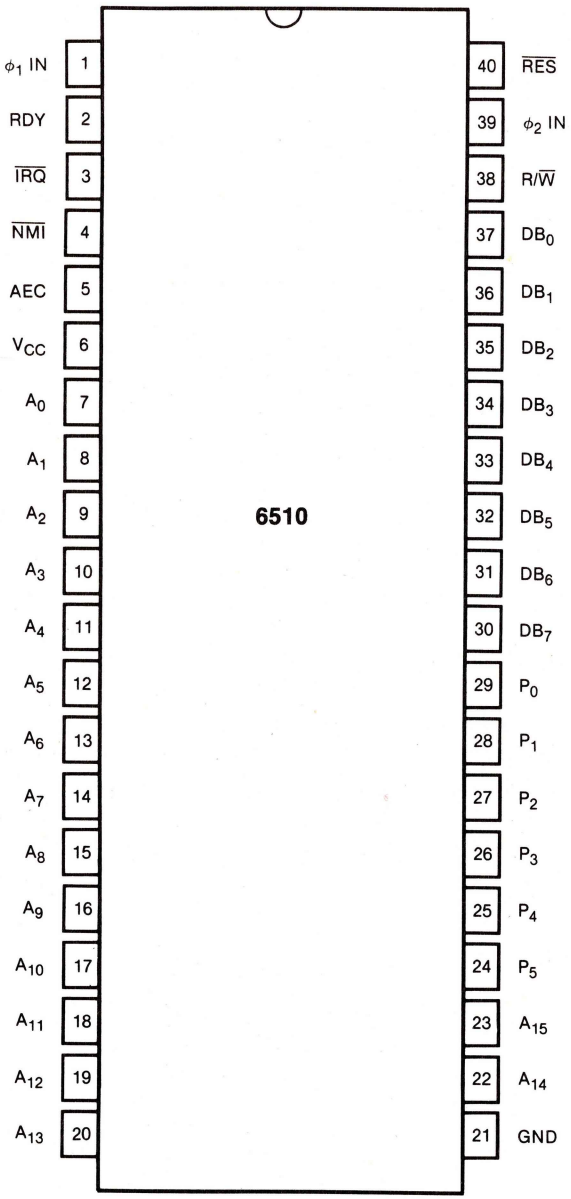
BESCHREIBUNG

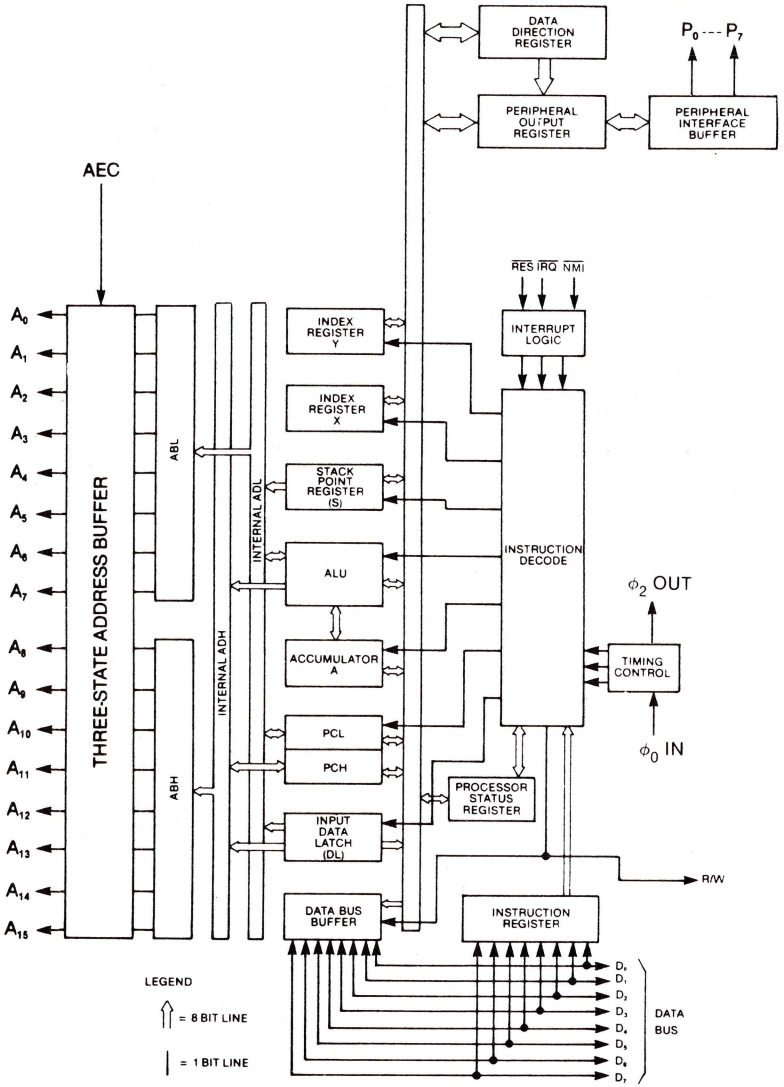
Die 6510-Familie ist ein Mikrocomputersystem, das imstande ist, viele Probleme aus dem Bereich der Mikrocomputer und Peripheriegeräte mit minimalen Kosten zu lösen. Ein 8-Bit bidirektionaler I/O-Port befindet sich "on chip", dessen Ausgaberegister unter Adresse 0000 und dessen Daten-Richtungsregister bei Adresse 0001 erreichbar sind. Das I/O-Port ist Bit für Bit programmierbar.
Der Tri-state 16-Bit Adreßbus ermöglicht auf einfache Weise DMA und den Zugriff mehrerer CPU auf ein und denselben Speicher.
Die interne Prozessorarchitektur ist identisch mit der des MOS TECHNOLOGY 6502, um die Software kompatibel zu machen.

BESONDERHEITEN DES 6510:

- 8-Bit bidirektionaler I/O-Port
- 5 Volt Versorgungsspannung
- 8-Bit Datenlänge
- 56 Befehle
- Dezimale und binäre Arithmetik
- 13 Adressierungsarten
- Absolute indirekte Adressierung
- Programmierbarer Stackzeiger
- Variable Stacklänge
- Interruptmöglichkeiten
- 65K Bytes adressierbar
- Direkt Memory Access (DMA)
- Bus-compatibel zum M6800
- "Pipeline" Architektur
- 1-MHz und 2-MHz Takt

PIN-ANORDNUNG





6510 BLOCK-DIAGRAMM

MERKMALE DES 6510

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{CC}	−0.3 to +7.0	V_{DC}
INPUT VOLTAGE	V_{in}	−0.3 to +7.0	V_{DC}
OPERATING TEMPERATURE	T_A	0 to +70	°C
STORAGE TEMPERATURE	T_{STG}	−55 to +150	°C

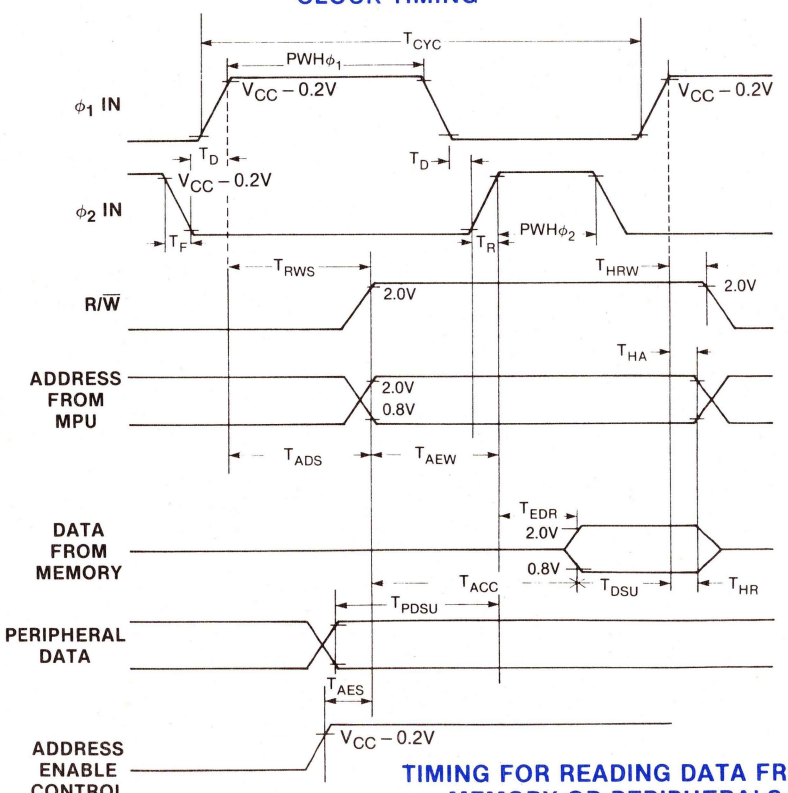
Anmerkung: Dieses Gerät ist gegen Schäden durch Hochspannung oder elektrische Felder geschützt; Spannungen über dem angegebenen max. Nennwert sollten jedoch nicht angelegt werden.

ELEKTRISCHE EIGENSCHAFTEN
(V_{CC} = 5,0 V +/− 5%, V_{SS} = 0, T_A = 0° bis +70° C)

CHARACTERISTIC	SYM-BOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage ϕ_1, $\phi_{2(in)}$	V_{IH}	V_{CC} − 0.2	—	V_{CC} + 1.0V	V_{DC}
Input High Voltage $\overline{RES}$, P_0-P_7 $\overline{IRQ}$, Data		V_{SS} + 2.0	—	—	V_{DC}
Input Low Voltage ϕ_1, $\phi_{2(in)}$	V_{IL}	V_{SS} − 0.3	—	V_{SS} + 0.2	V_{DC}
$\overline{RES}$, P_0-P_7 $\overline{IRQ}$, Data		—	—	V_{SS} + 0.8	V_{DC}
Input Leakage Current (V_{in} = 0 to 5.25V, V_{CC} = 5.25V) Logic	I_{in}	—	—	2.5	μA
ϕ_1, $\phi_{2(in)}$		—	—	100	μA
Three State (Off State) Input Current (V_{in} = 0.4 to 2.4V, V_{CC} = 5.25V) Data Lines	I_{TSI}	—	—	10	μA
Output High Voltage (I_{OH} = −100μA_{DC}, V_{CC} = 4.75V) Data, A0-A15, R/W, P_0-P_7	V_{OH}	V_{SS} + 2.4	—	—	V_{DC}

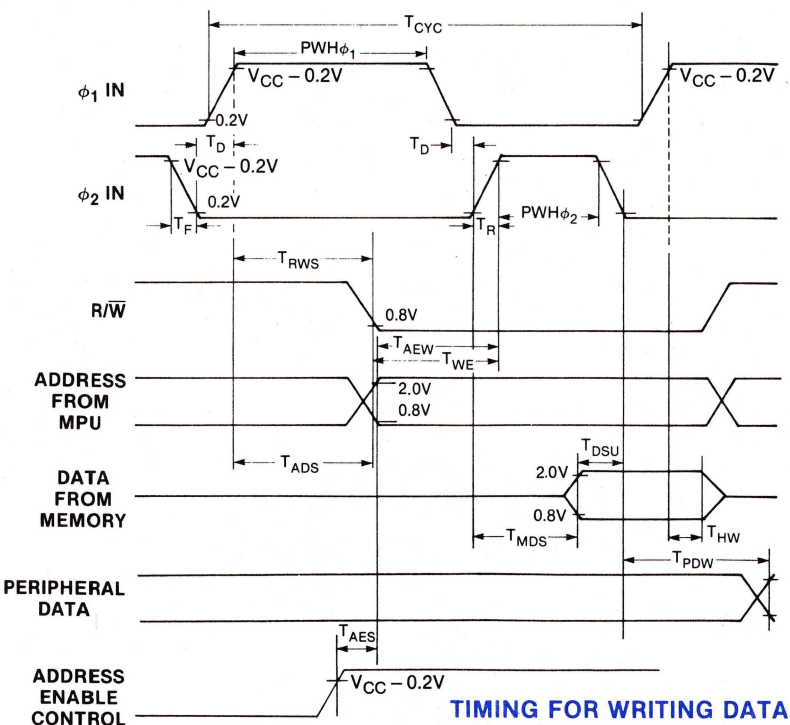
CHARACTERISTIC	SYM-BOL	MIN.	TYP.	MAX.	UNIT
Out Low Voltage (I_{OL} = 1.6mA$_{DC}$, V_{CC} = 4.75V) Data, A0-A15, R/W, P_0-P_7	V_{OL}	—	—	V_{SS} + 0.4	V_{DC}
Power Supply Current	I_{CC}	—	125		mA
Capacitance (V_{in} = 0, T_A = 25°C, f = 1MHz)	C				pF
Logic, P_0-P_7	C_{in}	—	—	10	
Data		—	—	15	
A0-A15, R/W	C_{out}	—	—	12	
ϕ_1	$C\phi_1$	—	30	50	
ϕ_2	$C\phi_2$	—	50	80	

CLOCK TIMING



TIMING FOR READING DATA FROM MEMORY OR PERIPHERALS

CLOCK TIMING



TIMING FOR WRITING DATA TO MEMORY OR PERIPHERALS

WS-EIGENSCHAFTEN

ELEKTRISCHE EIGENSCHAFTEN (V_{CC} = 5,0 V ± 5%, V_{SS} = 0 V, T_A = 0°–70° C)

CLOCK TIMING		1MHz TIMING			2 MHz TIMING			
CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	UNITS
Cycle Time	T_{CYC}	1000	—	—	500	—	—	ns
Clock Pulse Width $\phi 1$	PWH$\phi 1$	430	—	—	215	—	—	ns
(Measured at V_{CC} − 0.2V) $\phi 2$	PWH$\phi 2$	470	—	—	235	—	—	ns
Fall Time, Rise Time (Measured from 0.2V to V_{CC} − 0.2V)	T_F, T_R	—	—	25	—	—	15	ns
Delay Time between Clocks (Measured at 0.2V)	T_D	0	—	—	0	—	—	ns

READ/WRITE TIMING (LOAD = 1TTL)		1 MHz TIMING			2 MHz TIMING			
CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	UNITS
Read/Write Setup Time from 6508	T_{RWS}	—	100	300	—	100	150	ns
Address Setup Time from 6508	T_{ADS}	—	100	300	—	100	150	ns
Memory Read Access Time	T_{ACC}	—	—	575	—	—	300	ns

Data Stability Time Period	T_{DSU}	100	—	—	50			ns
Data Hold Time-Read	T_{HR}		—	—				ns
Data Hold Time-Write	T_{HW}	10	30	—	10	30		ns
Data Setup Time from 6510	T_{MDS}	—	150	200	—	75	100	ns
Address Hold Time	T_{HA}	10	30	—	10	30		ns
R/W Hold Time	T_{HRW}	10	30	—	10	30		ns
Delay Time, Address valid to $\phi 2$ positive transition	T_{AEW}	180	—	—				ns
Delay Time, $\phi 2$ positive transition to Data valid on bus	T_{EDR}	—	—	395				ns
Delay Time, Data valid to $\phi 2$ negative transition	T_{DSU}	300	—	—				ns
Delay Time, R/W negative transition to $\phi 2$ positive transition	T_{WE}	130	—	—				ns
Delay Time, $\phi 2$ negative transition to Peripheral Data valid	T_{PDW}	—	—	1				μs
Peripheral Data Setup Time	T_{PDSU}	300	—	—				ns
Address Enable Setup Time	T_{AES}			60			60	ns

SIGNALBESCHREIBUNG

TAKT ($\emptyset_1$, $\emptyset_2$)

Die CPU 6510 benötigt einen sich nicht überlappenden Zweiphasentakt, der den Pegel V_{cc} hat.

ADRESSBUS (A_0–A_{15})

Diese Ausgänge sind TTL-kompatibel (1 Standardeingang + 103 pF).

DATENBUS (D_0–D_7)

Diese 8 Pins übermitteln die Daten von der CPU zum Speicher (oder anderen Bausteinen) und umgekehrt. Es sind Tristate-Buffer, die TTL-Standard + 103 pF treiben können.

RESET (RES):

Dieser Eingang wird benutzt, um den Prozessor nach dem Einschalten zu starten oder (im Betrieb) in einen definierten Zustand zu bringen. Liegt dieser Eingang auf L-Pegel, kann der Prozessor nicht ein- oder ausgeben. Wenn auf diesen Eingang eine positive Flanke geschaltet wird, beginnt der Prozessor mit der Reset-Prozedur. Nach einer Systeminitialisierungszeit von 6 Taktzyklen wird das Interruptflag gesetzt, und der Prozessor lädt den Programmzähler mit dem Inhalt der Adresse $FFFC und $FFFD. Wenn nach dem Einschalten V_{cc} 4,75 V erreicht, muß RESET noch 2 Taktzyklen auf Low gehalten werden. Während dieser Zeit wird R/W gültig. Wenn RESET dann auf H geschaltet wird, beginnt der oben beschriebene Zyklus.

INTERRUPT REQUEST ($\overline{\text{IRQ}}$):

Wenn dieser TTL-Eingang nach Low geschaltet wird, beginnt der Prozessor mit einer Interruptroutine, nachdem er den vor dem IRQ-Befehl gültigen Befehl abgearbeitet hat. Dann wird das Interruptflag im Flagregister geprüft. Falls das Interruptflag nicht gesetzt ist, beginnt der Prozessor mit der Interruptroutine. Der Programmzähler und das Flagregister werden im Stack gespeichert.
Dann setzt der Prozessor das Interruptflag, damit derselbe Interrupt nicht noch einmal bearbeitet wird. Am Ende dieses Ablaufs wird der Programmzähler mit dem Inhalt von Adresse FFFE (Low-Byte) und FFFF (High-Byte) geladen.

ADRESS ENABLE CONTROL (AEC):

Der Adreßbus ist nur gültig, wenn AEC auf High geschaltet ist. Wenn AEC Low geschaltet ist, befinden sich die Adreßausgänge im hochohmigen Zustand (Tri-state). Dies erleichtert den direkten Speicherzugriff (DMA) und ermöglicht Multiprozessorsysteme.

I/O PORT (P_0–P_7)

6 Pins werden als Port genutzt, durch den Daten direkt an Peripheriegeräte geschickt werden können. Das Datenregister ist im RAM unter Adresse 0000 erreichbar. Die Ausgänge können einen Standard-TTL-Eingang und 130 pF treiben.

READ/WRITE (R/W):

Der Ausgang liegt immer auf High, er ist nur Low, wenn der Prozessor Daten in den Speicher oder in einen Peripheriebaustein schreiben will.

ADRESSIERARTEN

IMPLIZIERTE ADRESSIERUNG:

Dies ist ein 1-Byte-Befehl, der eine Operation in der CPU bewirkt.

UNMITTELBARE ADRESSIERUNG:

Das auf den Befehl folgende Byte ist der Operand, es wird keine weitere Adresse benötigt.

ABSOLUTE ADRESSIERUNG:

Hier stellt das 2. Byte das Low-Byte, das 3. Byte das High-Byte der resultierenden Adresse dar. Diese Art der Adressierung erlaubt den Zugriff auf den gesamten Speicher von 64 KB.

ZERO-PAGE-ADRESSIERUNG:

Diese Art erlaubt eine kürzere Codierung und damit eine schnellere Ausführung. Das 2. Byte des Befehls stellt das Low-Byte dar, das High-Byte wird als 0000 angenommen.

INDIZIERTE ZERO-PAGE-ADRESSIERUNG:

Diese Adressierungsart benutzt die Indexregister. Die resultierende Adresse wird errechnet, indem das 2. Byte des Befehls zum Indexregister X oder Y addiert wird. Dies ist eine Zero-Page-Adresse, es wird kein Übertrag zum High-Byte addiert, Pages werden also nicht überschritten.

INDIZIERTE ABSOLUTE ADRESSIERUNG:

Diese Art wird im Zusammenhang mit den Indexregistern X oder Y benutzt. Die resultierende Adresse wird durch Addition des Inhaltes der Register X oder Y zum 2. Byte des Befehls gebildet. Wenn nötig, wird ein Übertrag zum 3. Byte, dem High-Byte, gebildet. Diese Art der Adressierung ermöglicht es, jede Speicherstelle zu erreichen und durch Indexregisteroperationen beliebige Datenfelder mit einer Basisadresse zu erreichen.

RELATIVE ADRESSIERUNG:

Diese Art wird nur im Zusammenhang mit Verzweigungsbefehlen gebraucht und bestimmt die Zieladresse des Sprunges. Das 2. Byte des Befehls ist ein Offset, der zu dem Low-Byte des Programmzählers addiert wird, wenn der Sprung ausgeführt werden soll. Die Zieladresse kann 128 Stellen niedriger oder 127 Stellen höher als die Adresse des nächsten Befehls sein.

INDIZIERT-INDIREKTE ADRESSIERUNG:

Hier wird das 2. Byte des Befehls zum Inhalt des X-Indexregisters ohne Carry addiert. Das Ergebnis bestimmt eine Speicherstelle in der Zero-Page, wo sich das Low-Byte der resultierenden Adresse befindet. Das High-Byte befindet sich in der nächsten Speicherstelle. Beide Speicherstellen müssen sich in der Zero-Page befinden.

INDIREKT-INDIZIERTE ADRESSIERUNG:

Hier bestimmt das 2. Byte des Befehls eine Speicherstelle in der Zero-Page.
Der Inhalt wird zum Inhalt des Y-Indexregisters addiert, das Ergebnis ist das Low-Byte der resultierenden Adresse. Der Übertrag dieser Addition wird zum Inhalt der nächsten Speicherstelle addiert, das Ergebnis ist das High-Byte der echten Adresse.

ABSOLUT-INDIREKTE ADRESSIERUNG:

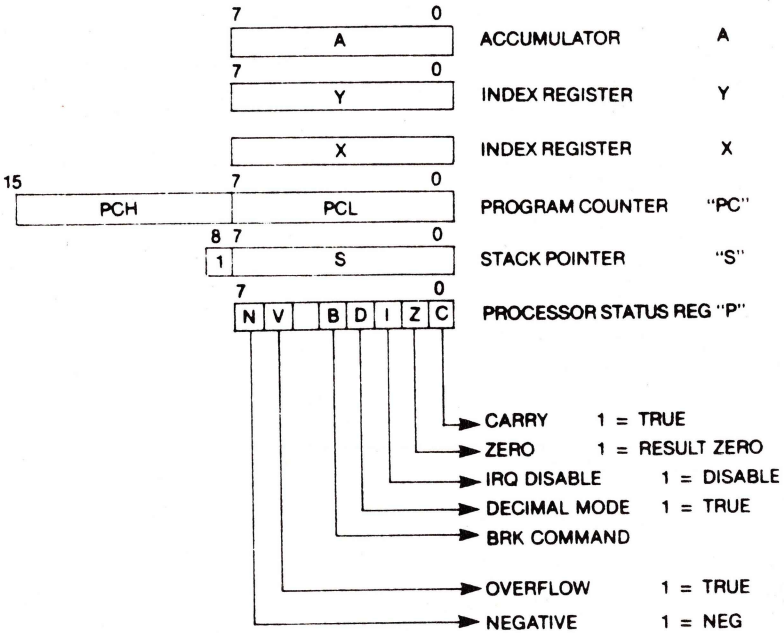
Das 2. Byte des Befehls ist das Low-Byte, das 3. Byte das High-Byte einer Speicheradresse, in der sich das Low-Byte der echten Adresse befindet. Das High-Byte der echten Adresse befindet sich auf dem nächsten Speicherplatz. Diese Adresse wird in den Programmzähler geladen.

ANWEISUNGSSATZ – ALPHABETISCHE REIHENFOLGE

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-OR" Memory with Accumulator

INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right (Memory or Accumulator)
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Register
TYA	Transfer Index Y to Accumulator

PROGRAMMIERBEISPIEL



INSTRUCTIONS		Immediate			Absolute			Zero Page			Accum.			Implied			(Ind.) X			(Ind.) Y			Z, Page, X			Abs. X			Abs. Y			Relative			Indirect			Z, Page, Y			CONDITION CODES						
Mnemonic	Operation	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
ADC	A + M + C → A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										✓	✓	✓	—	—	✓
AND	A∧M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										✓	✓	—	—	—	—
ASL	C ← [7 … 0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													✓	✓	✓	—	—	—
BCC	BRANCH ON C = 0 (2)																															90	2	2							—	—	—	—	—	—
BCS	BRANCH ON C = 1 (2)																															B0	2	2							—	—	—	—	—	—
BEQ	BRANCH ON Z = 1 (2)																															F0	2	2							—	—	—	—	—	—
BIT	A∧M				2C	4	3	24	3	2																															M_7	✓	—	—	M_6
BMI	BRANCH ON N = 1 (2)																															30	2	2							—	—	—	—	—	—
BNE	BRANCH ON Z = 0 (2)																															D0	2	2							—	—	—	—	—	—
BPL	BRANCH ON N = 0 (2)																															10	2	2							—	—	—	—	—	—
BRK	(See Fig. 1)													00	7	1																									—	—	—	1	—	—
BVC	BRANCH ON V = 0 (2)																															50	2	2							—	—	—	—	—	—
BVS	BRANCH ON V = 1 (2)																															70	2	2							—	—	—	—	—	—
CLC	0 → C													18	2	1																									—	—	0	—	—	—
CLD	0 → D													D8	2	1																									—	—	—	—	0	—
CLI	0 → I													58	2	1																									—	—	—	0	—	—
CLV	0 → V													B8	2	1																									—	—	—	—	—	0
CMP	A − M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										✓	✓	✓	—	—	—
CPX	X − M	E0	2	2	EC	4	3	E4	3	2																															✓	✓	✓	—	—	—
CPY	Y − M	C0	2	2	CC	4	3	C4	3	2																															✓	✓	✓	—	—	—
DEC	M − 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													✓	✓	—	—	—	—
DEX	X − 1 → X													CA	2	1																									✓	✓	—	—	—	—
DEY	Y − 1 → Y													B8	2	1																									✓	✓	—	—	—	—
EOR	A∨M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										✓	✓	—	—	—	—
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													✓	✓	—	—	—	—
INX	X + 1 → X													E8	2	1																									✓	✓	—	—	—	—
INY	Y + 1 → Y													C8	2	1																									✓	✓	—	—	—	—
JMP	JUMP TO NEW LOC.				4C	3	3																													6C	5	3				—	—	—	—	—	—
JSR	(See Fig. 2) JUMP SUB				20	6	3																																			—	—	—	—	—	—
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										✓	✓	—	—	—	—

INSTRUCTIONS		Immediate			Absolute			Zero Page			Accum.			Implied			(Ind.) X			(Ind.) Y			Z, Page, X			Abs. X			Abs. Y			Relative			Indirect			Z, Page, Y			CONDITION CODES						
Mnemonic	Operation	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
LDX	M→X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	↙	↙	—	—	—	—
LDY	M→Y (1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3												↙	↙	—	—	—	—
LSR	0→[7 0]→C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3												0	↙	↙	—	—	—
NOP	NO OPERATION													EA	2	1																								—	—	—	—	—	—
ORA	AVM→A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										↙	↙	—	—	—	—
PHA	A→M_S S − 1→S													48	3	1																								—	—	—	—	—	—
PHP	P→M_S S − 1→S													08	3	1																								—	—	—	—	—	—
PLA	S + 1→S M_S→A													68	4	1																								↙	↙	—	—	—	—
PLP	S + 1→S M_S→P													28	4	1																								(RESTORED)					
ROL	[7 0]←[C]				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3												↙	↙	↙	—	—	—
ROR	[C]→[7 0]				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3												↙	↙	↙	—	—	—
RTI	(See Fig. 1) RTRN INT													40	6	1																								(RESTORED)					
RTS	(See Fig. 2) RTRN SUB													60	6	1																								—	—	—	—	—	—
SBC	A − M − $\overline{C}$→A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										↙	↙	(3)	—	—	↙
SEC	1→C													38	2	1																								—	—	1	—	—	—
SED	1→D													F8	2	1																								—	—	—	—	1	—
SEI	1→I													78	2	1																								—	—	—	1	—	—
STA	A→M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										—	—	—	—	—	—
STX	X→M				8E	4	3	86	3	2																														96	4	2	—	—	—	—	—	—
STY	Y→M				8C	4	3	84	3	2													94	4	2																—	—	—	—	—	—
TAX	A→X													AA	2	1																								↙	↙	—	—	—	—
TAY	A→Y													A8	2	1																								↙	↙	—	—	—	—
TSX	S→X													BA	2	1																								↙	↙	—	—	—	—
TXA	X→A													8A	2	1																								↙	↙	—	—	—	—
TXS	X→S													9A	2	1																								—	—	—	—	—	—
TYA	Y→A													98	2	1																								↙	↙	—	—	—	—

(1) ADD 1 TO "N" IF PAGE BOUNDARY IS CROSSED.
(2) ADD 1 TO "N" IF BRANCH OCCURS TO SAME PAGE.
ADD 2 TO "N" IF BRANCH OCCURS TO DIFFERENT PAGE.
(3) CARRY NOT = BORROW.
(4) IF IN DECIMAL MODE Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT.

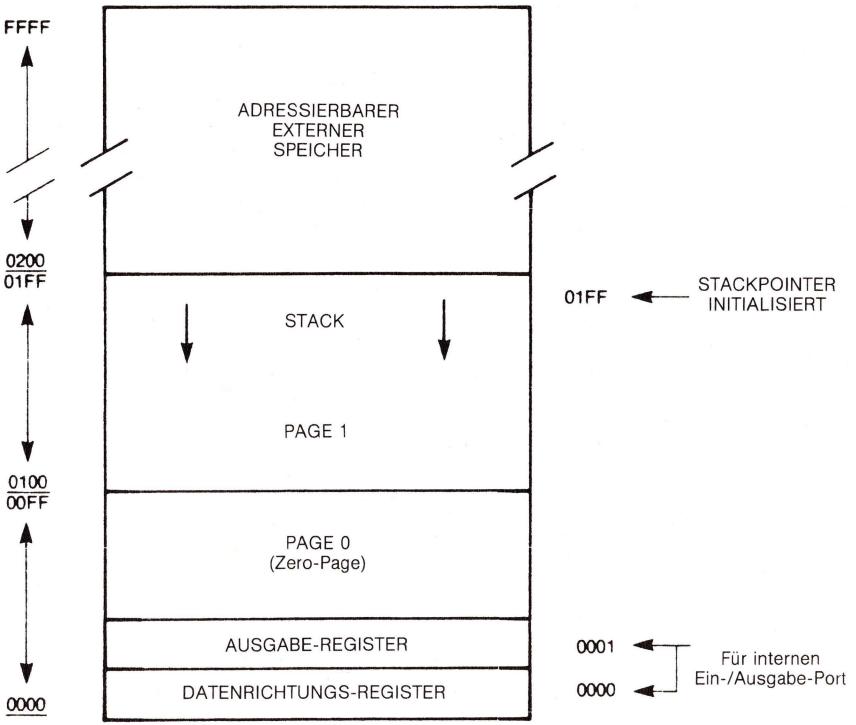
X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M_S MEMORY PER STACK POINTER

\+ ADD
− SUBTRACT
∧ AND
V OR
∀ EXCLUSIVE OR

↙ MODIFIED
— NOT MODIFIED
M_7 MEMORY BIT 7
M_6 MEMORY BIT 6
N NO. CYCLES
NO. BYTES

NOTE: COMMODORE SEMICONDUCTOR GROUP cannot assume liability for the use of undefined OP CODES.

6510 SPEICHER-KONFIGURATION



ANMERKUNGEN:

Dadurch, daß das Datenregister des I/O-Ports in der Zero-Page liegt, werden die vorteilhaften Zero-Page-Adressierungsarten noch verstärkt.
Indem die I/O-Pins durch das Datenrichtungsregister als Eingänge geschaltet werden, hat der Benutzer die Möglichkeit, den Inhalt des Speicherplatzes 0001 durch Peripheriegeräte zu verändern. Diese Möglichkeit im Zusammenhang mit den Adressierungsbefehlen für die Zero-Page erschließt neue und ungewöhnlich vielseitige Programmierungstechniken, die es noch nicht gab.

ACHTUNG:

Der Baustein ist gegen statische Aufladung geschützt, trotzdem sollten Vorkehrungen getroffen werden, damit die Grenzwerte nicht überschritten werden.

ANHANG M

6526 COMPLEX INTERFACE ADAPTER (CIA)

BESCHREIBUNG

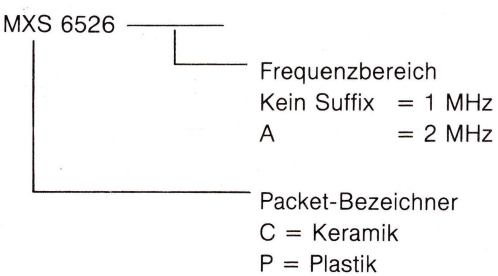
Der Baustein 6526 ist ein Interface-Adapter, mit dem 65XX-Bus kompatibel, mit flexiblem Timing und diversen Ein-/Ausgabemöglichkeiten.

BESONDERHEITEN

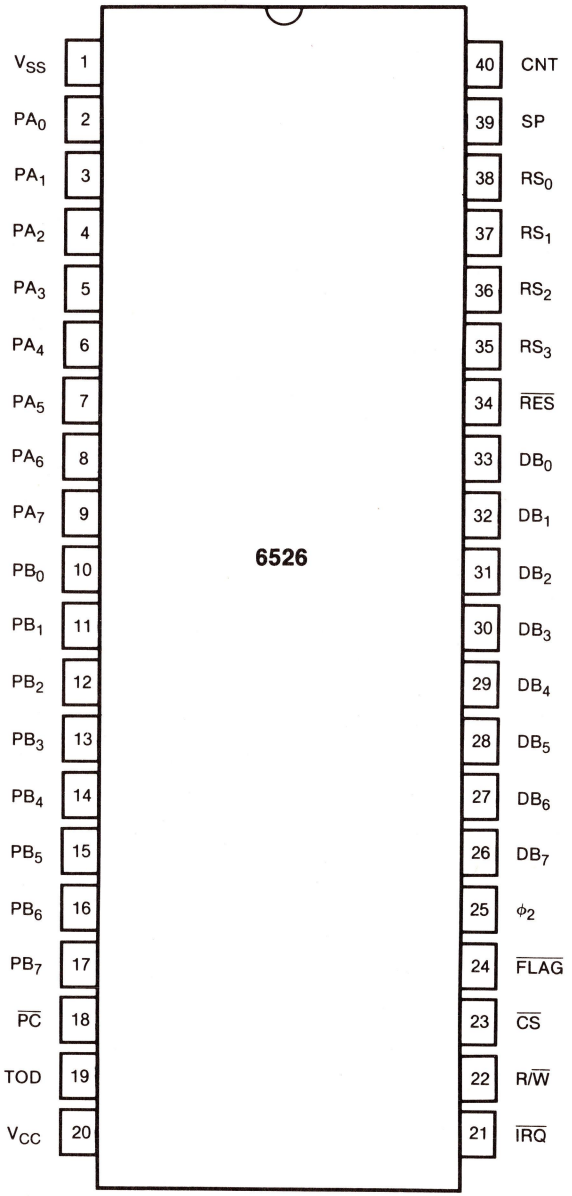
- 16 einzeln programmierbare Ein-/Ausgabeleitungen
- 8- oder 16-Bit-Datentransport mit Handshaking-Betrieb beim Lesen oder Schreiben
- 2 unabhängige, verknüpfbare 16-Bit-Intervalltimer
- 24-Stunden-(AM/PM)-Zeituhr mit programmierbarem Alarm
- 8-Bit-Schieberegister für serielle Ein-/Ausgabe
- 2 TTL-Eingänge können gespeist werden
- CMOS-kompatibel
- 1- oder 2-MHz-Takt

BESTELLUNGSHINWEISE

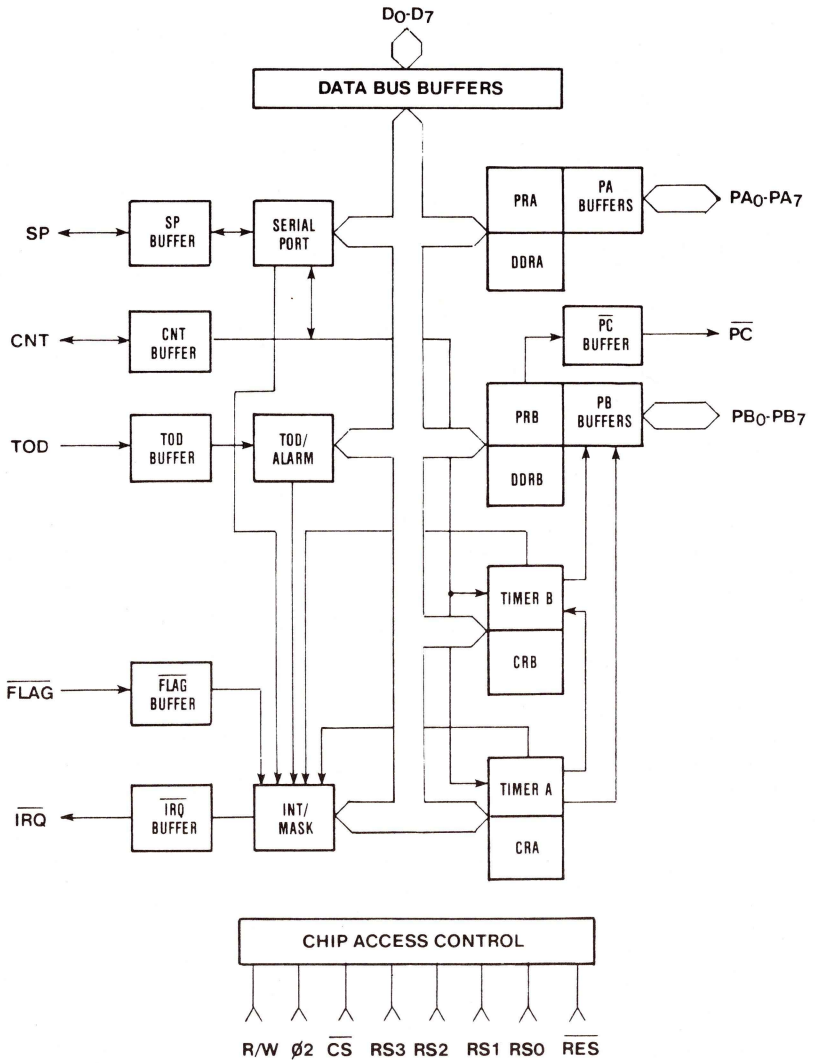
MXS 6526



PIN-ANORDNUNG



6526
BLOCKSCHALTBILD



MAX. NENNWERTE

Versorgungsspannung V_{CC}	−0,3 V bis +7,0 V
Ein-/Ausgangsspannung V_{IN}	−0,3 V bis +7,0 V
Betriebstemperatur T_{OP}	0° C bis 70° C
Lagertemperatur T_{STG}	−55° C bis 150° C

Alle Eingänge haben Schutzschaltungen, um Schäden durch hohe statische Entladungen zu vermeiden. Spannungen überhalb der zulässigen Grenzwerte sollten jedoch nur wenn unbedingt notwendig und mit äußerster Vorsicht angelegt werden.

KOMMENTAR

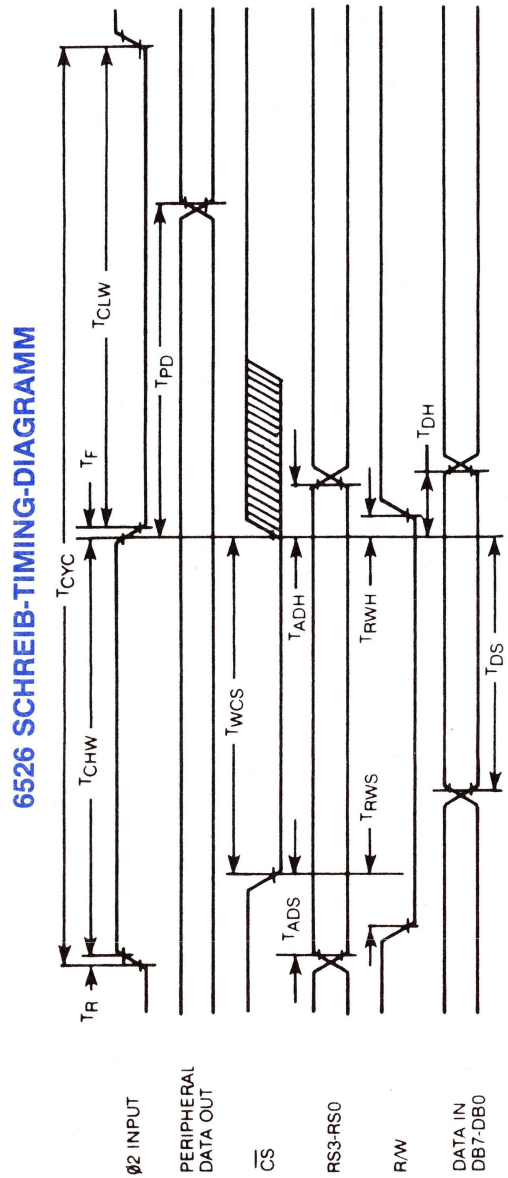
Spannungen, die über den angegebenen max. Nennwerten liegen, können zu Schäden oder Beschädigungen des Geräts führen. Unter "max. Nennwerte" sind nur Spannungswerte aufgeführt. Wird das Gerät mit höheren als angegebenen Spannungen betrieben oder werden über einen längeren Zeitraum die max. Nennwerte gewählt, kann dies die Gerätezuverlässigkeit beeinträchtigen.

ELEKTRISCHE EIGENSCHAFTEN ($V_{CC} \pm 5\%$, $V_{SS} = 0$ V, $T_A = 0$–70° C)

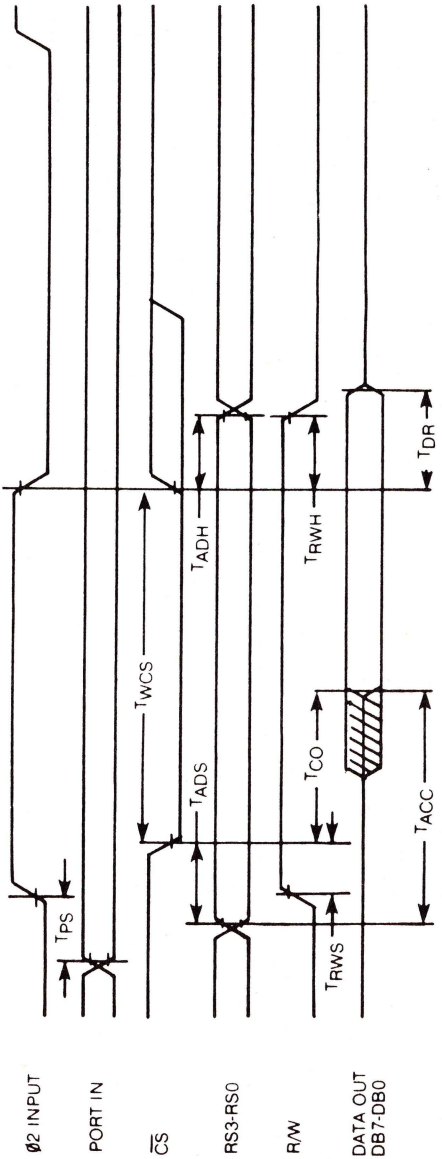
CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage	V_{IH}	+2.4	—	V_{CC}	V
Input Low Voltage	V_{IL}	−0.3	—	—	V
Input Leakage Current; $V_{IN}=V_{SS}$ +5V ($\overline{TOD}$, R/W, $\overline{FLAG}$, $\phi 2$, $\overline{RES}$, RS0-RS3, $\overline{CS}$)	I_{IN}	—	1.0	2.5	μA

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Port Input Pull-up Resistance	R_{PI}	3.1	5.0	—	KΩ
Output Leakage Current for High Impedance State (Three State); V_{IN} = 4V to 2.4V; (DB0–DB7, SP, CNT, $\overline{IRQ}$)	I_{TSI}	—	±1.0	±10.0	μA
Output High Voltage V_{CC}=MIN, I_{LOAD} < −200μA (PA0–PA7, $\overline{PC}$ PB0–PB7, DB0–DB7)	V_{OH}	+2.4	—	V_{CC}	V
Output Low Voltage V_{CC}= MIN, I_{LOAD} < 3.2 mA	V_{OL}	—	—	+0.40	V
Output High Current (Sourcing); V_{OH} > 2.4V (PA0–PA7, PB0–PB7, PC, DB0–DB7	I_{OH}	−200	−1000	—	μA
Output Low Current (Sinking); V_{OL} < .4V (PA0–PA7, $\overline{PC}$, PB0–PB7, DB0–DB7)	I_{OL}	3.2	—	—	mA
Input Capacitance	C_{IN}	—	7	10	pf
Output Capacitance	C_{OUT}	—	7	10	pf
Power Supply Current	I_{CC}	—	70	100	mA

6526 SCHREIB-TIMING-DIAGRAMM



6526 LESE-TIMING-DIAGRAMM



6526 PINBELEGUNG

Ø2-TAKTEINGANG

TTL-kompatibler Takteingang zur Steuerung der internen Funktionen und ein Timing-Bezug für die Kommunikation mit dem Systemdatenbus.

$\overline{\text{CS}}$-CHIP SELECT

Der Baustein reagiert nur dann auf die Steuereingänge RS und R/W, wenn dieser Eingang auf Low und der Takt auf High liegt.

R/W READ/WRITE

Das R/W-Signal wird normalerweise vom Prozessor erzeugt und kontrolliert die Richtung des Datentransportes. R/W = High bedeutet, daß die Daten aus dem 6526 gelesen werden können, bei Low können Daten hineingeschrieben werden.

RS3-RS0-Adreßeingänge

Damit werden die internen Register angesprochen (siehe Registerbelegung Seite 420).

DB7-BD0-Datenbus, Ein-/Ausgänge

Diese Pins verbinden den Chip mit dem Systemdatenbus und sind hochohmig, außer wenn $\overline{\text{CS}}$ Low, R/W und 02 High liegen, um Daten aus dem 6526 zu lesen. Dann sind die Datenausgangsbuffer aktiviert und übertragen die Daten vom ausgewählten Register an den Bus.

$\overline{\text{IRQ}}$ INTERRUPT REQUEST AUSGANG

Dies ist ein "open-drain"-Ausgang, der normalerweise mit dem Interrupteingang des Prozessors verbunden ist. Durch den externen Pullup-Widerstand ist es möglich, mehrere $\overline{\text{IRQ}}$-Ausgänge miteinander zu verbinden. Wie dieser Ausgang aktiviert (auf Low gezogen) werden kann, wird im Folgenden noch beschrieben.

$\overline{\text{RES}}$ RESET-EINGANG

Wenn dieser Eingang Low-Pegel hat, werden alle internen Register gelöscht. Die Ports werden als Eingänge und die Portregister auf Null geschaltet (obwohl die Ports durch die Pullup-Widerstände als High gelesen werden würden). Die Intervalltimerregister werden auf Null und die Latches auf Eins gesetzt. Alle anderen Register werden auf Null gesetzt.

6526 TIMING-CHARAKTERISTIKEN

Symbol	Characteristic	1MHz MIN	1MHz MAX	2MHz MIN	2MHz MAX	Unit
	ϕ2 Clock					
T_{CYC}	Cycle Time	1000	20,000	500	20,000	ns
T_R, T_F	Rise and Fall Time	—	25	—	25	ns
T_{CHW}	Clock Pulse Width (High)	420	10,000	200	10,000	ns
T_{CLW}	Clock Pulse Width (Low)	420	10,000	200	10,000	ns
	Write Cycle					
T_{PD}	Output Delay From ϕ2	—	1000	—	500	ns
T_{WCS}	$\overline{\text{CS}}$ low while ϕ2 high	420	—	200	—	ns
T_{ADS}	Address Setup Time	0	—	0	—	ns
T_{ADH}	Address Hold Time	10	—	5	—	ns
T_{RWS}	R/W Setup Time	0	—	0	—	ns
T_{RWH}	R/W Hold Time	0	—	0	—	ns
T_{DS}	Data Bus Setup Time	150	—	75	—	ns
T_{DH}	Data Bus Hold Time	0	—	0	—	ns
	Read Cycle					
T_{PS}	Port Setup Time	300	—	150	—	ns
T_{WCS}(2)	$\overline{\text{CS}}$ low while ϕ2 high	420	—	20	—	ns
T_{ADS}	Address Setup Time	0	—	0	—	ns
T_{ADH}	Address Hold Time	10	—	5	—	ns
T_{RWS}	R/W Setup Time	0	—	0	—	ns
T_{RWH}	R/W Hold Time	0	—	0	—	ns

Symbol	Characteristic	1MHz		2MHz		Unit
		MIN	MAX	MIN	MAX	
T_{ACC}	Data Access from RS3-RS0	—	550	—	275	ns
T_{CO}(3)	Data Access from $\overline{CS}$	—	320	—	150	ns
T_{DR}	Data Release Time	50	—	25	—	ns

NOTES: 1—All timings are referenced from V_{IL} max and V_{IH} min on inputs and V_{OL} max and V_{OH} min on outputs.

2—T_{WCS} is measured from the later of $\phi 2$ high or $\overline{CS}$ low. $\overline{CS}$ must be low at least until the end of $\phi 2$ high.

3—T_{CO} is measured from the later of $\phi 2$ high or $\overline{CS}$ low. Valid data is available only after the later of T_{ACC} or T_{CO}.

REGISTERBELEGUNG

RS3	RS2	RS1	RS0	REG	NAME	
0	0	0	0	0	PRA	PERIPHERAL DATA REG A
0	0	0	1	1	PRB	PERIPHERAL DATA REG B
0	0	1	0	2	DDRA	DATA DIRECTION REG A
0	0	1	1	3	DDRB	DATA DIRECTION REG B
0	1	0	0	4	TA LO	TIMER A LOW REGISTER
0	1	0	1	5	TA HI	TIMER A HIGH REGISTER
0	1	1	0	6	TB LO	TIMER B LOW REGISTER
0	1	1	1	7	TB HI	TIMER B HIGH REGISTER
1	0	0	0	8	TOD 10THS	10THS OF SECONDS REGISTER
1	0	0	1	9	TOD SEC	SECONDS REGISTER
1	0	1	0	A	TOD MIN	MINUTES REGISTER
1	0	1	1	B	TOD HR	HOURS—AM/PM REGISTER
1	1	0	0	C	SDR	SERIAL DATA REGISTER
1	1	0	1	D	ICR	INTERRUPT CONTROL REGISTER
1	1	1	0	E	CRA	CONTROL REG A
1	1	1	1	F	CRB	CONTROL REG B

FUNKTIONSBESCHREIBUNG

Ein-/Ausgangsports (PRA, PRB, DDRA, DDRB)

Jeder der beiden Ports A und B bestehen aus einem 8-Bit-Datenregister (PRA bzw. PRB) und einem Datenrichtungsregister (DDRA bzw. DDRB). Ist eines der Bits im DDR Eins gesetzt, wird das entsprechende Bit im PR ausgegeben; ist das Bit im DDR Null, wird das entsprechende Bit als Eingang geschaltet. Beim Lesen stellt das PR das am Ausgang (PA0-PA7, PB0-PB7) gültige Bit dar, unabhängig davon, ob der betreffende Pin als Ausgang oder Eingang geschaltet ist. Beide Ports sind sowohl TTL- als auch CMOS-kompatibel (durch aktive und passive Pullup-Elemente) und können zwei TTL-Einheiten treiben. Zusätzlich zur normalen Funktion übernehmen PB6 und PB7 die Funktion eines Intervalltimer-Ausgangs.

Handshaking

Dieses Datenübertragungsverfahren kann durch Benutzung des Ausgangs $\overline{PC}$ und des Eingangs $\overline{FLAG}$ realisiert werden. $\overline{PC}$ wird für einen Taktzyklus Low geschaltet, wenn in PRB ein- oder ausgelesen wurde. Dieses Signal kann also als "data ready"- oder "data accepted"-Signal für PRB benutzt werden. (Bei 16-Bit-Datenübertragungen [mit PRA und PRB] würde also PRA zuerst gelesen werden). Der $\overline{FLAG}$-Eingang reagiert auf negative Flanken. Mit ihm kann das $\overline{PC}$-Signal von einem anderen 6526 empfangen werden, oder er wird als Interrupteingang benutzt. Jede negative Flanke an $\overline{FLAG}$ setzt das Flag-Interrupt-Bit 4.

REG	NAME	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	PRA	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
1	PRB	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
2	DDRA	DPA_7	DPA_6	DPA_5	DPA_4	DPA_3	DPA_2	DPA_1	DPA_0
3	DDRB	DPB_7	DPB_6	DPB_5	DPB_4	DPB_3	DPB_2	DPB_1	DPB_0

INTERVALL-TIMER (TIMER A, TIMER B)

Beide bestehen aus je einem 16-Bit-Intervalltimer (nur Lesen) und einem 16-Bit-Latch (nur Schreiben). Beim Schreiben werden die Daten in das Latch geschrieben, während beim Lesen der Inhalt des Intervalltimers angezeigt wird. Die Timer können sowohl unabhängig voneinander als auch zusammen benutzt werden. Die verschiedenen Betriebsarten erlauben Zeitverzögerungen, variable Impulslängen, Impulsfolgen und Signale unterschiedlicher Frequenz.

Mit dem Eingang CNT können die Zähler externe Impulse zählen und Frequenzen, Impulslängen und Verzögerungszeiten messen. Jeder Zähler hat ein eigenes Kontrollregister zur unabhängigen Überwachung folgender Funktionen:

START/STOP

Ein Kontrollbit (cb) ermöglicht dem Prozessor, den Zähler zu jeder Zeit zu starten und zu stoppen.

PB ON/OFF

Ein Kontrollbit steuert die Ausgabe des Zählerüberlaufs an Port B (PB6 für Timer A, PB7 für Timer B). Dieses Bit überschreibt das DDRB-Kontrollbit und schaltet den entsprechenden Pin auf Ausgang.

TOGGLE/PULSE

Ein Kontrollbit bestimmt die Art des Ausgangssignals, das an Port B erscheint. Am Ende jedes Zählerzyklus (underflow) kann der Ausgang entweder von Low nach High und umgekehrt wechseln, oder ein einzelner positiver Impuls (Länge: 1 Taktzyklus ∅2) erzeugt werden. Der Toggle-Ausgang wird auf High gesetzt, wenn der Zähler gestartet wird. Durch $\overline{\text{RES}}$ auf Low gesetzt.

ONE SHOT/CONTINUOUS

Ein Kontrollbit wählt eine der beiden Betriebsarten. Im One-Shot-Modus wird von dem Wert im Latch bis Null gezählt, ein Interrupt erzeugt, der Wert erneut geladen und der Zähler gestoppt. Im Continuous-Modus wird nicht gestoppt, sondern dieser Vorgang kontinuierlich wiederholt.

FORCE LOAD

Dieses Strobe-Bit erzwingt, daß der Inhalt des Latches in den Zeitzähler geladen wird, unabhängig davon, ob der Zähler läuft oder nicht.

INPUT-MODE

Kontrollbits erlauben die Auswahl des Taktes, der zur Dekrementierung des Zählers benutzt wird. Timer A kann Ø2-Taktimpulse oder externe Impulse über CNT zählen. Timer B kann zusätzlich "underflow"-Impulse des Timers A zählen. Zusätzlich besteht eine Steuermöglichkeit über den Pin CNT. Der Inhalt des Latches wird bei jedem Zählerunterlauf, "force load" oder nach einem Schreiben des H-Byte des Latches (bei angehaltenem Zähler) in den Zähler übernommen. Wenn der Zähler läuft, bewirkt das Schreiben des H-Bytes nur ein Laden des Latches, kein Laden des Zählers.

LESEN (TIMER)

REG	NAME								
4	TA LO	TAL_7	TAL_6	TAL_5	TAL_4	TAL_3	TAL_2	TAL_1	TAL_0
5	TA HI	TAH_7	TAH_6	TAH_5	TAH_4	TAH_3	TAH_2	TAH_1	TAH_0
6	TB LO	TBL_7	TBL_6	TBL_5	TBL_4	TBL_3	TBL_2	TBL_1	TBL_0
7	TB HI	TBH_7	TBH_6	TBH_5	TBH_4	TBH_3	TBH_2	TBH_1	TBH_0

SCHREIBEN (VORTEILER)

REG	NAME								
4	TA LO	PAL_7	PAL_6	PAL_5	PAL_4	PAL_3	PAL_2	PAL_1	PAL_0
5	TA HI	PAH_7	PAH_6	PAH_5	PAH_4	PAH_3	PAH_2	PAH_1	PAH_0
6	TB LO	PBL_7	PBL_6	PBL_5	PBL_4	PBL_3	PBL_2	PBL_1	PBL_0
7	TB HI	PBH_7	PBH_6	PBH_5	PBH_4	PBH_3	PBH_2	PBH_1	PBH_0

UHRZEIT (TOD – Time Of Day)

Die TOD-Clock ist ein spezieller Zähler für Echtzeitanwendungen. Sie besteht aus einer 24-Stunden-Uhr mit einer Auflösung von 1/10-Sekunden. Sie ist in 4 Register aufgeteilt:
1/10-Sekunden,
Sekunden,
Minuten,
Stunden.
Das AM/PM-Flag ist das MSB des Stundenregisters, um das Lesen zu vereinfachen. Jedes Register wird im BCD-Code gelesen, damit die Konvertierung für das

Betreiben von Anzeigegeräten vereinfacht wird. Die Uhr benötigt einen Takt mit 50 Hz oder 60 Hz (programmierbar) mit TTL-Pegel. Ein programmierbarer ALARM ist dafür vorgesehen, einen Interrupt zu einer bestimmten Zeit auszulösen. Die zugehörigen Register liegen auf den gleichen Adressen wie die Register der Uhr, der Zugriff auf die ALARM-Register erfolgt über ein Bit im Kontrollregister. In die ALARM-Register kann nur geschrieben werden; ein Leseimpuls auf die Adressen der TOD-Register ergibt immer die Zeit, unabhängig vom Zustand des ALARM-Kontrollbits.
Um die Zeit zu lesen oder zu setzen, muß eine bestimmte Reihenfolge eingehalten werden. TOD wird automatisch gestoppt, wenn ein Schreibimpuls für die Stundenregister gültig wird, und wird erst wieder gestartet, nachdem in die 1/10-Sekunden-Register geschrieben wurde. Dies stellt sicher, daß TOD immer mit der gewünschten Zeit gestartet wird. Da ein Übertrag von einem Register zum nächsten sich auch während eines Lesezyklus ereignen könnte, werden während eines Lesezyklus alle Registerinhalte konstant gehalten (gelatcht). Alle vier Register werden gespeichert, sobald die Stunden gelesen werden, und bleiben gespeichert, bis die 1/10-Sekunden gelesen wurden. Erst danach zeigen die Register die aktuellen Werte. Wenn nur ein Register gelesen werden soll, gibt es kein Problem mit dem Übertrag. Das Register kann sofort gelesen werden. Nach dem Stunden-Register muß aber immer das 1/10-Sekunden-Register gelesen werden, um die Verriegelung aufzuheben.

LESEN

REG	NAME								
8	TOD 10THS	0	0	0	0	T_8	T_4	T_2	T_1
9	TOD SEC	0	SH_4	SH_2	SH_1	SL_8	SL_4	SL_2	SL_1
A	TOD MIN	0	MH_4	MH_2	MH_1	ML_8	ML_4	ML_2	ML_1
B	TOD HR	PM	0	0	HH	HL_8	HL_4	HL_2	HL_1

SCHREIBEN

CRB_7=0 TOD
CRB_7=1 ALARM
(Gleiches Format wie LESEN)

SERIELLER PORT (SDR)

Dies ist ein gebuffertes, synchrones, 8 Bit breites Schieberegistersystem. Ein Kontrollbit wählt entweder Ein-/oder Ausgabemodus. Im Eingabemodus werden die Daten vom Pin SP, gesteuert durch positive Taktflanken am Pin CNT, in ein Schieberegister geschoben. Nach 8 Impulsen am Eingang CNT werden die Daten in das serielle Datenregister übernommen, und ein Interrupt wird erzeugt. Wird dieser Port als Ausgang benutzt, bestimmt Timer A die Baudrate. Die Daten werden mit 1/2 der Underflowrate von Timer A an Pin SP herausgeschoben. Die größte mögliche Baudrate ist Ø2/4; sie wird aber durch Kabelkapazitäten und der Geschwindigkeit, mit der der Empfänger auf den Dateneingang reagiert, begrenzt.
Die Übertragung beginnt, nachdem in das serielle Datenregister geschrieben wurde (vorausgesetzt, Timer A läuft im Modus CONTINUOUS). Das Taktsignal von Timer A erscheint als Ausgangssignal an Pin CNT. Die Daten aus dem seriellen Datenregister werden in das Schieberegister übernommen und ausgegeben, wenn an CNT ein Impuls erscheint. Die Ausgabe wird mit der negativen Flanke von CNT gültig und bleibt gültig bis zur nächsten negativen Flanke. Nach 8 Impulsen an Pin CNT wird ein Interrupt erzeugt, um anzuzeigen, daß die nächsten Daten übertragen werden können. Falls das serielle Datenregister vor diesem Interrupt mit neuen Daten geladen wurde, werden diese automatisch in das Schieberegister geladen, und die Übertragung wird fortgesetzt. Falls also der Prozessor das Schieberegister rechtzeitig nachlädt, ist die Übertragung kontinuierlich. Wenn keine Daten mehr übertragen werden sollen, erscheint nach 8 Impulsen an CNT an diesem Ausgang ein H-Pegel, und Pin SP bleibt auf dem Pegel, der dem zuletzt übertragenen Bit entspricht.
SDR gibt zuerst das MSB aus, diese Reihenfolge sollte auch bei der Eingabe verwendet werden. Weil die benutzten Pins bidirektional sind, können viele 6526-Bausteine auf einen seriellen Bus zusammengeschaltet werden, wobei einer der Bausteine als Master, der Daten und Takt ausgibt, und alle anderen als Slaves fungieren. Deshalb sind diese Pins "open-drain"-Schaltungen. Die Vorschrift für Verteilung von Master-/Slavefunktion kann über den seriellen Bus oder spezielle Leitungen übertragen werden.

REG	NAME								
C	SDR	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

INTERRUPT CONTROL (ICR)

Es gibt 5 mögliche Quellen für einen Interrupt: Underflow von Timer A oder B; TOD ALARM; serieller Port voll/leer und $\overline{\text{FLAG}}$.
Die Maskier- und Interruptinformationen sind in einem Register zusammengefaßt. Das INTERRUPTKONTROLLREGISTER besteht aus einem Maskenregister, in das nur hineingeschrieben werden kann, und einem Datenregister, das nur gelesen werden kann. Jeder Interrupt setzt ein entsprechendes Bit im Datenregister. Wird der Interrupt durch das Maskenregister nicht gesperrt, wird das MSB des Datenregisters gesetzt (IR-Bit) und der Pin $\overline{\text{IRQ}}$ Low geschaltet. Sind mehrere 6526 zusammengeschaltet, können die IR-Bits abgefragt werden, um festzustellen, welcher Baustein den Interrupt ausgelöst hat. Nachdem das Datenregister gelesen wurde, wird es gelöscht und $\overline{\text{IRQ}}$ High gelegt. Da das Datenregister unabhängig vom Maskenregister gesetzt wird und jedes Interruptbit einzeln maskiert werden kann, um einen Interrupt zu verhindern, ist es möglich, Interruptanforderungen und ausgeführte Interrupts zu mischen.
Wenn das Bit IR abgefragt wird, wird das Datenregister gelöscht, die Informationen müssen also vom Benutzer gerettet werden.
Das Maskenregister ermöglicht eine einfache Steuerung der Maskierung. Wenn man in das Register schreibt und das 7. Bit der geschriebenen Daten (SET/$\overline{\text{CLEAR}}$) 0 ist, werden alle Bits, die 1 gesetzt sind, gelöscht, während die Bits, die 0 sind, nicht beeinflußt werden. Falls das 7. Bit der geschriebenen Daten 1 ist, wird jedes Maskierungsbit, das 1 ist, gesetzt, während diejenigen, die 0 sind, nicht berührt werden. Damit IR gesetzt werden und ein Interrupt ausgelöst werden kann, muß das korrespondierende Maskierungsbit gesetzt sein.

LESEN (INT DATA)

REG	NAME								
D	ICR	IR	0	0	FLG	SP	ALRM	TB	TA

SCHREIBEN (INT MASK)

REG	NAME								
D	ICR	S/$\overline{\text{C}}$	X	X	FLG	SP	ALRM	TB	TA

STEUERREGISTER

Der 6526 hat zwei Steuerregister: CRA und CRB. CRA ist mit TIMER A und CRB mit TIMER B verbunden. Es gilt folgendes Registerformat:

CRA:

Bit	Name	Funktion
0	START	1=START TIMER A, 0=STOP TIMER A. Dieses Bit wird automatisch rückgestellt, wenn es im one-shot-mode zu einem Unterlauf kommt.
1	PBON	1=TIMER AUSGABE A liegt an PB6 an, 0=PB6 Normalbetrieb.
2	OUTMODE	1=TOGGLE, 0=PULS
3	RUNMODE	1=ONE-SHOT, 0=KONTINUIERLICH
4	LOAD	1=FORCE LOAD (dies ist eine STROBE-Eingabe. Es erfolgt keine Datenspeicherung, Bit 4 liest stets eine 0, und das Schreiben einer 0 hat keinen Einfluß).
5	INMODE	1=TIMER A zählt positive CNT-Übergänge, 0=TIMER A zählt Ø2-Impulse.
6	SPMODE	1=AUSGABE SERIELLER PORT, 0=SERIELLER PORT (externer Taktgeber erforderlich).
7	TODIN	1=50-Hz-clock am TOD-Pin ergibt korrekte Uhrzeit. 0=60-Hz-clock am TOD-Pin ergibt korrekte Uhrzeit.

CRB:

Bit	Name	Funktion
		(Bits CRB0–CRB4 entsprechen CRA0–CRA4 von TIMER B. Bit 1 steuert jedoch die TIMER-Ausgabe B auf PB7.)
5, 6	INMODE	Bits CRB5 und CRB6 wählen eine der vier Eingabemodi von TIMER B:

CRB6	CRB5	
0	0	TIMER B zählt Ø2-Impulse.
0	1	TIMER B zählt positive CNT-Übergänge.
1	0	TIMER B zählt Unterlauf-Impulse von TIMER A.
1	1	TIMER B zählt Unterlauf-Impulse von TIMER A, während CNT H-Pegel hat.

7	ALARM	1=ALARM setzen durch Schreiben in TOD-Register, 0=TOD-clock setzen durch Schreiben in TOD-Register.

REG	NAME	TOD IN	SP MODE	IN MODE	LOAD	RUN MODE	OUT MODE	PB ON	START
E	CRA	0=60Hz 1=50Hz	0=INPUT 1=OUTPUT	0=ϕ2 1=CNT	1=FORCE LOAD (STROBE)	0=CONT. 1=O.S.	0=PULSE 1=TOGGLE	0=PB_6OFF 1=PB_6 ON	0=STOP 1=START

TA

REG	NAME	ALARM	IN	MODE	LOAD	RUN MODE	OUT MODE	PB ON	START
F	CRB	0=TOD 1=ALARM	0 1 1 1	0=ϕ2 1=CNT 0=TA 1=CNT·TA	1=FORCE LOAD (STROBE)	0=CONT. 1=O.S.	0=PULSE 1=TOGGLE	0=PB_7 OFF 1=PB_7 ON	0=STOP 1=START

TB

Alle nicht benutzten Register-Bits werden durch das Schreiben nicht beeinflußt und beim Lesen auf Null gesetzt.

Änderungen vorbehalten. **COMMODORE SEMICONDUCTOR GROUP** übernimmt keinerlei Verantwortung für Schäden, die aus der Anwendung der hier beschriebenen Produkte oder Schaltungen entsteht. Es werden keinerlei Lizenzen hinsichtlich der Patentrechte oder anderer Rechte erteilt.

ANHANG N

6566/6567 VIDEO-INTERFACE-CONTROLLER (VIC-II) CHIP SPECIFICATIONS

Beschreibung
Die Bausteine 6566 und 6567 sind Mehrzweck-Farb-Video-Bausteine sowohl für den Einsatz in Computer-Videoterminals als auch in Videospielen. Sie enthalten 47 Kontrollregister, auf die über einen normalen 8-Bit-Mikroprozessorbus (65XX) zugegriffen werden kann, und können auf bis zu 16KB RAM zugreifen, um Videoinformationen abzulegen.
Im Folgenden werden die verschiedenen Betriebsarten und deren Optionen beschrieben.

ZEICHENDARSTELLUNGSMODUS

In dieser Betriebsart holt der Baustein Characterzeiger aus dem VIDEO-MATRIX-Bereich des Speichers und übersetzt diese in die Adresse der Punktmatrix des Zeichens, welche sich in dem 2048 Byte großen CHARACTER-BASE-Bereich des Speichers befindet. Die Videomatrix umfaßt 1000 aufeinanderfolgende Speicherplätze, welche alle einen 8-Bit-Characterzeiger enthalten. Die Plazierung der Videomatrix im Speicher wird durch VM13-VM10 in Register 24 ($18) festgelegt. Diese 4 Bit bilden die 4 MSB der Videomatrixadresse. Die 10 unteren Bits werden von einem internen Zähler bereitgestellt, der die 1000 Zeichenspeicherplätze durchzählt. Man beachte, daß die Bausteine nur 14 Adreßausgänge haben, deshalb ist zusätzliche Systemhardware notwendig, um den gesamten Speicherbereich des Systems ansprechen zu können.

ZEICHENZEIGERADRESSE

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Die 8 Bit langen Characterzeiger erlauben, daß bis zu 256 verschiedene Zeichen gleichzeitig verfügbar sind. Jedes Zeichen ist im Character-Base-Bereich als 8x8-Punktematrix in 8 aufeinanderfolgenden Bytes abgelegt. Die Plazierung der Character-Base wird durch CB13-CB11 im Register 24 ($18) festgelegt, diese bilden die 3 MSB der Adresse. Die 11 unteren Bits werden aus dem Characterzeiger (8 Bit) aus der Videomatrix, der ein bestimmtes Zeichen definiert, und einem 3-Bit-Rasterzähler (RC2-RC0), der eines der 8 Zeichenbytes auswählt, gebildet. Die resultierenden Zeichen werden in 25 Zeilen zu jeweils 40 Zeichen zusammengefaßt. Zusätzlich zum Characterzeiger gehört zu jeder Stelle der Videomatrix ein 4-Bit-FARBNYBBLE (der Videomatrixspeicher muß also 12 Bit breit sein), welcher eine von 16 verschiedenen Farben für jedes Zeichen einzeln auswählt.

ZEICHENDATENADRESSE

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	CB12	CB11	D7	D6	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

Betriebsart "STANDARDZEICHEN" (MCM=BMM=ECM=0)

In dieser Betriebsart werden die 8 aufeinanderfolgenden Bits der Character Base direkt als die 8 Zeilen des Zeichens dargestellt. Bei einer 0 wird die Hintergrundfarbe #0 (aus Register 33 ($21)), bei einer 1 die Farbe, die durch das Farbnybble bestimmt wird, dargestellt (siehe Farbcodetabelle).

FUNKTION	ZEICHENBIT	FARBANZEIGE
Hintergrund	0	Hintergrundfarbe #0 (Register 33 ($21))
Vordergrund	1	Durch 4-Bit-Farbnybble gewählte Farbe

Somit hat jedes Zeichen eine Farbe (festgelegt durch das Farbnybble), und alle Zeichen haben die gleiche Hintergrundfarbe.

Betriebsart "MEHRFARBIGE ZEICHEN" (MCM=1, BMM=ECM=0)

Diese Betriebsart ermöglicht es, vierfarbige Zeichen mit geringer Auflösung darzustellen. Sie wird eingeschaltet, wenn das Bit MCM in Register 22 ($16) 1 gesetzt wird, wodurch die in der Character Base gespeicherten Daten unterschiedlich interpretiert werden. Ist das MSB des Farbnybbles 0, wird das Zeichen wie bei der Betriebsart "Standardzeichen" dargestellt. Dies erlaubt es, die beiden Betriebsarten zu mischen, es sind jedoch nur die 8 ersten Farben darstellbar. Wenn das MSB des Farbnybbles 1 ist (falls MCM:MSB(CM)=1), werden immer je 2 Bits folgendermaßen interpretiert:

FUNCTION	CHARACTER BIT PAIR	COLOR DISPLAYED
Background	00	Background #0 Color (register 33 ($21))
Background	01	Background #1 Color (register 34 ($22))
Foreground	10	Background #2 Color (register 35 ($23))
Foreground	11	Color specified by 3 LSB of color nybble

Da immer 2 Bits benötigt werden, um einen Punkt zu beschreiben, wird das Zeichen jetzt als 4x8-Punktematrix dargestellt, wobei jeder Punkt doppelt so breit ist wie im Normalbetrieb. Man beachte, daß jedes Zeichenfeld jetzt 4 Farben beinhalten kann, 2 als Vordergrund, 2 als Hintergrund (siehe MOB-Priorität).

Betriebsart "ERWEITERTE FARBE" (ECM=1, BMM=MCM=0)

Diese Betriebsart erlaubt es, für jedes einzelne Zeichen mit einer Auflösung von 8x8 Punkten eine von 4 Hintergrundfarben auszuwählen. Diese Betriebsart wird eingeschaltet, wenn das Bit ECM des Registers 17 ($11) 1 gesetzt wird. Die Punktmatrix des Zeichens wird genauso wie bei der Betriebsart "Standardzeichen" dargestellt (durch eine 1 wird die durch das Farbnybble bestimmte Vordergrundfarbe dargestellt), aber die 2 MSB des Characterzeigers bestimmen die Hintergrundfarbe des Zeichens nach folgendem Schema:

CHARACTERZEIGER MSB-PAAR	HINTERGRUND-FARBE FÜR BIT 0
00	Hintergrundfarbe #0 (Register 33 ($21))
01	Hintergrundfarbe #1 (Register 34 ($22))
10	Hintergrundfarbe #2 (Register 35 ($23))
11	Hintergrundfarbe #3 (Register 36 ($24))

Da die 2 MSB des Characterzeigers zur Auswahl der Hintergrundfarbe gebraucht werden, können nur noch 64 verschiedene Zeichen dargestellt werden. Der Baustein interpretiert CB10 und CB9 unabhängig vom Characterzeiger als 0, so daß nur die ersten 64 Zeichen dargestellt werden können.
In dieser Betriebsart kann für jedes Zeichen eine der 16 Vordergrundfarben und eine von 4 verfügbaren Hintergrundfarben bestimmt werden.

Anmerkung: Die beiden Betriebsarten "Mehrfarbige Zeichen" und "Erweiterte Farbe" sollten nicht gleichzeitig eingeschaltet werden.

BIT MAP MODUS

In dieser Betriebsart holt der Baustein Daten auf eine andere Art und Weise aus dem Speicher und stellt sie so dar, daß eine direkte Beziehung zwischen dem dargestellten Punkt und dem Bit im Speicher besteht. Diese Betriebsart verfügt über eine Auflösung von 320 Punkten horizontal und 200 Punkten vertikal. Sie wird eingeschaltet, indem das Bit BMM im Register 17 ($11) 1 gesetzt wird. Auf die Videomatrix wird noch genauso wie bei der Zeichendarstellung zugegriffen, aber ihr Inhalt wird jetzt nicht mehr als Characterzeiger, sondern als Farbinformation interpretiert. Der Videomatrixzähler wird als Adresse benutzt, um die Daten für die Darstellung der Punkte aus dem 8000 Byte umfassenden Anzeigespeicher zu holen. Die Adresse ist folgendermaßen zusammengesetzt:

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

VCx bezeichnet den Ausgang des Videomatrixzählers, RCx den Rasterlinienzähler, und CB13 stammt aus Register 24 ($18). Der Videomatrixzähler wählt für 8 Rasterlinien die gleichen Speicherplätze an, während der Rasterzähler nach jeder horizontalen Zeile um 1 erhöht wird. Nachdem die 8. Zeile geschrieben ist, wählt der Videomatrixzähler die nächsten 40 Speicherplätze an. Aus dieser Adressierungsart resultiert, daß immer 8 aufeinanderfolgende Speicherplätze eine 8x8-Punktematrix auf dem Bildschirm bilden.

Betriebsart "STANDARD BIT MAP" (BMM=1, MCM=0)

Hierbei wird die Farbinformation nur aus den Daten der Videomatrix abgeleitet (der Farbnybble wird nicht beachtet). Die 8-Bit-Daten aus der Videomatrix werden in 2 4-Bit-Daten aufgeteilt, wodurch es möglich wird, 2 verschiedene Farben in jeder 8x8-Punktematrix darzustellen. Wenn ein Bit des Anzeigespeichers 0 ist, erscheint der Punkt in der Farbe, die durch die unteren 4 Bit definiert wird. Wenn das Bit 1 ist, wird der Punkt in der Farbe gesetzt, die durch die oberen 4 Bit des entsprechenden Datenwortes in der Videomatrix festgelegt wird.

BIT	ANZEIGEFARBE
0	Unteres Nybble des Videomatrix-Zeigers
1	Oberes Nybble des Videomatrix-Zeigers

"MEHRFARBIGE DARSTELLUNG" (BMM=MCM=1)

Diese Betriebsart wird eingeschaltet, indem das Bit MCM im Register 22 ($16) und das Bit BMM im Register 17 ($11) 1 gesetzt werden. Sie benutzt dieselbe Ansteuerung des Speichers wie die Standardbetriebsart, interpretiert die Daten jedoch anders. Je zwei Bits werden zusammengefaßt und nach folgendem Schema ausgewertet:

BIT-PAAR	ANZEIGEFARBE
00	Hintergrundfarbe #0 (Register 33 ($21))
01	Oberes Nybble des Videomatrix-Zeigers
10	Unteres Nybble des Videomatrix-Zeigers
11	Videomatrix-Farbnybble

Man beachte, daß das Farbnybble bei dieser Betriebsart benutzt wird. Da immer 2 Bit benötigt werden, um die Farbe eines Punktes zu bestimmen, sind die Punkte doppelt so breit wie in der Standardbetriebsart; es können also nur 160 Punkte horizontal und 200 Punkte vertikal dargestellt werden. Wenn man diese Betriebsart nutzt, können also 3 voneinander unabhängig ausgewählte Farben für jede 8x8-Punktematrix zusätzlich zur Hintergrundfarbe auf dem Bildschirm dargestellt werden.

DARSTELLUNG VON BEWEGLICHEN OBJEKTEN

Bewegliche Objekte (engl. movable objekt block, MOB) sind eine spezielle Art von Zeichen, die an jedem beliebigen Ort des Bildschirms unabhängig von der 8x8-Punktematrix erzeugt werden können. Bis zu 8 MOBs können gleichzeitig erzeugt werden, jedes wird durch 63 Bytes im Speicher beschrieben und als Anordnung von 24x21 Punkten dargestellt (s.u.). Eine Anzahl von Sonderfunktionen macht die MOBs besonders für Videospiele und -graphiken interessant.

MOB-ANZEIGEBLOCK

BYTE	BYTE	BYTE
00	01	02
03	04	05
.	.	.
.	.	.
.	.	.
57	58	59
60	61	62

MOB EINSCHALTEN

Jeder MOB kann einzeln durch Setzen des entsprechenden Bits MnE im Register 21 ($15) auf dem Bildschirm dargestellt werden. Wenn das entsprechende Bit 0 ist, ist der MOB nicht nur abgeschaltet, er wird auch bei der Ausführung der MOB-Sonderfunktionen nicht berücksichtigt.

POSITION

Die Lage des MOB auf dem Bildschirm wird durch die X- und Y-Koordinaten mit einer Auflösung von 512 Punkten horizontal und 256 Punkten vertikal bestimmt, die in den entsprechenden Registern abgelegt sind. Dabei beziehen sich die Koordinaten auf den linken oberen Punkt des MOB. Wenn X zwischen 23 und 347 ($17–$157) und Y zwischen 50 und 249 ($32–$F9) liegt, ist der MOB sichtbar. Da der MOB nicht in jeder Position sichtbar ist, kann er übergangslos vom Bildschirm verschwinden und wieder erscheinen.

FARBE

Zur Festlegung der Farbe hat jeder MOB ein eigenes 4-Bit-Register. Es gibt zwei Betriebsarten:

Normale Darstellung (MnMC=0)

In dieser Betriebsart ist der MOB an den Stellen, wo eine 0 geschrieben ist, durchsichtig, es erscheint also die Hintergrundfarbe. Eine 1 bewirkt, daß die Farbe erscheint, die durch das Farbregister bestimmt wird.

Mehrfarbige MOBs (MnMC=1)

Jeder MOB kann unabhängig von den anderen mehrfarbig gestaltet werden, indem das entsprechende Bit MnMC im Register 28 ($1C) 1 gesetzt wird. Dann werden die Datenbits des MOBs paarweise folgendermaßen interpretiert:

BIT PAIR	COLOR DISPLAYED
00	Transparent
01	MOB Multi-color #0 (register 37 ($25))
10	MOB Color (registers 39-46 ($27-$2E))
11	MOB Multi-color #1 (register 38 ($26))

Da immer zwei Bits benötigt werden, um einen Punkt zu bestimmen, wird die Auflösung auf 12x21 Punkte reduziert; da die Punkte aber doppelt so breit gezeichnet werden, ändert sich die Größe des MOBs nicht.
Man beachte, daß bis zu 3 verschiedene Farben pro MOB zur Verfügung stehen, aber 2 Farben für alle mehrfarbigen MOBs gültig sind.

VERGRÖSSERUNG

Jeder MOB kann einzeln um den Faktor 2 in beiden Richtungen vergrößert werden. Zwei Register enthalten die Kontrollbits für die Vergrößerung:

REGISTER	FUNCTION
23 ($17)	Horizontal expand MnXE—"1"=expand; "0"=normal
29 ($1D)	Vertical expand MnYE—"1"=expand; "0"=normal

Wenn die MOBs vergrößert werden, findet keine Verbesserung der Auflösung statt. Die 24x21-(bzw. 12x21)-Punktematrix wird nur entsprechend vergrößert (der kleinste Punkt eines MOBs kann also in der mehrfarbigen Darstellung bis zu viermal größer erscheinen).

PRIORITÄT

Die Priorität des MOBs in bezug auf andere auf dem Bildschirm dargestellte Informationen kann für jedes MOB einzeln durch Setzen des entsprechenden Bits (MnDP) im Register 27 ($1B) beeinflußt werden:

REG BIT	PRIORITY TO CHARACTER OR BIT MAP DATA
0	Non-transparent MOB data will be displayed (MOB in front)
1	Non-transparent MOB data will be displayed only instead of Bkgd #0 or multi-color bit pair 01 (MOB behind)

MOB – DISPLAY DATA PRIORITY

MnDP = 1	MnDP = 0
MOBn	Foreground
Foreground	MOBn
Background	Background

Untereinander haben die MOBs eine feste Rangfolge, wobei MOB 0 den höchsten und MOB 7 den niedrigsten Rang besitzen. Wenn Punkte von 2 MOBs (ausgenommen transparente Punkte) zusammenfallen, werden immer die des MOBs mit der niedrigsten Nummer dargestellt.

ERKENNEN VON KOLLISIONEN

Zwei Arten von Berührungen werden erkannt: die Berührung zweier MOBs und die Überlappung eines MOBs mit einer anderen dargestellten Information:

MOB-MOB-Berührung:

Eine Berührung zweier MOBs findet statt, wenn die nichttransparenten Teile zweier MOBs an der gleichen Stelle abgebildet werden sollen (die Berührung transparenter Teile hat keine Folgen). Dann werden die Bits MnM für die beiden beteiligten MOB im Register 30 ($1E) 1 gesetzt. Diese bleiben gesetzt, bis das Register ausgelesen wird, dann werden alle Bits automatisch 0 gesetzt. Berührungen werden auch dann festgestellt, wenn sich die MOBs außerhalb des Bildschirms befinden.
Überlappung mit anderen Informationen:
Wenn ein MOB einen Bildpunkt berührt, der nicht in der Hintergrundfarbe dargestellt wird, wird im Register 31 ($1F) das entsprechende Bit MnD 1 gesetzt. Transparente Teile des MOB spielen auch hier keine Rolle. Für spezielle Anwendungen wird auch die Überlappung mit dem Datenpaar 01 (Mehrfarbige Darstellung) nicht als Kollision erkannt. Auf diese Weise können Daten dargestellt werden, ohne daß diese Einfluß auf das Erkennen von Berührungen haben.
Eine solche Berührung zwischen einem MOB und einer anderen, auf dem Bildschirm dargestellten Information kann auch außerhalb des Bildschirms in der horizontalen Richtung stattfinden, wenn eine gültige Information durch "scrolling" (s.u.) außerhalb des Bildschirms gelangt ist.
Die zuständigen Interruptlatches werden gesetzt, sobald in dem betreffenden Register das erste Bit gesetzt wird. Sobald ein Bit in dem Register gesetzt ist, wird durch nachfolgende Berührungen kein Interruptflag mehr gesetzt, bis das betreffende Register durch Auslesen gelöscht wurde.

MOB-SPEICHERZUGRIFF

Die Daten für jeden MOB werden in 63 aufeinanderfolgenden Bytes im Speicher abgelegt. Die 8 Blocks werden durch 8 MOB-Zeiger definiert, die am Ende der Videomatrix abgelegt sind. Da die Videomatrix nur 1000 Bytes benötigt, ist es möglich, von Platz 1016–1023 der Videomatrix die MOB-Zeiger 0–7 abzulegen. Dieser 8 Bit lange Zeiger bildet zusammen mit dem 6 Bit langen MOB-Bytezähler (um 63 verschiedene Bytes zu adressieren) eine 14 Bit lange Adresse.

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
MP7	MP6	MP5	MP4	MP3	MP2	MP1	MP0	MC5	MC4	MC3	MC2	MC1	MC0

MPx bezeichnet die Bits des MOB-Zeigers und MCx die des MOB-Bytezählers, die intern erzeugt werden. Die MOB-Zeiger werden am Ende jeder Videozeile eingelesen. Wenn der Inhalt eines Y-Registers mit dem des Rasterlinienzählers übereinstimmt, beginnt der Zugriff auf die Daten des zugehörigen MOB. Der MOB-Bytezähler durchläuft automatisch die 63 Bytes und stellt immer 3 Bytes in jeder Zeile dar.

SONSTIGE MERKMALE

BILDSCHIRM ABSCHALTEN

Der Bildschirm kann abgeschaltet werden, indem das Bit DEN in Register 17 ($11) "0" gesetzt wird. Dann erscheint der gesamte Bildschirm in der Farbe, die durch Register 32 ($20) festgelegt wird. Dann wird nur die erste Phase des Speicherzugriffs benötigt, wodurch der Systembus vollständig dem Prozessor zur Verfügung steht. Allerdings greift der VIC noch auf MOB-Daten zu, wenn diese nicht abgeschaltet sind.
DEN muß für normalen Videobetrieb 1 gesetzt sein.

AUSWAHL DER REIHEN UND SPALTEN

Das normale Anzeigeformat besteht aus 25 Reihen zu je 40 Zeichen. Für Spezialanwendungen kann das Anzeigefenster auf 24 Reihen zu 38 Zeichen reduziert werden. Dies hat keinen Einfluß auf die Größe der dargestellten Zeichen, außer, daß Zeichen, die vorher an die Begrenzung stießen, jetzt von dieser überdeckt werden. Diese Betriebsart wird durch 2 Bits gesteuert: RSEL aus Register 17 ($11) und CSEL aus Register 22 ($16). Sie haben folgende Bedeutung:

RSEL	NUMBER OF ROWS	CSEL	NUMBER OF COLUMNS
0	24 rows	0	38 columns
1	25 rows	1	40 columns

Normalerweise wird man das größere Fenster benutzen, das kleinere wird hauptsächlich in Verbindung mit "scrolling" benutzt.

SCROLLING

Die Anzeige kann jeweils um eine Zeichenstelle in horizontaler und vertikaler Richtung verschoben werden. Wenn dies in Verbindung mit dem kleineren Anzeigeformat benutzt wird, kann eine leichte Schwenkbewegung der Anzeige durchgeführt werden, während der Systemspeicher nur aktualisiert zu werden braucht, wenn eine neue Zeile oder Spalte geschrieben werden muß. "Scrolling" kann auch dazu benutzt werden, um eine feste Anzeige im Fenster zu zentrieren.

BITS	REGISTER	FUNCTION
X2,X1,X0	22 ($16)	Horizontal Position
Y2,Y1,Y0	17 ($11)	Vertical Position

LIGHT PEN

Bei einer negativen Flanke am Lightpen-Eingang wird die gerade gültige Bildschirmposition in das Registerpaar 19 (LPX) und 20 (LPY) geschrieben. Da in Register 19 nur die 8 MSB der X-Position gespeichert werden, insgesamt aber 516 verschiedene Positionen unterschieden werden müßten (dazu benötigte man 9 Bits), beträgt die Auflösung in der X-Position nur 2 Punkte.
In der Y-Richtung reichen die 8 Bit zur Auflösung des Bildschirms aus. Das Lightpen-Register kann nur einmal pro Bilddurchlauf getriggert werden, mehrmaliges Triggern hat keinen Einfluß. Deshalb muß man das Lightpen-Register einige Male abfragen, bevor man den Lightpen auf den Bildschirm richtet (die Anzahl der Abfragen hängt von den Eigenschaften des Griffels ab).

RASTERREGISTER

Dieses Register hat 2 Funktionen: Wenn dieses Register gelesen wird, erscheinen die 8 unteren Bit der z. Z. gültigen Rasterposition (das MSB –RC8– steht in Register 17 ($11)). Dies kann man dazu benutzen, um den Inhalt der Anzeige ohne Flackern zu ändern, indem die Änderung außerhalb des sichtbaren Bereichs vorgenommen wird. Der sichtbare Bereich liegt zwischen Raster 51 und 251 ($033–$0FB). Wenn in das Register geschrieben wird (einschließlich RC8), wird der Wert für einen internen Vergleich gespeichert. Wird der gespeicherte Wert erreicht, wird das Rasterinterruptflag gesetzt (Register 25).

INTERRUPTREGISTER

Das Interruptregister (Register 25($19)) zeigt den Status der 4 Interruptquellen. Ein Bit wird 1 gesetzt, wenn die entsprechende Interruptquelle einen Interrupt verlangt. In der Tabelle sind die 4 Bits und die zugehörigen Quellen aufgeführt.

LATCH BIT	ENABLE BIT	WHEN SET
IRST	ERST	Set when (raster count) = (stored raster count)
IMDC	EMDC	Set by MOB–DATA collision register (first collision only)
IMMC	EMMC	Set by MOB–MOB collision register (first collision only)
ILP	ELP	Set by negative transition of LP input (once per frame)
IRQ		Set high by latch set and enabled (invert of IRQ/ output)

Damit ein Interrupt durchgeführt und der Ausgang IRQ 0 gesetzt werden kann, muß das entsprechende Bit in Register 26 ($1A) (Interrupt enable) 1 gesetzt werden. Wenn ein Interruptbit gesetzt ist, wird es erst gelöscht, wenn an dieser Stelle eine 1 geschrieben wird. Dadurch wird eine beliebige Abarbeitung der Interrupts ermöglicht, ohne daß Speicherplätze oder Software zur Erhaltung der Interrupt-Information benötigt wird.

REFRESH FÜR DYNAMISCHE RAMS

Im Baustein ist eine Schaltung eingebaut, die den Refresh dynamischer RAMs kontrolliert. Nach jeder Rasterlinie werden 5 8-Bit-Zeilenadressen zur Auffrischung der RAMs erzeugt, wodurch garantiert ist, daß bei Speicherorganisation von 128 Reihen zu 512 Speicherplätzen jede Reihe mindestens alle 2 ms aufgefrischt wird (bei 256x256 organisierten Speichern alle 3,66 ms). Da der Refreshimpuls während der 1. Phase des Systemtaktes erzeugt wird, beeinflußt er andere Bausteine (Prozessor, I/O-Port etc.) auf dem 65XX-Systembus nicht.

Der Baustein erzeugt auch RAS- und CAS-Signale, die normalerweise direkt mit den dynamischen RAMs verbunden sind, und zwar während der 2. Phase des Systemtaktes und für jeden Videospeicherzugriff (einschließlich Refresh), so daß keine externe Takterzeugung notwendig ist.

RESET

Das Reset-Bit (RES) in Register 22 ($16) wird für den normalen Betrieb nicht benötigt. Es sollte demzufolge bei der Initialisierung des Video-Chips auf 0 gesetzt werden. Wenn das Bit auf 1 gesetzt wird, unterbricht der Video-Chip seine Funktion einschließlich Video-Ausgangssignal, Refresh für die dynamischen RAMs und System-Bus-Zugriff.

FUNKTIONSWEISE DES 6566/6567

Der Videobaustein 6566/6567 arbeitet auf besondere Art und Weise mit dem Systembus. Da die 65XX-Familie nur während der 2. Phase des Taktes (High) auf den Bus zugreift, benutzt der Videobaustein den Bus normalerweise nur während der 1. Phase des Taktes. Deshalb stören solche Operationen wie Refresh oder der Zugriff auf Zeichendaten den Prozessor nicht und beeinflussen nicht dessen Arbeitsgeschwindigkeit. Der Baustein stellt alle Kontrollsignale zur Verfügung, die benötigt werden, um diese Aufteilung des Busses aufrechtzuerhalten.
Der Videobaustein liefert das Signal AEC (Address enable control), das die Adreßtreiber des Prozessors hochohmig schaltet, damit der Videochip auf den Bus zugreifen kann. AEC ist aktiv, wenn der Ausgang 0 geschaltet ist, somit kann der Ausgang direkt an die AEC-Eingänge der 65XX-Familie gelegt werden.
Normalerweise ist dieses Signal nur während der 1. Phase des Taktes gültig, so daß der Prozessor nicht gestört wird. Aufgrund dieser zeitlichen Aufteilung müssen alle Speicherzugriffe in der halben Zykluszeit durchgeführt werden. Da der Videochip einen 1-MHz-Takt liefert, müssen alle Speicherzyklen wie Adresseanlegen, Datenzugriff und Datentransport zu den lesenden Bausteinen in 500 ms erledigt sein.
Einige Funktionen des Bausteins erfordern mehr Daten, als während der 1. Taktphase gelesen werden können, so z. B. der Zugriff auf Characterzeiger in der Videomatrix und das Lesen der MOB-Daten, wenn diese dargestellt werden sollen. Dann muß ein Zugriff des Prozessors auf den Bus verhindert und auch während der 2. Taktphase gelesen werden. Dies wird durch das Signal BA (Bus available) erreicht. Dieses Signal ist normalerweise 1, es wird jedoch während der 1. Taktphase auf 0 gelegt, um zu zeigen, daß der Videochip während der 2. Taktphase auf

den Bus zugreifen will. Dann bleiben dem Prozessor noch drei 2. Taktphasen, um laufende Speicherzugriffe abzuschließen. Während der vierten 2. Taktphase, nachdem BA auf Low geschaltet wurde, bleibt AEC auf Low, damit der Videochip die Daten holen kann.
Der Ausgang BA ist normalerweise mit den Eingängen RDY der anderen 65XX-Bausteine verbunden. Der Zugriff auf Characterzeiger geschieht alle 8 Rasterlinien innerhalb des Anzeigefensters und erfordert 40 aufeinanderfolgende Zugriffe während der 2. Taktphase, um die Videomatrixzeiger zu holen. Das Einlesen der MOB-Daten erfordert folgende 4 Speicherzugriffe:

PHASE	DATA	CONDITION
1	MOB Pointer	Every raster
2	MOB Byte 1	Each raster while MOB is displayed
1	MOB Byte 2	Each raster while MOB is displayed
2	MOB Byte 3	Each raster while MOB is displayed

Die MOB-Zeiger werden nach jeder Zeile während der 1. Taktphase gelesen. Falls erforderlich, werden zusätzliche Zyklen zum Einlesen der MOB-Daten benutzt. Alle notwendigen Signale zur Steuerung des Busses werden also von dem Videochip zur Verfügung gestellt.

SPEICHERANSTEUERUNG

Die zwei Versionen des Bausteins unterscheiden sich in der Art und Weise, wie sie die Adressen anlegen. 6566 hat 13 vollständig dekodierte Adressen, die direkt mit den Adressen des Systembusses verbunden werden können.
Die Adressen von Baustein 6567 werden gemultiplext, um sie direkt mit den Adressen von dynamischen 64K RAMs verbinden zu können. Die Adressen A00–A06 sind an den Ausgängen A00–A06 gültig, wenn der Ausgang RAS Low geschaltet ist, während die Adressen A08–A013 an den Ausgängen A00–A05 erscheinen, wenn CAS Low geschaltet wird. Die Ausgänge A07–A11 an diesem Baustein sind statische Adreßausgänge, die mit einem ROM (2Kx8) verbunden werden können (dann müssen die unteren Adressen zwischengespeichert werden).

SCHNITTSTELLE ZUM PROZESSOR

Abgesehen von den speziellen, oben beschriebenen Speicherzugriffen kann auf die Register des Bausteins genauso wie bei jedem anderen Peripheriebaustein zugegriffen werden. Folgende Signale stehen für die Schnittstelle mit dem Prozessor zur Verfügung:

DATENBUS (DB7–DB0)

Dies ist ein bidirektionaler Datenport, der von den Signalen an den Pins CS, RW und Phase 0 kontrolliert wird. Auf den Datenbus kann nur zugegriffen werden, wenn AEC=Phase0=1 und CA=0 gilt.

CHIP SELECT (CS/)

Wenn dieser Pin Low gelegt wird, kann man in Verbindung mit dem Signal RW und den Adressen auf die Register des Bausteins zugreifen. Dieser Eingang wird nur beachtet, wenn AEC=Phase0=1 gilt.

READ/WRITE (R/W)

Mit diesem Eingang wird in Verbindung mit CS die Richtung des Datenflusses festgelegt. Bei RW=1 werden Daten aus dem angewählten Register auf den Datenbus geschrieben, ist RW=0, ist der Datenfluß umgekehrt.

ADRESSBUS (A05–A00)

Die Anschlüsse A0–A5 sind bidirektional. Wenn der Prozessor auf den Videochip zugreift, sind es Adreßeingänge, und die angelegte Bitkombination wählt eines der Register an (siehe Tabelle).

CLOCK OUT (Phase 0)

An diesem Ausgang erscheint der 1-MHz-Takt für den Prozessor. Alle Systemoperationen beziehen sich auf diesen Takt, der aus dem 8-MHz-Takt durch Teilung gewonnen wird.

INTERRUPT (IRQ/)

Dieser Ausgang wird Low gelegt, wenn von einer eingeschalteten Interruptquelle ein Interrupt ausgelöst wird. Es ist ein "open-drain"-Ausgang, der einen externen pull-up-Widerstand benötigt.

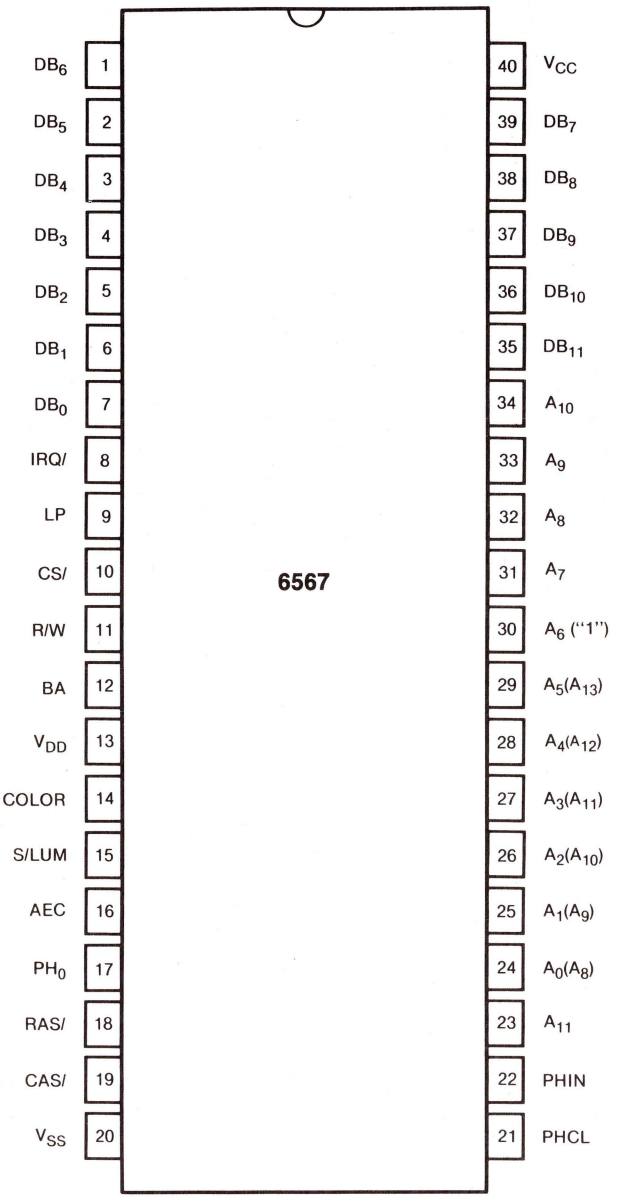
VIDEOANSCHLUSS

Das Videosignal der Bausteine besteht aus zwei getrennten Signalen, die extern gemischt werden müssen. SYNC/LUM beinhaltet alle Videoinformationen wie horizontale und vertikale Synchronisierung und die Hell-/Dunkelsteuerung, und ist ein "open-drain"-Ausgang, der einen externen pull-up-Widerstand benötigt. COLOR enthält alle Farbinformationen, auch den Farbhilfsträger, und ist ein "open-source"-Ausgang, der einen externen Widerstand von 1000 Ohm gegen Masse benötigt. Nach geeigneter Mischung dieser Signale kann das resultierende Signal einen Videomonitor oder mit einem entsprechenden Modulator einen normalen Fernseher ansteuern.

ZUSAMMENFASSUNG DER BUS-AKTIVITÄTEN BEIM 6566/6567

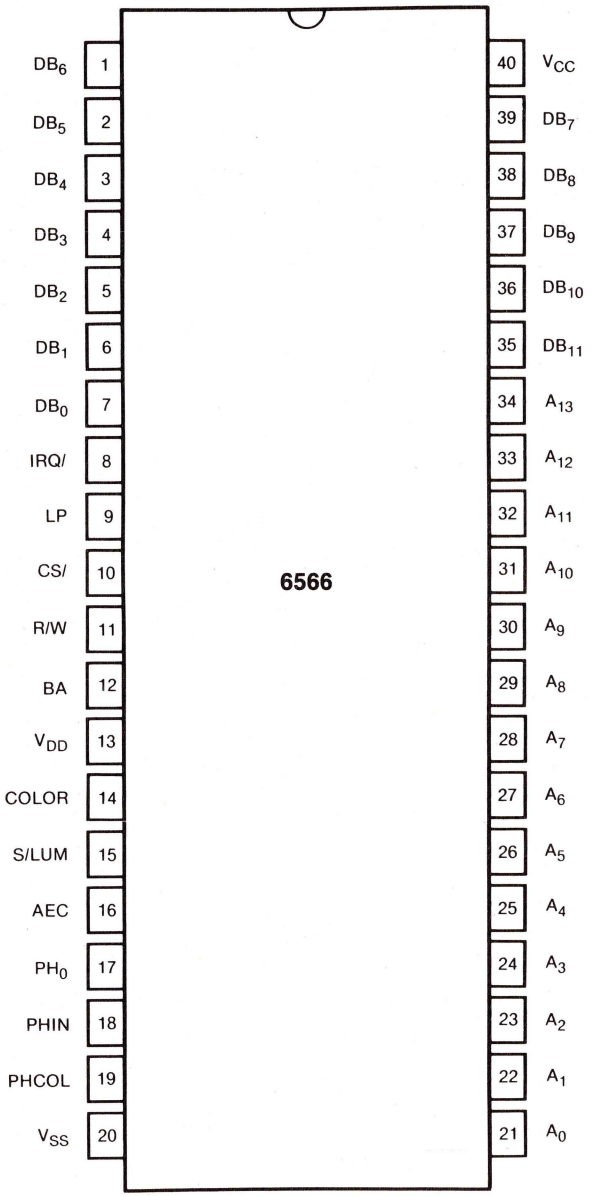
AEC	PH0	CS/	R/W	ACTION
0	0	X	X	PHASE 1 FETCH, REFRESH
0	1	X	X	PHASE 2 FETCH (PROCESSOR OFF)
1	0	X	X	NO ACTION
1	1	0	0	WRITE TO SELECTED REGISTER
1	1	0	1	READ FROM SELECTED REGISTER
1	1	1	X	NO ACTION

PIN-ANORDNUNG



(Multiplex-Adressen in Klammern)

PIN-ANORDNUNG



REGISTERBELEGUNG

ADDRESS		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
00	($00)	M0X7	M0X6	M0X5	M0X4	M0X3	M0X2	M0X1	M0X0	MOB 0 X-position
01	($01)	M0Y7	M0Y6	M0Y5	M0Y4	M0Y3	M0Y2	M0Y1	M0Y0	MOB 0 Y-position
02	($02)	M1X7	M1X6	M1X5	M1X4	M1X3	M1X2	M1X1	M1X0	MOB 1 X-position
03	($03)	M1Y7	M1Y6	M1Y5	M1Y4	M1Y3	M1Y2	M1Y1	M1Y0	MOB 1 Y-position
04	($04)	M2X7	M2X6	M2X5	M2X4	M2X3	M2X2	M2X1	M2X0	MOB 2 X-position
05	($05)	M2Y7	M2Y6	M2Y5	M2Y4	M2Y3	M2Y2	M2Y1	M2Y0	MOB 2 Y-position
06	($06)	M3X7	M3X6	M3X5	M3X4	M3X3	M3X2	M3X1	M3X0	MOB 3 X-position
07	($07)	M3Y7	M3Y6	M3Y5	M3Y4	M3Y3	M3Y2	M3Y1	M3Y0	MOB 3 Y-position
08	($08)	M4X7	M4X6	M4X5	M4X4	M4X3	M4X2	M4X1	M4X0	MOB 4 X-position
09	($09)	M4Y7	M4Y6	M4Y5	M4Y4	M4Y3	M4Y2	M4Y1	M4Y0	MOB 4 Y-position
10	($0A)	M5X7	M5X6	M5X5	M5X4	M5X3	M5X2	M5X1	M5X0	MOB 5 X-position
11	($0B)	M5Y7	M5Y6	M5Y5	M5Y4	M5Y3	M5Y2	M5Y1	M5Y0	MOB 5 Y-position
12	($0C)	M6X7	M6X6	M6X5	M6X4	M6X3	M6X2	M6X1	M6X0	MOB 6 X-position
13	($0D)	M6Y7	M6Y6	M6Y5	M6Y4	M6Y3	M6Y2	M6Y1	M6Y0	MOB 6 Y-position
14	($0E)	M7X7	M7X6	M7X5	M7X4	M7X3	M7X2	M7X1	M7X0	MOB 7 X-position
15	($0F)	M7Y7	M7Y6	M7Y5	M7Y4	M7Y3	M7Y2	M7Y1	M6Y0	MOB 7 Y-position
16	($10)	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	MSB of X-position
17	($11)	RC8	ECM	BMM	DEN	RSEL	Y2	Y1	Y0	See text
18	($12)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Raster register
19	($13)	LPX8	LPX7	LPX6	LPX5	LPX4	LPX3	LPX2	LPX1	Light Pen X
20	($14)	LPY7	LPY6	LPY5	LPY4	LPY3	LPY2	LPY1	LPYC	Light Pen Y
21	($15)	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	MOB Enable
22	($16)	—	—	RES	MCM	CSEL	X2	X1	X0	See text
23	($17)	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	MOB Y-expand

REGISTERBELEGUNG

ADDRESS		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
24	($18)	VM13	VM12	VM11	VM10	CB13	CB12	CB11	—	Memory Pointers
25	($19)	IRQ	—	—	--	ILP	IMMC	IMBC	IRST	Interrupt Register
26	($1A)	—	—	—	—	ELP	EMMC	EMBC	ERST	Enable Interrupt
27	($1B)	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	MOB-DATA Priority
28	($1C)	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	MOB Multicolor Sel
29	($1D)	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	MOB X-expand
30	($1E)	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	MOB-MOB Collision
31	($1F)	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	MOB-DATA Collision
32	($20)	—	—	—	—	EC3	EC2	EC1	EC0	Exterior Color
33	($21)	—	—	—	—	B0C3	B0C2	B0C1	B0C0	Bkgd #0 Color
34	($22)	—	—	—	—	B1C3	B1C2	B1C1	B1C0	Bkgd #1 Color
35	($23)	—	—	—	—	B2C3	B2C2	B2C1	B2C0	Bkgd #2 Color
36	($24)	—	—	—	—	B3C3	B3C2	B3C1	B3C0	Bkgd #3 Color
37	($25)	—	—	—	—	MM03	MM02	MM01	MM00	MOB Multicolor #0
38	($26)	—	—	—	—	MM13	MM12	MM11	MM10	MOB Multicolor #1
39	($27)	—	—	—	—	M0C3	M0C2	M0C1	M0C0	MOB 0 Color
40	($28)	—	—	—	—	M1C3	M1C2	M1C1	M1C0	MOB 1 Color
41	($29)	—	—	—	—	M2C3	M2C2	M2C1	M2C0	MOB 2 Color
42	($2A)	—	—	—	—	M3C3	M3C2	M3C1	M3C0	MOB 3 Color
43	($2B)	—	—	—	—	M4C3	M4C2	M4C1	M4C0	MOB 4 Color
44	($2C)	—	—	—	—	M5C3	M5C2	M5C1	M5C0	MOB 5 Color
45	($2D)	—	—	—	—	M6C3	M6C2	M6C1	M6C0	MOB 6 Color
46	($2E)	—	—	—	—	M7C3	M7C2	M7C1	M7C0	MOB 7 Color

Anmerkung: Ein Strich zeigt an, daß kein Anschluß besteht. In diesem Fall lauten alle Lesungen "1".

FARB-CODES

D4	D3	D1	D0	HEX	DEC	COLOR
0	0	0	0	0	0	BLACK
0	0	0	1	1	1	WHITE
0	0	1	0	2	2	RED
0	0	1	1	3	3	CYAN
0	1	0	0	4	4	PURPLE
0	1	0	1	5	5	GREEN
0	1	1	0	6	6	BLUE
0	1	1	1	7	7	YELLOW
1	0	0	0	8	8	ORANGE
1	0	0	1	9	9	BROWN
1	0	1	0	A	10	LT RED
1	0	1	1	B	11	DARK GREY
1	1	0	0	C	12	MED GREY
1	1	0	1	D	13	LT GREEN
1	1	1	0	E	14	LT BLUE
1	1	1	1	F	15	LT GREY

ANHANG O

6581 SOUND INTERFACE DEVICE (SID) CHIP SPECIFICATIONS

KONZEPT

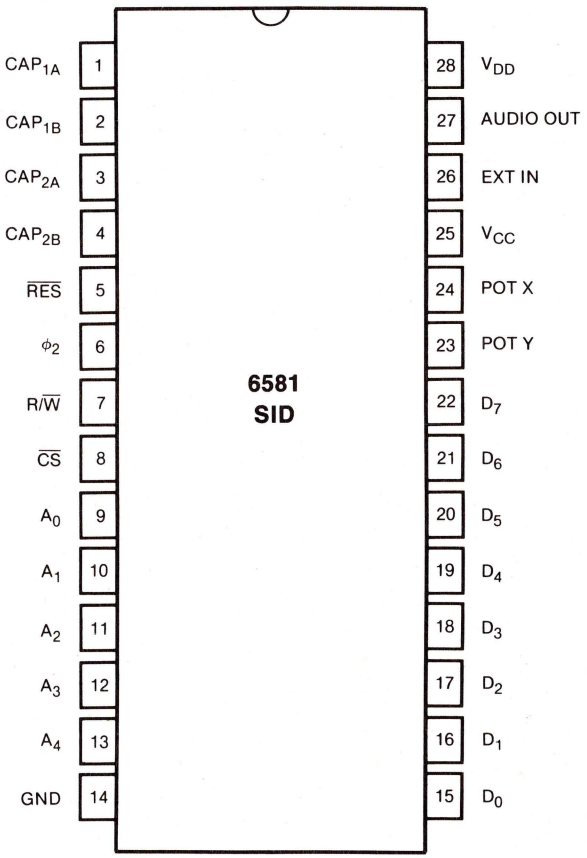
Der SID 6581 ist ein dreistimmiger, elektronischer Musik-/Geräuschgenerator, buskompatibel mit der Prozessorfamilie 65XX und ähnlichen Prozessoren. Die Tonfrequenz kann ebenso wie Klang und Lautstärke in einem weiten Bereich mit hoher Genauigkeit eingestellt werden. Spezielle Schaltkreise verringern die nötige Software, was den Einsatz in Heimcomputern und preiswerten Musikinstrumenten ermöglicht.

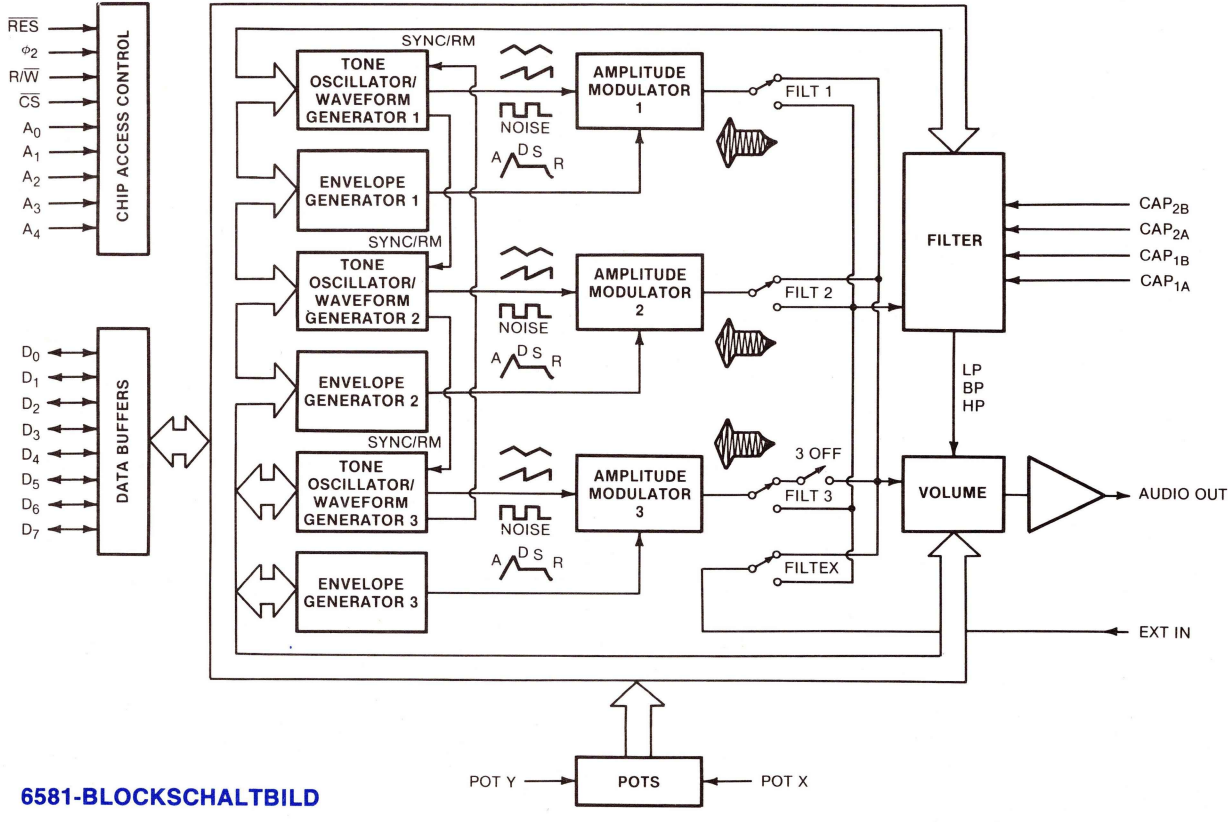
BESONDERHEITEN

- 3 Tongeneratoren, 0–4 kHz
- 4 Kurvenformen pro Generator wählbar:
  Sinus, Dreieck, Rechteck (einstellbar) oder Rauschen
- 3 Amplitudenmodulatoren, jeweils 48 dB
- 3 Hüllkurvengeneratoren
  exponentieller Kurvenverlauf
  Anstiegszeit: 2 ms–8 s
  Abfallzeit: 6 ms–24 s
  Sustain-(Halte-)Pegel: 0–max. Lautstärke
  Ausklingzeit: 6 ms–24 s
- Synchronisierung der Oszillatoren
- Ringmodulation
- Programmierbare Filter
  Eck- bzw. Mittenfrequenz: 30 Hz–12 kHz
  Abfall: 12 dB/Oktave
  Tiefpaß, Bandpaß, Hochpaß oder Notchfilter

- Gesamtlautstärkeeinstellung
- Zufallsgenerator
- Anschlußmöglichkeit für 2 Potentiometer
- Audioeingang

PIN-ANORDNUNG





6581-BLOCKSCHALTBILD

BESCHREIBUNG

Der 6581 hat 3 Stimmen, die voneinander unabhängig, miteinander oder mit externen Audioquellen kombiniert eingesetzt werden können. Jede Stimme besteht aus einem Tongenerator, einem Hüllkurvengenerator und einem Amplitudengenerator. Die Tonhöhe kann über einen weiten Bereich eingestellt werden. Der Generator produziert 4 Kurvenformen mit der eingestellten Frequenz. Mit den jeweiligen harmonischen Obertönen jeder Kurvenform läßt sich die Klangfarbe beeinflussen. Die Dynamik der Lautstärke wird vom Amplitudengenerator eingestellt, welcher wiederum vom Hüllkurvengenerator beeinflußt wird. Wenn er angesteuert wird, entsteht eine Hüllkurve mit programmierbarer Anstiegs- und Abfallzeit. Zusätzlich zu den 3 Stimmen gibt es noch ein programmierbares Filter, mit dem es möglich ist, komplexe, dynamische Klangfarben herzustellen (subtraktive Synthese).
SID erlaubt dem Prozessor, die Veränderungen am Ausgang des 3. Generators und den 3. Hüllkurvengenerator zu lesen. Diese Ausgänge können dazu benutzt werden, dem Prozessor die notwendigen Informationen zur Steuerung eines Vibrato, Wobbelgenerators, durchstimmbaren Filters etc. zu liefern. Der dritte Oszillator kann auch als Zufallsgenerator für Spiele benutzt werden. Zwei A/D-Umsetzer sind für den Anschluß von zwei Potentiometern vorgesehen. Diese können als "PADDLE" in einem Spiel oder zur Steuerung in einem Musiksynthesizer benutzt werden. Der SID kann externe Audiosignale verarbeiten, wodurch mehrere SIDs zu einer sogenannten "Daisy chain" oder einem polyphonen System zusammengeschaltet werden können.

SID-KONTROLLREGISTER

Es gibt 29 8-Bit-Register im SID, die die Klangerzeugung steuern. Hierbei handelt es sich um Nur-Schreib- oder Nur-Leseregister, die in Tabelle 1 aufgelistet sind.

Tabelle 1 SID-Registerbelegung

	ADDRESS A_4	A_3	A_2	A_1	A_0	REG # (HEX)	DATA D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REG NAME	REG TYPE
															Voice 1	
0	0	0	0	0	0	00	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE-ONLY
1	0	0	0	0	1	01	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE-ONLY
2	0	0	0	1	0	02	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE-ONLY
3	0	0	0	1	1	03	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE-ONLY
4	0	0	1	0	0	04	NOISE				TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-ONLY
5	0	0	1	0	1	05	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE-ONLY
6	0	0	1	1	0	06	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE-ONLY
															Voice 2	
7	0	0	1	1	1	07	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE-ONLY
8	0	1	0	0	0	08	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE-ONLY
9	0	1	0	0	1	09	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE-ONLY
10	0	1	0	1	0	0A	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE-ONLY
11	0	1	0	1	1	0B	NOISE				TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-ONLY
12	0	1	1	0	0	0C	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE-ONLY
13	0	1	1	0	1	0D	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE-ONLY
															Voice 3	
14	0	1	1	1	0	0E	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE-ONLY
15	0	1	1	1	1	0F	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE-ONLY
16	1	0	0	0	0	10	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE-ONLY
17	1	0	0	0	1	11	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE-ONLY
18	1	0	0	1	0	12	NOISE				TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE-ONLY
19	1	0	0	1	1	13	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE-ONLY
20	1	0	1	0	0	14	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE-ONLY
															Filter	
21	1	0	1	0	1	15	—	—	—	—	—	FC_2	FC_1	FC_0	FC LO	WRITE-ONLY
22	1	0	1	1	0	16	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	FC HI	WRITE-ONLY
23	1	0	1	1	1	17	RES_3	RES_2	RES_1	RES_0	FILTEX	FILT 3	FILT 2	FILT 1	RES/FILT	WRITE-ONLY
24	1	1	0	0	0	18	3 OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	MODE/VOL	WRITE-ONLY
															Misc.	
25	1	1	0	0	1	19	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	POT X	READ-ONLY
26	1	1	0	1	0	1A	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	POT Y	READ-ONLY
27	1	1	0	1	1	1B	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	OSC_3/RANDOM	READ-ONLY
28	1	1	1	0	0	1C	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	ENV_3	READ-ONLY

KONTROLLREGISTER

GENERATOR 1

FREQUENZ LOW/FREQUENZ HIGH (00,01)

Zusammen stellen diese Register ein 16-Bit-Wort dar, welches die Frequenz des 1. Oszillators nach folgender Gleichung festlegt:

$$F_{out} = (F_n * F_{clK}/16777216)Hz = (F_n * 0{,}0587214734)Hz$$

F_n ist die 16-Bit-Zahl aus den Registern, F_{clK} ist der Systemtakt, der am Eingang ∅2 anliegt.
Dadurch kann die Tonhöhe ohne wahrnehmbare Tonschritte durchgestimmt werden.

PW LO/PW HI (02,03)

Diese Register bilden eine 12-Bit-Zahl (Bit 4–7 von PW HI werden nicht genutzt), welche das Tastverhältnis des Rechteckgenerators 1 bestimmt. Das Verhältnis errechnet sich wie folgt:

$$PW_{out} = (PW_n/40{,}95)\%$$

PW_n ist hier die 12-Bit-Zahl in den PW-Registern. Das Tastverhältnis kann so ohne wahrnehmbare Schritte verändert werden. Diese Register haben nur dann einen hörbaren Effekt, wenn der Rechteckgenerator 1 eingeschaltet ist. Wenn in den Registern 0 oder 4095 steht, entsteht ein DC-Signal, 2048 ergibt dagegen ein Rechteck mit 50% Tastverhältnis.

KONTROLLREGISTER (04)

Dieses Register enthält 8 Kontrollbits:

GATE (Bit 0)

Steuert den Hüllkurvengenerator. Wenn es 1 gesetzt ist, beginnt der Zyklus Attack/Decay/Sustain. Wenn es 0 gesetzt wird, beginnt der Zyklus Release (genauere Erklärung im Kapitel Hüllkurvengenerator).

SYNC (Bit 1)

Wenn dieses Bit 1 gesetzt ist, wird die Frequenz des Generators 1 mit der Frequenz des Generators 3 synchronisiert ("Hard-Sync"-Effekte).
Wenn die Frequenz des Generators 1 unter Berücksichtigung der Frequenz des Generators 3 variiert wird, entsteht eine große Zahl komplexer harmonischer Strukturen. Wenn Sync funktionieren soll, muß die Frequenz des dritten Generators kleiner als die des ersten Generators sein (nicht 0). Keine anderen Parameter der 3. Stimme beeinflussen Sync.

RING MOD (Bit 2)

Wenn dieses Bit 1 gesetzt ist, wird der Dreieckgenerator der 1. Stimme durch eine mit Frequenz 1 und 3 ringmodulierte Spannung ersetzt. Wenn jetzt die Frequenz 1 verändert wird, entstehen nichtharmonische Obertöne, welche für Klingel- oder Gonggeräusche und Spezialeffekte gebraucht werden. Hierfür muß bei Generator 1 Dreieck und bei Generator 3 eine Frequenz größer als Null eingestellt sein. Andere Parameter der 3. Stimme haben keine Wirkung.

TEST (Bit 3)

Wenn dieses Bit 1 gesetzt ist, wird der 1. Generator zurückgesetzt und auf 0 gehalten, bis das Testbit gelöscht ist. Der Rauschgenerator ist abgestellt, und der Rechteckgenerator wird auf DC gehalten. Zwar wird dieses Bit normalerweise für Testzwecke benutzt, es kann jedoch Generator 1 auch mit externen Ereignissen synchronisieren (kompliziertere Kurvenformen, Realzeit-Verarbeitung).

BIT 4

Wenn dieses Bit gesetzt ist, ist der Dreieckgenerator eingeschaltet. Diese Kurvenform ist arm an Obertönen und hat einen weichen, einer Flöte ähnlichen Charakter.

BIT 5

Wenn dieses Bit gesetzt ist, ist der Sägezahngenerator eingeschaltet. Dieser ist reich an geraden und ungeraden Obertönen und ergibt einen breiten, an Blechbläser erinnernden Klang.

BIT 6

Wenn dieses Bit gesetzt ist, ist der Rechteckgenerator ausgewählt. Der Obertonanteil kann durch das Tastverhältnis eingestellt werden, die Möglichkeiten reichen vom hellen, hohlen Rechteckklang bis zum nasalen, schrillen Klang kurzer Impulse. Wenn das Tastverhältnis beim Spielen verändert wird, entsteht ein "pashing"-Effekt, der den Eindruck einer Bewegung erweckt. Schnelles Hin- und Herschalten zwischen verschiedenen Tastverhältnissen kann interessante Sequenzen erzeugen.

BIT 7

Wenn dieses Bit gesetzt ist, ist der Rauschgenerator eingeschaltet.
Dieser produziert Rauschen, das die Klangfarbe vom tiefen Rumpeln bis zum zischenden weißen Rauschen durch die Frequenzeinstellung des Generators 1 verändern kann. Rauschen braucht man, um Explosionen, Gewehrschüsse, Düsenjäger, Wind und ähnliche Geräusche zu erzeugen, oder für Trommeln und Becken. Indem man die Frequenz beim Spielen verändert, kann man Stürme nachbilden.
Obwohl einer dieser Generatoren eingeschaltet sein muß, um die 1. Stimme am Ausgang erklingen zu lassen, ist es nicht notwendig, die einzelnen Generatoren auszuschalten, um die Stimme abzustellen. Die Lautstärke wird nur durch den Hüllkurvengenerator bestimmt.

Bemerkung: Die Oszillatorausgänge können nicht addiert werden. Wenn mehr als ein Oszillator eingeschaltet ist, wird das Ergebnis eine logische "Und"-Verknüpfung der Kurvenform sein. Obwohl damit neue Kurvenformen erzeugt werden können, sollte dies vorsichtig benutzt werden. Wenn Rauschen eingeschaltet ist und zusätzlich eine Kurvenform eingeschaltet wird, verstummt das Rauschen, bis das Testbild zurückgesetzt oder der Pin 5 (RES) Low geschaltet wird.

ATTACK/DECAY (05)

Bit 4–7 wählt eine von 16 möglichen Anstiegszeiten (Attack) für den Hüllkurvengenerator der 1. Stimme. Dies bestimmt, wie schnell der Ausgang auf volle Lautstärke anschwillt, wenn der Hüllkurvengenerator eingeschaltet wird (Gate).
Bit 0 bis 3 wählen eine von 16 möglichen Abschwellzeiten (Decay) aus. Diese Zeit gibt an, wie schnell die Lautstärke vom Spitzenwert auf den ausgewählten Haltepegel (Sustain) abfällt.

SUSTAIN/RELEASE (06)

Bit 4–7 wählt einen von 16 möglichen Halte-(Sustain-)Pegeln des Hüllkurvengenerators aus. Diese Phase folgt dem Abfall, der Pegel wird gehalten, solange das Gatebit gesetzt ist. Der Pegel kann von Stille (0) bis zur Spitzenlautstärke (16) linear eingestellt werden. Ein Wert von 8 würde demnach der halben Lautstärke, die beim Anstieg (Attack) erreicht wird, entsprechen.
Mit Bit 0–3 kann eine der 16 Ausklingarten gewählt werden. Der Ausklingzyklus folgt der Haltezeit, wenn das Gatebit zurückgesetzt wird. Dann fällt die Lautstärke vom Haltepegel auf Null in der eingestellten Zeit. Die Ausklingzeiten mit den Werten 0–16 sind identisch mit den Abfallzeiten 0–16.

Bemerkung: Der geschilderte Ablauf kann ohne Einschränkung jederzeit durch das Gatebit verändert werden. Wenn das Gatebit z. B. zurückgesetzt wird, bevor die Anschlagszeit abgelaufen ist, beginnt sofort bei dem erreichten Pegel die Ausklingzeit. Wenn jetzt das Gatebit wieder gesetzt wird, beginnt sofort eine neue Anstiegszeit bei dem jetzt erreichten Pegel. Dadurch können komplizierte Amplitudenverläufe durch Realzeitprogrammierung erzeugt werden.

Tabelle 2 Hüllkurvenraten

WERT	ANSTIEGSRATE	ABKLING/ABFALLRATE
DEZIMAL (HEX)	(Takt/Zyklus)	(Takt/Zyklus)
0 (0)	2 ms	6 ms
1 (1)	8 ms	24 ms
2 (2)	16 ms	48 ms
3 (3)	24 ms	72 ms
4 (4)	38 ms	114 ms
5 (5)	56 ms	168 ms
6 (6)	68 ms	204 ms
7 (7)	80 ms	240 ms
8 (8)	100 ms	300 ms
9 (9)	250 ms	750 ms
10 (A)	500 ms	1.5 s
11 (B)	800 ms	2.4 s
12 (C)	1 s	3 s
13 (D)	3 s	9 s
14 (E)	5 s	15 s
15 (F)	8 s	24 s

Bemerkung zur Tabelle: Die angegebenen Werte beziehen sich auf eine Taktfrequenz von 1 MHz. Wenn die Taktfrequenz abweicht, müssen die Werte mit 1 MHz/F(clk) multipliziert werden.
Die angegebenen Zeiten beziehen sich auf die Zeit, die benötigt wird, um den Zyklus abzuschließen. Eine Anstiegszeit von 16 ms (Wert 2) bedeutet z. B., daß die Lautstärke nach 16 ms von Pegel 0 den Spitzenwert erreicht. Die Abfall-/Ausklingzeiten beziehen sich auf die Zeit, die benötigt wird, um vom Spitzenwert auf Null zu sinken.

STIMME 2

Die Register $07–$0D kontrollieren die Stimme 2 und haben die gleiche Funktion wie die Register 00–06, mit folgenden Ausnahmen:

1) SYNC synchronisiert den Generator 2 mit Generator 1.
2) RING MOD ersetzt die Dreieckspannung durch die ringmodulierte Kombination der Generatoren 1 und 2.

STIMME 3

Die Register $0E–$14 haben für die 3. Stimme die gleiche Funktion wie die Register 00–06, mit folgenden Ausnahmen:

1) SYNC synchronisiert Generator 3 mit Generator 2.
2) RING MOD ersetzt die Dreieckspannung durch ringmodulierte Kombination der Generatoren 2 und 3.

Wenn man einen Ton ansprechen will, muß man also Frequenz, Kurvenform, Effekte (SYNC, RING MOD) und Hüllkurve bestimmen. Dann kann man den Ton jederzeit mit dem Gatebit abrufen. Der Ton hält solange an, bis das Gatebit zurückgesetzt wird. Jede Stimme kann einzeln, mit unterschiedlichen Parametern oder mit anderen Stimmen zusammen benutzt werden, um eine einzelne, kräftige Stimme zu erhalten. Dabei kann eine leichte Verstimmung der Oszillatoren untereinander oder die Stimmung in musikalischen Intervallen einen wirkungsvollen Effekt ergeben.

FILTERREGISTER

FC LO/FC HI (Register $15,$16)

Diese Register bilden zusammen eine 11-Bit-Zahl (Bit 3–7 des Registers FC LO werden nicht genutzt). Diese bestimmt linear die Mitten- bzw. Eckfrequenz, sie kann von 30 Hz bis 12 kHz eingestellt werden.

RES/FILT (Register $17)

Bit 4–7 dieses Registers bestimmen die Resonanz des Filters. Dieser Effekt hebt die Frequenzen in der Nähe der Eckfrequenz an, dadurch ergibt sich ein schärferer Klang. Es können 16 verschiedene Einstellungen vorgenommen werden (linear von 0 bis 16). Bit 0–3 legt fest, welches Signal gefiltert wird:

FILT 1 (Bit 0):

Eine 0 in diesem Register bedeutet, daß die Stimme 1 ohne Veränderung auf den Audioausgang geschaltet wird (Bypass). Wenn es gesetzt ist, wird die 1. Stimme gefiltert, ihr Obertonanteil verändert sich.

FILT 2 (Bit 1):

Gleiche Wirkung wie Bit 0 für die 2. Stimme.

FILT 3 (Bit 2):

Gleiche Wirkung wie Bit 0 für die 3. Stimme.

FILTEX (Bit 3):

Gleiche Wirkung wie Bit 0 für den Audioeingang.

MODE/VOL (Register $18)

Bits 4–7 bestimmen verschiedene Filter- und Ausgabearten:

LP (Bit 4): Wenn dieses Bit gesetzt ist, ist der Tiefpaß eingeschaltet, d. h. alle Frequenzen unterhalb der Eckfrequenz bleiben unverändert, alle Frequenzen oberhalb werden mit 12 dB/Oktave abgeschwächt. Es entstehen volle Klänge.

BP (Bit 5):

Das gleiche für den Bandpaß. Alle Frequenzen unter und oberhalb der Mittenfrequenz werden mit 6 dB/Oktave abgeschwächt. Es entstehen offene, dünne Klänge.

HP (Bit 6):

Das gleiche für den Hochpaß. Alle Frequenzen oberhalb der Eckfrequenz bleiben unverändert, unterhalb werden sie mit 12 dB/Oktave abgeschwächt. Es entstehen summende und blecherne Klänge.

3 OFF (Bit 7):

Eins gesetzt, trennt dieses Bit die 3. Stimme vom Audioausgang ab. Wenn man Stimme 3 am Filter vorbei schaltet (mit FILT 3=0) und 3 OFF gesetzt ist, wird die 3. Stimme nicht auf den Ausgang geschaltet, kann aber zur Modulation der anderen Stimmen benutzt werden.

Bemerkung: Die Filter können zusammengeschaltet werden. Z. B. ergibt LP zusammen mit HP ein Notchfilter (Bandsperre). Damit der Filtereffekt hörbar wird, muß ein Filter eingeschaltet sein und eine Stimme durch das Filter geführt werden. Das Filter ist vielleicht das wichtigste Element im SID, da es durch die subtraktive Synthese viele Klangmöglichkeiten schafft (das Filter entzieht dem obertonreichen Eingangssignal bestimmte Frequenzen). Gute Ergebnisse erzielt man, wenn man die Eck- bzw. Mittenfrequenz während des Spielens variiert.

VOL 0–VOL 3 (Bit 0–3):

Hiermit wird die Gesamtlautstärke zwischen 0 (leise) und 15 (laut) in linearen Stufen eingestellt. Hiermit kann die Lautstärke beim Zusammenschalten mehrerer Chips abgestimmt oder Effekte wie Tremolo erzeugt werden. Bei VOL=0 ist der Ausgang stumm.

WEITERE EIGENSCHAFTEN

POTX (Register $19)

Dieses Register erlaubt dem Prozessor, die Position eines Potentiometers, das an Pin 24 angeschlossen ist, in Schritten von 0 bei kleinstem Widerstand bis 255 bei vollem Widerstand zu erkennen. Das Ergebnis liegt immer vor und wird alle 512 Takte erneuert.

POTY (Register $1A)

Das gleiche für ein zweites Potentiometer (an Pin 23).

OSC 3/RANDOM (Register $1B)

Dieses Register erlaubt dem Prozessor, die 8 oberen Bits des Ausgangs von Oszillator 3 zu lesen. Die Art der Ziffernfolge, die entsteht, ist direkt mit der Kurvenform verknüpft. Beim Sägezahn wächst die Zahlenfolge von 0 bis 255, um dann wieder bei 0 zu beginnen. Beim Dreieck wächst die Zahl von 0 bis 255, um dann von 255 bis 0 zu fallen. Beim Rechteck springt die Zahl zwischen 0 und 255 hin und her. Beim Rauschen wird eine Kette von Zufallszahlen erzeugt, deshalb kann dieses Register auch als Zufallszahlengenerator benutzt werden. Es gibt viele Anwendungsmöglichkeiten für dieses Register, die wichtigste ist vielleicht die Steuerung von Modulationen. Die Zahlen, die erzeugt werden, können per Software zum Inhalt der Oszillator- oder Filterfrequenzregister addiert werden etc. So können viele dynamische Effekte erzeugt werden: Sirenen, indem OSC3 (Sägezahn) zum Frequenzregister eines anderen Oszillators addiert wird. Vibrato entsteht, wenn OSC3 (Dreieck, 7 Hz) zum Frequenzregister einer anderen Stimme addiert wird. Dabei sollte der Audioausgang der 3. Stimme abgeschaltet sein (3OFF=1).

ENV 3 (Register $1C)

Im Prinzip das gleiche wie OSC3, es wird jedoch der Ausgang des Hüllkurvengenerators 3 gelesen. Die Zahlen können z. B. zum Inhalt des Filterfrequenzregisters addiert werden, es entstehen sog. "Harmonische Hüllkurven" und Wahwah-Effekte. "Phasing" entsteht, wenn dieser Ausgang zum Frequenzregister eines Oszillators addiert wird. Um dieses Signal zu erzeugen, muß das Gatebit geschaltet werden. Der Ausgang OSC3 spiegelt immer die Veränderungen am Ausgang des 3. Oszillators wider, er wird nicht vom Hüllkurvengenerator beeinflußt.

PINBESCHREIBUNG

CAP1A, CAP1B, (Pins 1, 2)/ CAP2A, CAP2B (Pins 3, 4):

Hier sollten zwei Kondensatoren für das programmierbare Filter angeschlossen werden. C1 und C2 sollten i. A. 2200 pf haben und aus Polystyrene bestehen. Wenn mehrere SIDs zusammen arbeiten sollen, sollten die Kapazitäten abgeglichen werden.
Der Frequenzbereich (normalerweise 30 Hz bis 12 kHz) kann auf spezielle Probleme zugeschnitten werden. So kann z. B. die obere Eckfrequenz beschnitten werden, um eine bessere Kontrolle über die unteren Frequenzen zu erhalten.

Die obere Eckfrequenz kann nach folgender Gleichung errechnet werden:

$$FC_{max} = 0{,}000026/C$$

C ist die Kapazität. Der Filterbereich erstreckt sich 9 Oktaven nach unten.

RES (Pin 5):

Reseteingang (TTL-Pegel) für den SID. Wenn dieser 10 Takte Low geschaltet ist, sind alle internen Register auf Null zurückgesetzt und der Audioausgang stumm. Er ist normalerweise mit der Resetleitung des Prozessors oder einer Einschaltlogik verbunden.

Ø2 (Pin 6):

Takteingang des SID (TTL-Pegel). Alle Parameter beziehen sich auf diesen Takt, er steuert auch den Datentransport zwischen CPU und SID: Daten können nur dann transportiert werden, wenn Ø2 High liest (somit ist Ø2 für den Datentransport eine Art Chip Select). Normalerweise ist Ø2 mit dem Systemtakt verbunden, dessen Frequenz ungefähr 1 MHz betragen sollte.

R/W:

Dieser TTL-Eingang steuert den Datentransport. Liegt High an, kann der Prozessor Daten auslesen, bei Low Daten in ein Register schreiben.

CS:

Dieser TTL-Eingang steuert den Datentransport, er muß Low sein, damit ein Transport stattfinden kann: Es kann nur gelesen werden, wenn CS=Low, 02=High und R/W=High ist. Geschrieben werden kann nur, wenn CS=Low, 02=High und R/W=Low ist. Normalerweise ist dieser Eingang mit einer Dekodierschaltung verbunden, um den SID im gesamten Adreßbereich plazieren zu können.

A0–A4:

Mit diesen TTL-Eingängen kann eines der 29 Register ausgewählt werden. Es könnten 32 Register angesprochen werden, 3 Adressen sind jedoch nicht belegt. Wenn dort geschrieben werden soll, wird dies ignoriert, beim Lesen werden ungültige Daten gelesen. Die Anschlüsse werden mit den entsprechenden Adressenleitungen des Prozessors verbunden, um den SID genauso ansprechen zu können wie einen Speicher.

GND:

Um beste Ergebnisse zu erzielen, sollte der SID eine vom Digitalteil getrennte Erdleitung erhalten.

D0—D7:

Diese bidirektionalen Leitungen werden zum Datentransport benutzt (TTL-Pegel, können als Ausgang 2 TTL-Eingänge treiben). Sie sind hochohmig, wenn der SID nicht angesprochen wird oder vom Prozessor in den SID geschrieben wird. Beim Lesen werden sie durchgeschaltet und übermitteln die Daten an den Prozessor. Sie werden mit dem Datenbus verbunden.

POTX,POTY:

Dies sind die Eingänge der A/D-Umsetzer, mit denen die Stellung der Potentiometer digitalisiert werden kann. Der Umsetzungsprozeß hängt von der Kapazität ab, die vom Pin nach GND geschaltet ist und über das Potentiometer von $+V_{cc}$ gespeist wird. Die Werte müssen folgender Gleichung entsprechen:

$$R*C = 0{,}00047$$

R ist der max. Widerstand des Potentiometers und C die Kapazität.
Je größer die Kapazität ist, um so kleiner muß R sein. Empfohlen werden: C = 1000 pF; R=470 kOhm. POTX und POTY können unterschiedliche Werte für R und C aufweisen, solange die Gleichung erfüllt ist.

V_{cc}:

Auch für die Spannungsversorgung (+5 V) sollte eine separate Leitung zur Verfügung stehen und ein Blockkondensator dicht am SID plaziert werden.

EXT IN:

Dieser Analogeingang erlaubt es, externe Signale mit dem Ausgangssignal des SID zu mischen oder sie zu filtern. Typische Quellen sind Gesang, Gitarre und Orgel. Der Eingangswiderstand beträgt 100 kOhm. Der Eingang hat einen Offset von 6 V und kann bis zu 3 Vp-p verarbeiten.

Deshalb sollte der Eingang mit einem Elektrolytkondensator von 1000–10000 nF entkoppelt werden. Mit FILTEX = 0 können viele SIDs zusammengeschaltet werden (Verstärkung = 1), die Anzahl wird nur durch den Geräuschspannungspegel im Ausgangssignal begrenzt.
Der Gesamtlautstärkeregler wirkt auch auf diesen Eingang.

AUDIO OUT:

Dieser Ausgang (Open-source) umfaßt die 3 Stimmen, den Filter und den externen Eingang. Der Pegel wird durch den Gesamtlautstärkeregler bestimmt und erreicht max. 2 Vp-p bei einem Offset von 6 V. Es muß ein Widerstand (1 kOhm) gegen Masse geschaltet werden, und ein Elektrolytkondensator von 1000–10000 nF sollte den Ausgang entkoppeln.

V_{DD}:

Auch hier sollte eine separate Leitung vorgesehen werden (+12 V).

MERKMALE VON 6581 SID

ABSOLUTE MAX. NENNWERTE

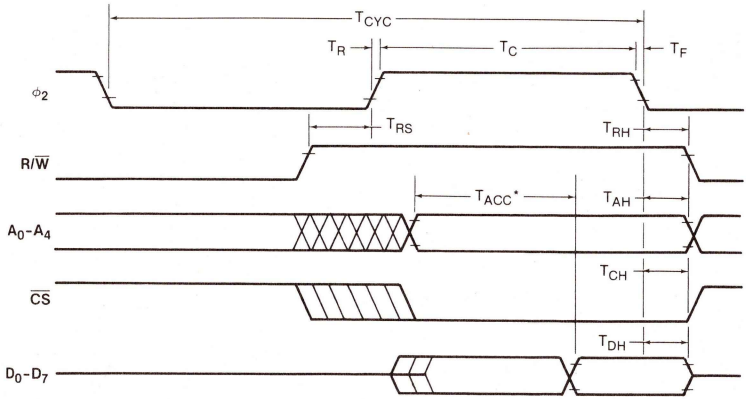
NENNWERT	SYMBOL	WERT	EINHEIT
Versorgungsspannung	V_{DD}	−0,3 bis +17	VDC
Versorgungsspannung	V_{CC}	−0,3 bis +7	VDC
Eingangsspannung (analog)	V_{ina}	−0,3 bis +17	VDC
Eingangsspannung (digital)	V_{ind}	−0,3 bis +7	VDC
Betriebstemperatur	T_A	0 bis 70	°C
Lagertemperatur	T_{STG}	−55 bis +150	°C

ELEKTRISCHE EIGENSCHAFTEN (V_{CC}=12 VDC ± 5%, V_{CC}=5 VDC ± 5%, T_A=0 bis 70°C)

CHARACTERISTIC		SYMBOL	MIN	TYP	MAX	UNITS
Input High Voltage	(RES, ϕ2, R/W, CS,	V_{IH}	2	—	V_{CC}	VDC
Input Low Voltage	A0–A4, D0–D7)	V_{IL}	−0.3	—	0.8	VDC
Input Leakage Current	(RES, ϕ2, R/W, CS, A0–A4; V_{in}=0–5 VDC)	I_{in}	—	—	2.5	μA
Three-State (Off)	(D0–D7; V_{CC}=max)	I_{TSI}	—	—	10	μA
Input Leakage Current	V_{in}=0.4–2.4 VDC					
Output High Voltage	(D0–D7; V_{CC}=min, I load=200 μA)	V_{OH}	2.4	—	V_{CC}−0.7	VDC
Output Low Voltage	(D0–D7; V_{CC}=max, I load=3.2 mA)	V_{OL}	GND	—	0.4	VDC
Output High Current	(D0–D7; Sourcing, V_{OH}=2.4 VDC)	I_{OH}	200	—	—	μA

Output Low Current	(D0–D7; Sinking, V_{OL}=0.4 VDC)	I_{OL}	3.2	—	—	mA
Input Capacitance	(RES, ϕ2, R/W, CS, A0–A4, D0–D7)	C_{in}	—	—	10	pF
Pot Trigger Voltage	(POTX, POTY)	V_{pot}	—	$V_{CC}/2$	—	VDC
Pot Sink Current	(POTX, POTY)	I_{pot}	500	—	—	μA
Input Impedance	(EXT IN)	R_{in}	100	150	—	kΩ
Audio Input Voltage	(EXT IN)	V_{in}	5.7 —	6 0.5	6.3 3	VDC VAC
Audio Output Voltage	(AUDIO OUT; 1 kΩ load, volume=max) One Voice on: All Voices on:	V_{out}	5.7 0.4 1.0	6 0.5 1.5	6.3 0.6 2.0	VDC VAC VAC
Power Supply Current	(V_{DD})	I_{DD}	—	20	25	mA
Power Supply Current	(V_{CC})	I_{CC}	—	70	100	mA
Power Dissipation	(Total)	P_D	—	600	1000	mW

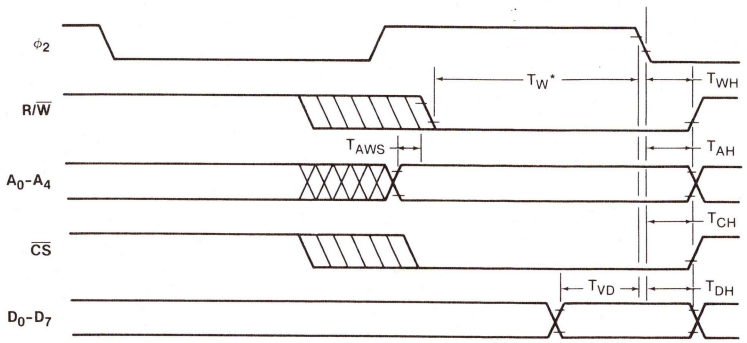
6581 SID-TIMING



*T_{ACC} wird nach dem letzten Auftreten von Ø2, $\overline{CS}$, A_0–A_4 gemessen.

LESEZYKLUS

SYMBOL	NAME	MIN	TYP	MAX	UNITS
T_{CYC}	Clock Cycle Time	1	—	20	µs
T_C	Clock High Pulse Width	450	500	10,000	ns
T_R, T_F	Clock Rise/Fall Time	—	—	25	ns
T_{RS}	Read Set-Up Time	0	—	—	ns
T_{RH}	Read Hold Time	0	—	—	ns
T_{ACC}	Access Time	—	—	300	ns
T_{AH}	Address Hold Time	10	—	—	ns
T_{CH}	Chip Select Hold Time	0	—	—	ns
T_{DH}	Data Hold Time	20	—	—	ns



*T_W wird nach dem letzten Auftreten von Ø2, $\overline{CS}$, $\overline{R/W}$ gemessen.

SCHREIBZYKLUS

SYMBOL	NAME	MIN	TYP	MAX	UNITS
T_W	Write Pulse Width	300	—	—	ns
T_{WH}	Write Hold Time	0	—	—	ns
T_{AWS}	Address Set-up Time	0	—	—	ns
T_{AH}	Address Hold Time	10	—	—	ns
T_{CH}	Chip Select Hold Time	0	—	—	ns
T_{VD}	Valid Data	80	—	—	ns
T_{DH}	Data Hold Time	10	—	—	ns

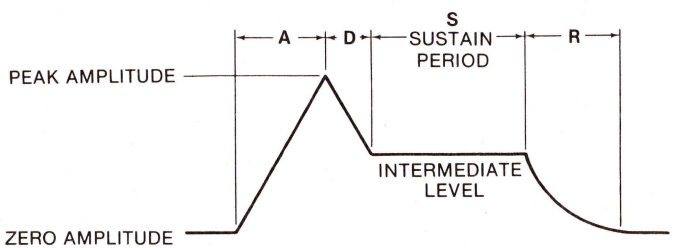
SID TONLEITER

Im Anhang E sind alle Werte aufgelistet, die in die Frequenzregister eingeschrieben werden müssen, um die Töne einer "wohltemperierten" Tonleiter zu erhalten. Diese besteht aus einer Oktave mit 12 Halbschritten: C, D, E, F, G, A, H, C und C#, D#, F#, G#, A#. Die Frequenz jedes Halbtones läßt sich durch Multiplikation der Frequenz des vorigen Halbtones mit der 12. Wurzel aus 2 errechnen. Der Tabelle liegt ein Systemtakt von 1,02 MHz zugrunde. Für andere Taktfrequenzen muß man die bei den Frequenzregistern angegebene Umrechnung anwenden. Die angegebene Stimmung bezieht sich auf A4 = 440 Hz. Es ist möglich, eine andere Stimmung zu verwenden oder diese Tonfolge umzustellen.
Obwohl dies eine einfache und schnelle Methode ist, die Tonleiter zu programmieren, werden allein zur Speicherung dieser Tabelle 192 Bytes benötigt. Diese Verschwendung des Speicherplatzes kann durch einen Algorithmus umgangen werden, mit dem die Notenwerte berechnet werden können. Da eine Oktave die Verdoppelung der Frequenz bedeutet, brauchen nur die 12 Notenwerte einer Oktave gespeichert zu werden. Wenn diese 12 Eingaben (24 Bytes) aus den Werten für die 8. Oktave bestehen (C7–H7), kann der Wert für jede beliebige Note errechnet werden, indem die Frequenz des entsprechenden Tones der 8. Oktave für jede Oktave Unterschied einmal durch 2 geteilt wird. Eine Division durch 2 ist in binärer Darstellung eine Verschiebung um ein Bit nach rechts. Deshalb kann die Berechnung durch eine einfache Routine durchgeführt werden. Obwohl die Frequenz von H7 von dem Oszillator nicht gebildet werden kann, sollte sie zur Berechnung in die Tabelle aufgenommen werden.
Für jeden Ton muß nun festgelegt werden, um welchen Halbton es sich handelt und in welcher Oktave er erklingen soll. Da man 4 Bit braucht, um 1 von 12 Halbtönen zu wählen, und 3 Bit, um eine von 8 Oktaven zu bestimmen, reicht ein Byte aus. Die unteren 4 Bit bestimmen z. B. den Halbton (sie adressieren einen Platz der Tabelle) und die oberen 4 Bit, um wieviel Stellen der Tabellenwert nach rechts verschoben werden muß.

SID HÜLLKURVENGENERATOR

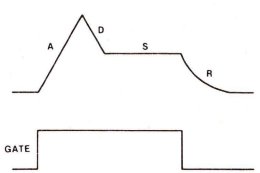
HÜLLKURVENGENERATOR

Der vierteilige ADSR (Attack, Decay, Sustain, Release) hat sich in der elektronischen Musik als optimaler Kompromiß zwischen Flexibilität und einfacher Bedienung erwiesen. Passende Wahl der Parameter erlaubt es, eine Vielzahl von Instrumenten nachzuahmen.
Die Geige ist ein gutes Beispiel für ein Instrument mit lang anhaltendem Ton: Er schwillt langsam an, erreicht eine Spitzenlautstärke und fällt dann auf einen niedrigeren Wert ab. Der Geiger kann diesen Ton lange halten, um ihn dann langsam ausklingen zu lassen. Ein "Schnappschuß" dieser Hüllkurve zeigt dieses Bild:



Diese Hüllkurve kann folgendermaßen nachgebildet werden:

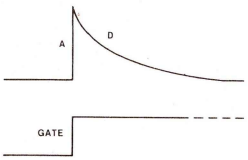
ATTACK:	10 ($A)	500 ms
DECAY:	8	300 ms
SUSTAIN:	10 ($A)	
RELEASE:	9	750 ms



Man beachte, daß der Ton solange anhält, bis das Gatebit zurückgesetzt wird. Mit wenigen Änderungen kann diese Hüllkurve für Blech- und Holzblasinstrumente und alle Streichinstrumente verwendet werden.
Eine ganz andere Hüllkurve besitzen Schlag- und Tasteninstrumente. Die Hüllkurve von Schlaginstrumenten wird von einem nahezu augenblicklichen Anstieg und einem darauf folgenden Abfall bestimmt, diese Instrumente können den Ton nicht auf einer konstanten Lautstärke halten. Eine Trommel erreicht in dem Moment, in dem sie angeschlagen wird, ihre volle Lautstärke, um dann schnell auszuklingen.

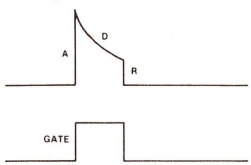
Die typische Hüllkurve eines Beckens wird hier gezeigt:

ATTACK:	0	2ms
DECAY:	9	750ms
SUSTAIN:	0	
RELEASE:	9	750ms



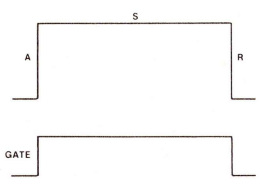
Man beachte, daß der Ton vollkommen ausklingt, obwohl das Gatebit nicht zurückgesetzt wird.
Der Amplitudenverlauf von Klavieren ist komplizierter, er kann aber mit dem ADSR leicht erzeugt werden. Der Ton erreicht seine volle Lautstärke, wenn die Taste angeschlagen wird, und beginnt dann abzuschwellen. Wenn die Taste losgelassen wird, wird der Ton durch die Mechanik abgedämpft. Diese Hüllkurve ist hier dargestellt:

ATTACK:	0	2 ms
DECAY:	9	750 ms
SUSTAIN:	0	
RELEASE:	0	6 ms



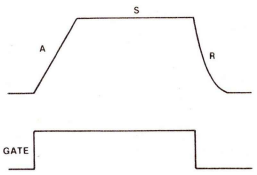
Man beachte, daß der Ton abklingt, bis das Gatebit zurückgesetzt und dann abgestellt wird.
Die einfachste Hüllkurve ist die einer Orgel: Solange eine Taste gedrückt ist, hat der Ton volle Lautstärke und wird sofort abgestellt, wenn die Taste wieder losgelassen wird.
Diese Hüllkurve ist hier dargestellt:

ATTACK:	0	2 ms
DECAY:	0	6 ms
SUSTAIN:	15 ($F)	
RELEASE:	0	6 ms



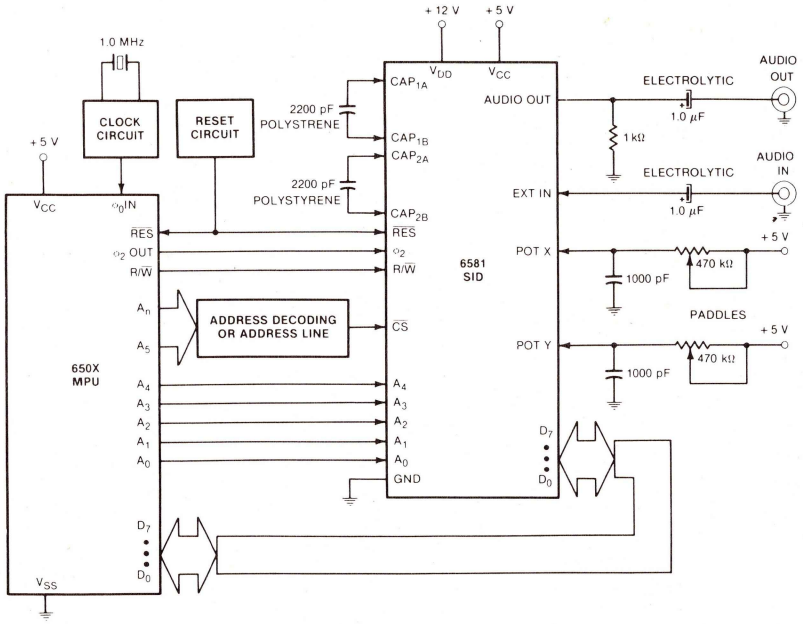
Die wirkliche Stärke des SID liegt aber in der Erzeugung künstlicher Klänge. Der ADSR kann Hüllkurven erzeugen, die bei keinem Instrument vorkommen. Ein gutes Beispiel ist hierfür die "Rückwärts"-Hüllkurve. Sie wird von einem langsamen Anstieg und einem scharfen Abfall bestimmt, was so klingt, als hätte man das Instrument auf Tonband aufgenommen und würde die Aufnahme rückwärts abspielen. Sie sieht folgendermaßen aus:

ATTACK:	10 ($A)	500 ms
DECAY:	0	6 ms
SUSTAIN:	15 ($F)	
RELEASE:	3	72 ms



Viele bemerkenswerte Klänge entstehen, wenn der Hüllkurvenverlauf des einen Instrumentes mit dem Klang eines anderen kombiniert wird. Dadurch entstehen Klänge, die bekannten Instrumenten ähneln, aber irgendwie fremd klingen. Da Klänge im allgemeinen subjektiv empfunden werden, muß man mit verschiedenen Klangfarben und Hüllkurven experimentieren, bis man den gewünschten Klang erhält.

TYPISCHE 6581/SID-ANWENDUNG



ANHANG P

GLOSSAR

ADSR	Anstieg-/Abkling-/Halte-/Abfallhüllkurve
ATTACK	Rate, mit der eine Musiknote die Spitzenlautstärke erreicht (Anstieg)
Binär	Zahlensystem mit der Basis 2
Boole'scher Operator	Logischer Operator
Byte	Speicherplatz
CHROMA-Rauschen	Farbverzerrung
CIA	Komplex-Interface-Adapter
DDR	Datenrichtungsregister
DECAY	Rate, mit der eine Musiknote von der Spitzenlautstärke bis zum Haltepegel abfällt (Abklingen)
Dezimal	Zahlensystem mit der Basis 10
e	Mathematische Konstante (ca. 2,71828173)
Hüllkurve	Lautstärkenkontur einer Note über einen bestimmten Zeitraum
FIFO	Zuerst eingegeben/Zuerst ausgeben
Hexadezimal	Zahlensystem mit der Basis 16
Ganze Zahl	Zahl ohne Dezimalpunkt
Jiffy-Uhr	Hardware-Intervall-Timer
NMI	Nicht maskierbare Unterbrechung
Oktal	Zahlensystem mit der Basis 8
Operand	Parameter
OS	Betriebssystem
Pixel	Auflösepunkt auf dem Bildschirm
Warteschlange	Einzel-Dateileitung
Register	Besonderer Speicherplatz
RELEASE	Rate, mit der eine Musiknote vom Haltepegel bis auf die Null-Lautstärke abfällt (Abfall)
ROM	Nur-Lesespeicher
SID	Sound-Interface-Vorrichtung
Vorzeichenzahlen	Positive oder negative Zahlen
Index	Indexvariable
SUSTAIN	Lautstärkepegel zum Halten einer Musiknote
Syntax	Programm-Satzstruktur
Abschneiden	Auslassen (nicht gerundet)
VIC-II	Video-Interface-Chip
Video-Bildschirm	Fernsehgerät

INDEX

6566/6567 Funktionsweise, 441
6581 SID, Merkmale von, 465
ABS, 35
ACPTR, 270
ADC, 231
ADDRESS ENABLE CONTROL, 403
ADRESSBUS, 402, 443
AND, 15, 35, 231
ACS, 37
ASCII-Code, 371
ASL, 232
ATN, 37
ATTACK, 182 ff.
ATTACK/DECAY, 457
Addition, 10
Adressierart, 219
Adressierung, Zero-Page-, 403
Adressierung, absolut-indirekte, 405
Adressierung, absolute, 403
Adressierung, implizierte, 403
Adressierung, indirekt-indizierte, 404
Adressierung, indiziert-indirekte, 404
Adressierung, indizierte Zero-Page-, 404
Adressierung, indizierte absolute, 404
Adressierung, relative, 404
Adressierung, unmittelbare, 403
Akkumulator, 211
Anführungszeichen, XI, 70, 331
Anweisungs-Adressierarten, 228
Anweisungssatz, 405
Anweisungssatz MCS6510, 228
Anweisungssyntax, XI
Anwendungshinweise, XII
Assembler, 307
Ausdruck, 10
Ausgabeport, 215

BAD DATA, 392
BAD SUBSCRIPT, 392
BASIC-Schlüsselwörter, XI
BCC, 232
BCS, 232
BEQ, 233
BIT, 233
BIT 4, 456
BIT 5, 456
BIT 6, 457
BIT 7, 457
BMI, 233
BNE, 234
BPL, 234
BRK, 234
BVS, 235
Bandpaßfilter, 198
Basic-Interpreter, 2, 16
Basic-Schlüsselwort, 31, 35
Basic-Schlüsselwörter, Abkürzung der, 366
Basic-Zeichensatz, 3
Befehlsregisterbelegung, 344
Betriebssystem, 2, 208, 264
Bildschirm-Anzeigecode, 368
Bildschirm-Code, 368
Bildschirm-Editor, 2, 12, 23, 94
Bildschirm-Rollen, 128
Bildschirm-Zeichenfarbe-Kombination, 151
Bildschirmausgabe, 330
Bildschirmcodes, 2
Bildschirmeditor, 94
Bildschirmlöschen, 149
Bildschirmspeicher, 102
Bit Map Modus, 100, 432
Bit-Map-Modus, Mehrfarben-, 122
Bit-Map-Modus, Standard-, 122
Bit-Mapping, 121
Boole'sche Wahrheitstabelle, 14

CAN'T CONTINUE, 392
CHAREN, 257
CHIP SELECT, 443
CHKIN, 271
CHKOUT, 272
CHR**$**, 38
CHR$-Code, 371
CHRIN, 273
CHROUT, 274
CINT, 276
CIOUT, 275
CLALL, 277
CLC, 235
CLD, 236
CLI, 236
CLOCK OUT, 443
CLOSE, 38, 277
CLR, 39
CLRCHN, 278
CLV, 236
CMD, 39
CMP, 237
COMMODORE CP/M^{R}, 362 ff.
CONT, 40
COS, 41
CPX, 237
CPY, 237
Complex Interface Adapter 6526, 411
Control Port 1, 387
Control Port 2, 387

Cursor, 71

DATA, 26, 42, 306
DATASSETTE™, 185, 256
DATENBUS, 443
DEC, 238
DECAY, 182 ff.
DEF FN, 42
DEVICE NOT PRESENT, 392
DEX, 238
DEY, 238
DIVISION BY ZERO, 392
DMA-Leitung, 361
DIM, 43
Datenumsetzung, 18
Direkt-Modus, 3
Disketten-Datenspeicherung, 335
Division, 11
Drehregler, 339
Dreieckswelle, 193
Dreieckswellenform, 202
Drucker-Steuerzeichencodes, 333

END, 44
EOR, 239
EXP, 45
EXTRA IGNORED, 392
Eckige Klammern, XI
Ein-/Ausgabe, 258
Ein-/Ausgabeanordnung, 317
Einfügemodus, 73
Einführung, IX
Eingabeanweisung, 18
Eingabeport, 212
Empfangspuffer, 350
Erweiterte Farbe, Betriebsart, 431
Erweiterter Hintergrundfarbmodus, 120
Erweiterungsanschluß, 359
Exponent, 6
Exponentialberechnung, 12

FILE NOT FOUND, 392
FILE NOT OPEN, 392
FILE OPEN, 392
FN, 45
FOR, 46
FORCE LOAD, 422
FORMULA TOO COMPLEX, 392
FRE, 47
FREQUENZ LOW/FREQUENZ HIGH, 455
Farbspeicher, 103
Farbspeicherbelegung, 375
Farbspeichermappe, 374
Farbsteuerung, 71
Fehlermeldungen, 309, 352
Felder, 9, 26
Filtereinstellung, 379
Filterregister, 460
Flag, 223
Fremde Basic-Programme – COMMODORE 64 Basic, 390

GET, 22, 23, 48
GETIN, 279
GOSUB, 26, 49
GOTO, 51
Ganze Zahl, 4, 7
Generator 1, 455
Gleitpunktzahl, 4, 7, 18
Glossar, 474
Graphikmöglichkeiten, 149
Graphikübersicht, 100
Graphikzeichen, 109
Graphikzeichen, Lage der, 101

HIRAM, 257
Handshaking, 421
Hexadezimaldarstellung, 214
Hochpaßfilter, 199
Hüllkurvengeber, 200
Hüllkurvengenerator, 194

I/O-PORT, 403
IF, 51
ILLEGAL DIRECT INPUT, 392
ILLEGAL QUANTITY, 392
INC, 239
INPUT, 19
INPUT, 53
INPUT-MODE, 423
INT, 54
INTERRUPT, 443
INTERRUPT CONTROL, 426
INTERRUPT REQUEST, 402
INX, 239
INY, 240
IOBASE, 280
IOINIT, 281
IRQ, 305
Index, 9
Indexregister, 223
Indexregister X, 211
Indexregister Y, 212
Indirekt indiziert, 221
Indizieren, 221
Indiziert indirekt, 222
Interface RS-232, 341
Interrupt-Aktivierungsregister, 150
Interrupt-Statusregister, 149
Interruptregister, 440
Intervall-Timer, 421

JMP, 240
JSR, 240, 266

Kanal RS-232, 342
Kernal, 2, 264
Kernal-Routine, 268 ff.
Kollision zwischen Sprites und Daten, 144
Kollision zwischen einzelnen Sprites, 144
Kollisionserkennung, 144
Komprimieren, 24, 155
Kontrollregister, 256, 455

LDA, 216, 241
LDX, 241
LDY, 242
LEFT**$**, 55
LEN, 55
LESEN (TIMER), 423
LET, 56
LIST, 56
LISTEN, 275, 281
LOAD, 57, 282, 392
LOG, 59
LORAM, 257
LSR, 242
Lautstärkeregelung, 184
Lese-Timing-Diagramm, 417
Lichtgriffel, 336, 341
Light Pen, 439
Listener, 356
Literaturverzeichnis, 380
Logische Operatoren, 13

MEMBOT, 283
MEMTOP, 284
MID**$**, 59
MOB, 434 ff.
Magnetbandkassette, 333
Mantisse, 6
Maschinencode, 209
Maschinensprache, 208
Maschinensprache + Basic, 304
Maschinensprache-Monitor, 307
Maschinensprache-Programme, 213
Maschinensprache-Routine, 306
Mathematische Funktionen, abgeleitete, 386
Matrize, 26
Mehrfarben-Bit-Mapping, 127
Mehrfarben-Modus, 115, 135
Mehrfarbige Graphiken, 115
Mikroprozessor 6510, 397
Mikroprozessor-Modul Z-80, 362
Modul-Steckplatz, 388
Monitor 45, 213, 224, 307
Multiplikation, 11
Musiknotenwerte, 376
Musiksynthesizer, 182

NEW, 60
NEXT, 61
NEXT WITHOUT FOR, 392
NOP, 242
NOT, 14, 62
NOT INPUT FILE, 393
NOT OUTPUT FILE, 393
Normal-Modus, 118
Numerische Variable, 19

ON, 62
ONE SHOT/CONTINUOUS, 422
OP-Schlüssel, 408
OPEN, 63, 285, 331
OR, 14, 66
ORA, 243
OUT OF DATA, 393
OUT OF MEMORY, 393
OVERFLOW, 393
Oberwelle, 193 ff.
Operator, 10

PB ON/OFF, 422
PEEK, 67
PHA, 243
PHP, 243
PLA, 244
PLOT, 286
PLP, 244
POKE, 67
POS, 68
PRINT, 68, 330
PRINT#, 74, 331
PW LO/PW HI, 455
Peripheriegerät, COMMODORE-, 362
Pinbelegung 6526, 418
Pixel, 121
Port-Pin-Beschreibung, 352
Positionierung, horizontale, 139
Positionierung, vertikale, 138
Priorität der Operationen, 15, 16
Programm-Modus, 4
Programmierbare Zeichen, 108
Programmieren von Zahlen und Variablen, 4
Programmiertechniken, 18
Programmzähler, 212
Prozessor-Schnittstelle, 443
Puffer, 92

QUOTE-Modus, 95

RAMTAS, 287
RDTIM, 287
READ, 26, 75
READ/WRITE, 403, 443
READST, 288
REDIM'D ARRAY, 393
REDO FROM START, 393
RELEASE, 182 ff.

REM, 21, 76
RESET, 402, 440
RESTOR, 289
RESTORE, 77
RETURN, 78, 331
RETURN WITHOUT GOSUB, 393
RIGHT$, 78
RING MOD, 456
RND, 79
ROL, 244
ROR, 245
RTI, 245
RTS, 245
RUN, 80
Rasterregister, 149, 440
Rauschgenerator, 200
Rauschwellenform, 203
Rechenausdruck, 10
Rechenoperation, 10
Rechteckwelle, 194
Refresh, 440
Register, 211
Ringmodulation, 205

SAVE, 80, 289
SBC, 246
SCHREIBEN (VORTEILER), 423
SCNKEY, 291
SCREEN, 291
SDR, 425
SEC, 246
SECOND, 292
SED, 246
SEI, 247
SERIAL ATN IN/OUT, 358
SERIAL CLK IN/OUT, 359
SERIAL DATA IN/OUT, 359
SERIAL SRQ IN, 357
SETLFS, 293
SETMSG, 294
SETTIM, 295
SETTMO, 296
SETNAM, 295
SGN, 82
SID, 450
SID-Hüllenkurvengenerator, 471
SID-Kontrollregister, 453
SID-Registerbelegung, 454
SID-Timing 6581, 468
SID-Tonleiter, 470
SIN, 82
SPC, 27, 83
SQR, 83
STA, 247
START/STOP, 422
STATUS, 84
STEP, 85

STOP, 86, 297
STR$, 86
STRING TOO LONG, 393
STX, 247
STY, 248
SUSTAIN, 182 ff.
SUSTAIN/RELEASE, 468
SYNC, 456
SYNTAX, 393
SYS, 87
SYS X, 304
Sägezahnwelle, 191
Schlüsselwort, 30
Schlüsselwortabkürzung, 24
Schrägstrich, XI
Schreib-Timing-Diagramm 6526, 416
Scrolling, 439
Sekundäradresse, 356
Serieller Bus, 356
Serieller Port, 425
Sonderzeichen, 73
Sound Interface Device 6581, 450
Speicher-Konfiguration 6510, 410
Speicheranforderung, 409
Speicherbelegung, 210, 259, 308
Speichermappe, 258
Speicherplatz, 210
Speicherverwaltung, 256
Spiegelung, 105
Spiele-Port, 336
Spitze Klammer, XI
Sprite, 100
Sprite, vergrößert, 136
Sprite-Anzeigepriorität, 143
Sprite-Erstellung, 162 ff.
Sprite-Pointer, 133
Sprite-Positionierung, 137, 139, 143
Sprite-Programmierung, 139
Sprite-Zeichnen, 144
Spritedefinition, 131
Spritepriorität, 160
Sprites, 131
Sprungtabelle, 264
Stack, 219
Stackpointer, 212
Standard-Bit-Mapping, 122
Standardzeichen-Betriebsart, 430
Standardzeichenmodus, 107
Stapel, 220
Stapelzeiger, 212, 265
Statusregister, 212
Statusregister RS-232, 348
Steckerbelegung, 387
Steuerknüppel, 336
Steuerregister CRA/CRB, 427
Steuerregisterbelegung, 343
Stimme, 459

String, 7, 17
Stringkonstante, 7
Stringvariable, 7, 19
Subtraktion, 11
Symbolbeschreibung, XII
Synchronisation, 205
Syntax, XI
Syntax-Format, XI
System-Ram-Vektor, 301

TAB, 27, 87
TAKT, 402
TALK, 298
TAN, 88
TAX, 248
TAY, 248
TEST, 456
TIME, 88
TIME**$**, 89
TKSA, 299
TOD, 423
TOGGLE/PULSE, 422
TSX, 249
TXA, 249
TXS, 249
TYA, 249
TYPE MISMATCH, 393
Talker, 356
Tastatur, 92
Tiefpaßfilter, 198
Timeout-Flag, 296
Timing-Charakteristiken 6526, 419
Token, 211
Tonfrequenzen, 184
Tongenerator, 182

UDTIM, 299
UNDEF'D FUNCTION, 393
UNDEF'D STATEMENT, 393
UNLST, 300
UNTLK, 301
USR, 89
USR(X), 304
Überprüfung, 223
Übertragungspuffer, 350
Unterlegte Zeichen, 71
Unterprogramm, 224
Unterstreichung, XI
User Port, 352, 389

VAL, 90
VECTOR, 301
VERIFY, 90, 393
VIC-Chip-Registerbelegung, 383
Variable, 18, 19, 25
Variablenname, 8
Vergleichsoperatoren, 12
Verkleinern der Programmzeilennummer, 25
Verschachtelung, 15
Verschieben, kontinuierliches, 128
Verzweigung, 223
Video-Bank, 101
Video-Interface-Controller 6566/6567, 429
Videoanschluß, 444
Vibrato, 200

WAIT, 91
Wellenform, Ändern der, 190

Zeichenanzeige, 100
Zeichendarstellungsmodus, 429
Zeichendefinition, 107
Zeichenketten, 4, 514
Zeichenkettenoperationen, 17
Zeichenspeicher, 103
Zero-Page, 219
Zero-Page-Adressen, 351

WAS IST ALLES ENTHALTEN?

Unser komplettes „BASIC-Lexikon" umfaßt BASIC-Befehle, Anweisungen und Funktionen in alphabetischer Reihenfolge. Wir haben eine Übersicht erstellt, in der alle Wörter und ihre Abkürzungen enthalten sind. In dem folgenden Abschnitt werden die einzelnen Begriffe genau definiert und anhand von Beispielprogrammen ihre Anwendung beschrieben.

Wenn Sie eine Einführung in die Anwendung der Maschinensprache für BASIC-Programme benötigen, wird für Sie unsere Übersicht hilfreich sein.

Ein leistungsstarker Bestandteil des Betriebssystems aller COMMODORE-Computer wird KERNAL genannt. Hierdurch wird sichergestellt, daß alle Programme, die Sie heute schreiben, auch noch auf den COMMODORE-Computern von morgen laufen können.

Der Abschnitt über Ein-/Ausgabeprogrammierung zeigt Ihnen, wie Sie Ihren Computer voll nutzen können. In diesem Abschnitt werden die möglichen Ergänzungen beschrieben – angefangen bei Lichtstiften und Joysticks bis hin zu Diskettenstationen, Druckern und Zusatzgeräten für Telekommunikation (Modems).

Wir zeigen Ihnen, wie man SPRITES und Sonderzeichen programmiert. Sie werden lernen, wie man Lauf-Bilder in hochauflösender Farbgraphik erzeugen kann.

Wir eröffnen Ihnen die Welt der Musik-Synthese und zeigen Ihnen, wie Sie eigene Songs schreiben und Klangeffekte mit dem eingebauten Synthesizer erzielen können.

Dem erfahrenen Programmierer zeigen wir, wie er den COMMODORE 64 mit weiteren anspruchsvollen Sprachen nutzen kann.

Das Programmierhandbuch COMMODORE 64 soll also ein nützliches Werkzeug sein, damit Ihnen das zukünftige Programmieren auch wirklich Spaß macht.



Commodore

Commodore GmbH
Lyoner Straße 38
D-6000 Frankfurt/M. 71

Commodore AG
Aeschenvorstadt 57
CH-4010 Basel

Commodore GmbH
Kinskygasse 40–44
A-1232 Wien

Nachdruck, auch auszugsweise, nur mit schriftlicher Genehmigung von Commodore.
Artikel-Nr. 556420/1.85 Änderungen vorbehalten ISBN 3-89133-000-6