Baloui • Brückmann • Englisch • Felt • Gelfand • Gerits • Krsnik

DAS NEUE COMMODORE 64 INTERN BUCH

Das ausführliche Referenzhandbuch zum C64 bietet Praxis-Know-how von A bis Z. Alles zur Hardware, Programmierung und täglichen Arbeit in einem Band.

DATA BECKER

Baloui
Brückmann
Englisch
Felt
Gelfand
Gerits
Krsnik

DAS NEUE COMMODORE 64 INTERN BUCH

DATA BECKER

Copyright © 1990 by DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf 1

1. Auflage 1990

Umschlaggestaltung Werner Leinhos

Text verarbeitet mit Word 4.0, Microsoft

Ausgedruckt mit Hewlett Packard LaserJet II

Druck und buchbinderische Verarbeitung Mohndruck, Gütersloh

ISBN 3-89011-307-9

Inhaltsverzeichnis

1.	**Der BASIC-Interpreter**	**11**
1.1	Hex-, Binär- und Dezimalsystem	11
1.2	Logische Verknüpfungen	14
1.3	Der Aufbau einer BASIC-Zeile	18
1.4	Ablage von Variablen	22
1.5	Wie erweitert man BASIC?	24
1.6	Übergabe von BASIC-Parametern über USR	30
1.7	Sprungvektoren und Autostart	39
1.8	Die Adressen der BASIC-Routinen	45
1.9	Fließkommaarithmetik	50
1.10	Der Virus-Killer	59
1.11	Der BASIC-Kompaktor	61
2.	**Der Aufstieg zu Assembler**	**73**
2.1	Maschinensprache und Assembler	78
2.2	Dualsystem und Hexadezimalsystem	82
2.3	Der Hauptspeicher	87
2.4	Speicherorganisation	88
2.5	Die Speicheraufteilung	91
2.6	Programmieren mit dem Monitor	94
2.6.1	Assembler-Programme mit einem Monitor eingeben	94
2.6.2	Speicherung eines Assembler-Programms im Rechner	97
2.6.3	Programme speichern und laden	100
2.7	Befehlsbearbeitung durch die CPU	102
2.8	Die Adressierungsarten	103
2.9	Programmschleifen	124
2.9.1	Schleifen bilden mit BNE	124
2.9.2	Verzögerungsschleifen	129
2.9.3	Effektiver Einsatz von Schleifen	133
2.10	Die wichtigsten Routinen des Betriebssystems	138
2.11	Schleifen "unter die Lupe genommen"	152
2.12	"Sprungweite" der Branch-Befehle	161
2.13	Rechnen in Maschinensprache	174
2.14	Die Vergleichsbefehle	182
2.15	Schleifen und Vergleichsbefehle	185

2.16 Stapeloperationen 209
2.17 Windowing 225
2.18 Logische Verknüpfung 239
2.19 Die Schiebe- und Rotierbefehle 247
2.20 Wie Basic-Programme abgelegt werden 277
2.21 Ein großes Programmprojekt: Die INPUT#-Routine 309
2.22 Der Maschinensprachemonitor 324

3. Die Grafik und ihre Programmierung 331

3.1 Der Video Interface Chip (VIC) 331
3.2 Das Video-RAM 332
3.3 Der Zeichengenerator 335
3.4 Das Farb-RAM 336
3.5 Extended Background Color Mode 340
3.6 Der Multicolormodus 341
3.7 Sprites und ihre Programmierung 343
3.8 Der Grafikbildschirm 351
3.9 Der Multicolorgrafikbildschirm 354
3.10 Wie wende ich das Grafikhilfsprogramm an? 363
3.11 Interruptprogrammierung 365
3.12 Feinscrolling 372
3.13 Screen-Scrolling 374
3.14 Registerbeschreibung des VIC 384
3.15 Pinbeschreibung des VIC 6567 387

4. Der Soundcontroller 389

4.1 Die Frequenz 389
4.2 Wellenform 391
4.3 Hüllkurve 393
4.4 Filter 398
4.5 Tongenerator 3 400
4.6 Der Analog/Digitalwandler 401
4.6.1 Die Handhabung des A/D-Wandlers 401
4.7 Registerbeschreibung des SID 402

5. Die CIAs 411

5.1 Datenein- und -ausgabe von Maschinenprogrammen 411
5.1.1 Ein- und Ausgabe von einzelnen Bytes 411
5.1.2 Ein- und Ausgabe über Peripheriegeräte 415
5.2 Die Technik der Datenspeicherung - LOAD und SAVE 418
5.2.1 Datenspeicherung auf Kassette 418
5.2.2 Datenspeicherung auf Diskette 424
5.3 Die CIAs 430
5.4 Die E/A-Port 431
5.4.1 Tastaturabfrage 431
5.4.2 Joystick 437
5.4.3 Die Verwendung von Paddles 438
5.4.4 Maus 440
5.4.5 Die 64 Maus 1351 446
5.5 Timer 451
5.6 Echtzeituhr 454
5.7 Die Programmierung der RS-232 Schnittstelle 458
5.8 Der IEC-BUS 465
5.8.1 Begriffsbestimmungen 467
5.8.2 Geräteadressen 468
5.8.3 Sekundäradressen 469
5.8.4 Die Systemvariable ST 470
5.8.5 Adressierung 471
5.8.6 Der Datentransfer 473
5.8.7 Die Programmierung des IEC-Bus 475
5.9 Das serielle Schieberegister 478
5.10 Pinbelegung der CIA 479
5.11 Der User-Port 481
5.12 Registerbeschreibung der CIA 484

6. Das ROM-Listing 489

6.1 Nutzung des ROM-Listings 489
6.2 Verzeichnis der wichtigsten ROM-Routinen 492
6.3 Alphabetisches Verzeichnis der ROM-Routinen 496
6.4 Die Belegung der Zeropage 504

6.5 Die Speicheraufteilung des C64 530
6.5 Commodore 64 ROM-Listing 533

7. C64 Pflegen und Warten 799

7.1 Allgemeines zu diesem Kapitel 799
7.2 Der Bildausfall 799
7.3 Nur Bildschirm und Rahmen 801
7.4 Farbige Zeichen auf dem Bildschirm 802
7.5 Die Tastatur funktioniert nicht richtig! 803
7.6 Der Joystick funktioniert nicht! 803
7.7 Wenn er nicht richtig lädt! 803
7.8 Fehler, die nach längerem Betrieb auftreten 804
7.9 Das Herausnehmen von ICs 805
7.10 Wie stelle ich mir meine Tastatur strammer? 806
7.11 Wie baue ich einen RESET-Taster ein? 806
7.12 Das Testprogramm 807

10. Vergleich der Rechner 811

Anhang 815

Anhang A: Der Diskmonitor 815
Anhang B: Glossar 824
Anhang C: Stichwortverzeichnis 828

1. Der BASIC-Interpreter

1.1 Hex-, Binär- und Dezimalsystem

Bevor Sie sich den weiteren Kapiteln dieses Buches, sei es Grafikprogrammierung oder Aufbau einer BASIC-Zeile, zuwenden, ist es nötig, sich mit einigen wichtigen Grundlagen vertraut zu machen. Es handelt sich dabei um verschiedene Zahlensysteme, die wir erläutern wollen. Sollten Sie mit hexadezimalen und binären Zahlen bereits vertraut sein, so können Sie diesen Unterpunkt des ersten Kapitels überschlagen und mit Kapiteln 1.2 fortfahren.

Das Wort dezimal kommt aus dem Lateinischen und bedeutet: auf die Grundzahl zehn bezogen. Mit den zehn Ziffern unseres Dezimalsystems (0, 1, 2, 3 bis 9), läßt sich jede beliebige Zahl darstellen. Für die Zahlen von null bis neun reicht eine Ziffer aus. Wollen wir aber eine Zahl darstellen, die größer als neun ist, reicht eine Ziffer nicht mehr aus, und wir müssen weitere Stellen belegen. Dies sei hier an der Zahl dreizehn verdeutlicht. Die ersten zehn 'Elemente' werden zu einem Zehner zusammengefaßt, die drei restlichen Elemente bleiben als Einer stehen. Wir erhalten also einen Zehner und drei Einer. Damit haben wir die Zahl 13 im Dezimalsystem dargestellt. Reichen aber auch neun Zehnerstellen und neun Einerstellen nicht aus, um unsere Zahl darzustellen, müssen wir noch einen Übertrag hinzunehmen. Diese Stelle enthält die Hundertereinheiten unserer Zahl. Die Gliederung einer Dezimalzahl sieht also wie folgt aus: von rechts nach links stehen die Einer, Zehner, Hunderter und so weiter. Die Einerstelle kann aber auch als 10^0 geschrieben werden. Ebenso ist $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$ und so weiter. Als Beispiel für die Einheiten betrachten wir die Zahl 2134 einmal genauer:

2	1	3	4
Tausender	Hunderter	Zehner	Einer
1000	100	10	1
10^3	10^2	10^1	10^0

Die Zahl 2134 kann demnach auch folgendermaßen dargestellt werden:

$2*10^3+1*10^2+3*10^1+4*10^0 = 2*1000+1*100+3*10+4*1 = 2134$

Der Mensch bedient sich des Dezimalsystems, weil er zehn Finger hat und diese anfangs zum Rechnen benutzte. Mittlerweile sind fast alle Einheiten im Zehnersystem definiert. So ist 1 km = 1000 m, 1 t = 1000 kg oder 1 dm = 10 cm, um nur einige Beispiele zu nennen.

Der Computer hat aber keine zehn, sondern nur zwei "Finger", um zu zählen. Er erkennt nur, ob Strom ein- oder ausgeschaltet ist und verfügt daher nur über zwei Ziffern, 0 und 1. Demnach arbeitet er mit einem Zweiersystem, auch Binärsystem genannt. Analog zum Dezimalsystem kann hier nur bis eins gezählt werden, danach findet ein Übertrag statt. Diese Übertragsstelle, links neben der Einerstelle, hat den Wert $2^1 = 2$. Hier werden also die Zweiereinheiten abgelegt. Die nächsten Einheitenstellen entsprechen im Dezimalsystem den Werten 4, 8, 16, 32, 64 und 128. Eine solche Einheitenstelle ist die kleinste Information, die der Prozessor verarbeiten kann. Die einzelnen Kleinstinformationen werden Bit genannt und sind die Grundlage für jede Rechneroperation. Acht Bits werden jeweils zu einem Byte zusammengezogen, das dann höchstens den Wert 11111111 = 255 haben kann.

```
Bit:   7 6 5 4 3 2 1 0
       1 1 1 1 1 1 1 1
  = 128+64+32+16+8 +4 +2 +1 = 255
```

Größere Rechner können auch 16 Bits zu einem sogenannten "Word" zusammenfassen. Da der 6502 nicht über diese Fähigkeit verfügt, bedient man sich hier eines kleinen Tricks. Die 16 Bits werden jeweils in zwei Bytes aufgeteilt und hintereinander im Speicher abgelegt. Hierbei ist zu beachten, daß die Bits 0 bis 7, das sogenannte LOW-Byte, immer zuerst abgelegt werden. Danach kommen erst die Bits 8 bis 15, die das höherwertige Byte der Zahl darstellen. Dieses Byte nennt man HIGH-Byte. Mit 16 Bits kann man $2^{16} = 65535$ Zahlen darstellen, was genau

dem Adressbereich des C64 entspricht. Wie man sieht, ist es für Adressierungen sehr günstig, wenn man Zahlen in zwei Bytes darstellt. Um alle Einheiten zu nennen, muß an dieser Stelle noch erwähnt werden, daß ein Byte auch in zwei Einheiten zu je vier Bits unterteilt werden kann. Die ersten vier Bits nennt man LOW-, die zweiten vier Bits HIGH-Nibble des Bytes.

Nun wollen wir versuchen, mit Binärzahlen zu rechnen. Die Addition zweier Zahlen geschieht genau wie im Dezimalsystem auch: die einzelnen Stellen der Zahlen werden addiert. Sollte ein Übertrag entstehen, wird dieser zu der links folgenden Stelle hinzuaddiert. Hierzu ein Beispiel:

```
  189 = 10111101
+  42 = 00101010
-------------------
= 231 = 11100111
```

Um im Umgang mit Zahlen im Bezug auf Computer völlig vertraut zu werden, fehlt noch die Einsicht in das Hexadezimalsystem und natürlich die nötige Übung.

Das Hexadezimalsystem verfügt über 16 Ziffern, von denen die ersten zehn, von 0 bis 9, mit denen des Dezimalsystems identisch sind. Danach folgen als Ziffern A, B, C, D, E und F, die auf den ersten Blick etwas mysteriös erscheinen, da sie uns bisher nur als Buchstaben bekannt sind. Im Hexadezimalsystem werden diese Symbole jedoch benutzt, um Ziffern für die Werte 10 bis 15 darzustellen. Die dezimalen Werte der einzelnen hexadezimalen Ziffern können Sie der folgenden Tabelle entnehmen:

```
Hexadezimal: 0 1 2 3 4 5 6 7 8 9  A  B  C  D  E  F
   Dezimal: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Der Unterschied zum Dezimalsystem besteht darin, daß nach der Ziffer 9 kein Übertrag entsteht, sondern mit A weitergezählt wird. Stattdessen findet der Übertrag erst nach der Ziffer F statt. Mit einer Stelle läßt sich also maximal die Zahl 15 darstellen. Im Binärsystem würde man dazu vier Stellen benötigen.

Mit zwei Ziffern läßt sich maximal die Zahl FF = $15*16^1+15*16^0$ = 255 darstellen. Im Gegensatz zum Binärsystem, in dem man für diesen Wert acht Stellen braucht, genügen hier zwei. Um den gesamten Adressbereich ansprechen zu können, kommt man jetzt mit vier statt mit 16 Stellen aus. Der Vollständigkeit halber wollen wir hier noch auf das Rechnen mit Hexadezimalzahlen anhand eines Beispiels eingehen:

```
Dezimal   Binär      Hexadezimal

  133 = 10000101 =     85
+  42 = 00101010 =     2A
-------------------------------
 =175 = 10101111 =     AF
```

Sie sehen, daß sich auch hier am Rechenprinzip nichts geändert hat, wenn man von den sechs neuen Ziffern absieht. Mit einiger Übung werden Sie sowohl das Rechnen mit den Zahlensystemen, als auch das Umrechnen zwischen diesen als Selbstverständlichkeit ansehen. Für das Umrechnen zwischen Binär- und Hexadezimalsystem gibt es eine kleine Hilfe, die auch Sie sich zu Nutze machen können:

Wie schon erwähnt, lassen sich die acht Bits eines Bytes in zwei Nibbles unterteilen. Nehmen wir die Zahl 42 = 00101010, so ist das LOW-Nibble 1010 = 10 = $A. Das HIGH-Nibble hat den Wert 0010 = $2. Schreiben wir die beiden Nibbles, in der Reihenfolge HIGH- und LOW-Nibble, hintereinander, erhalten wir die Zahl $2A = 00101010. Wir haben also die zwei Stellen der hexadezimalen Zahl einzeln betrachtet und umgerechnet. Dadurch arbeiten wir nur noch mit Zahlen im Bereich von 0 bis 15, beziehungsweise von 0 bis F, was uns das Rechnen erheblich erleichtert.

1.2 Logische Verknüpfungen

Die logischen Verknüpfungen stammen aus der Booleschen Algebra, so genannt nach dem Mathematiker George Boole. Im BASIC hat man Zugriff auf die Verknüpfungen NOT, AND und

OR. In Assembler kann man den Befehl NOT nicht benutzen. Dort hat man dafür aber den Befehl EOR, mit dem man unter anderem auch die Wirkung von NOT erzielen kann.

Die erste Verknüpfung, die wir behandeln wollen, ist NOT. Auf deutsch übersetzt bedeutet NOT 'NICHT'. Wenn man weiß, daß sich eine logische Verknüpfung immer auf die einzelnen Bits eines Bytes bezieht, dürfte es nicht schwer fallen, sich vorzustellen, welche Wirkung dieser Befehl hat. Jedes Bit wird negiert, also einfach umgedreht. Wenn ein Bit 1 ist, wird es 0 und umgekehrt. NOT ist also keine Verknüpfung im eigentlichen Sinn, da es sich nur auf eine Zahl bezieht und keine zwei Zahlen miteinander verknüpft werden. Zu NOT nun ein Beispiel:

```
NOT 195
Ergebnis: 60
```

Das Betrachten des Bitmusters veranschaulicht das Ergebnis:

```
Zahl    11000011 = 195
NOT     00111100 =  60
```

Es ist zu beachten, daß sich dieses Zahlenbeispiel nur auf das Bitmuster bezieht. Da die Zahlen im BASIC zuerst in das Integerformat umgewandelt und dann erst negiert werden, ist das Ergebnis hier nicht dasselbe. In BASIC würden Sie -196 erhalten.

Jetzt kommen wir zu den Verknüpfungen, die auch in Assembler Verwendung finden. Als erstes wäre hier AND, zu deutsch UND, zu nennen,. Hierbei müssen jeweils beide zu verknüpfenden Bits gesetzt sein, damit das Bit im Ergebnis auch gesetzt ist. Ist eines der beiden Bits 0 und das andere 1, so wird das Ergebnisbit 0.

```
PRINT 31 AND 85
Ergebnis: 21
```

Auch hier verdeutlichen die Bitkombinationen das Ergebnis:

```
    1. Zahl 00011111 = 31
AND 2. Zahl 01010101 = 85
-----------------------------
   Ergebnis 00010101 = 21
```

Ein Anwendungsbeispiel für die AND-Verknüpfung ist zum Beispiel das Löschen einzelner Bits. Möchte man beispielsweise das letzte Bit eines Bytes löschen, ohne die anderen Bits zu verändern, so braucht man die entsprechende Zahl nur mit 11111110 = 254 zu verknüpfen.

```
          Zahl: 01101101
verknüpft mit: 11111110
-----------------------------
        ergibt: 01101100
```

In diesem Beispiel wurde das letzte Bit gelöscht, ohne die anderen Bits dabei zu verändern. Dieses System ist vor allem dann von Nutzen, wenn die einzelnen Bits eines Bytes verschiedene Funktionen übernehmen und nicht alle Bits bekannt sind.

Eine Umkehrung dieser Funktion, also das Setzen eines beliebigen Bits, kann man mit OR erreichen. Bei OR muß mindestens eines der jeweiligen Bits gleich 1 sein, damit das Ergebnisbit gesetzt wird. Es können natürlich auch beide Bits gesetzt sein. Um das letzte Bit einer Zahl zu setzten, muß man es nur mit 1 OR-verknüpfen. War es gelöscht, so wird es jetzt gesetzt, da 0 OR 1 = 1 ist. War es aber schon 1, wird der Wert beibehalten, da 1 OR 1 auch 1 ist.

Ein Beispiel könnte so aussehen:

```
          Zahl: 10010100
verknüpft mit: 00000001
-----------------------------
        ergibt: 10010101
```

Da die restlichen Bits mit 0 verknüpft werden, bleiben sie unverändert. Aus dem BASIC heraus muß man die betreffende Zahl mit 1 OR-verknüpfen, wenn man das letzte Bit setzen will. Möchte man Bit 2 setzen, muß man die Zahl 00000010 = 21 = 2 benutzen.

Während die bislang besprochenen Verknüpfungen zum Standard fast jeder Programmiersprache gehören, muß EXOR fast als 'Exot' unter den Verknüpfungen angesehen werden. EX steht dabei für EXklusiv oder NUR. Eine wahre Aussage erhält man, wenn zwei verschiedene Bits miteinander verknüpft werden, es muß also *genau* ein Bit gesetzt sein. Um eventuellen Verwirrungen vorzubeugen, muß an dieser Stelle eingefügt werden, daß in der Boolschen Algebra eine 1 als Ergebnis einer Verknüpfung eine wahre Aussage bedeutet. Eine 0 als Ergebnis steht hingegen für eine falsche Aussage.

Auf den ersten Blick erscheint das Anwendungsgebiet von EXOR sehr begrenzt, doch das ändert sich, wenn man ein wenig damit arbeitet. Verknüpft man zum Beispiel eine Zahl A mit einer zweiten Zahl B, so sind im Ergebnis die übereinstimmenden Bits gelöscht. Diese Anwendung eröffnet neue Möglichkeiten, vor allem in der Assemblerprogrammierung. Da eine Zahl A zweimal mit einer Zahl B verknüpft wieder die Zahl A ergibt, eignet sich EXOR auch hervorragend zum Codieren eigener Programme, um diese vor fremden Eingriffen zu schützen. Dabei muß das Programm in codierter Form abgespeichert sein, nachdem man es zuvor 'per Hand' mit einer bestimmten Zahl EXOR-verknüpft hat. Dem Hauptprogramm muß dann eine kleine Decodierroutine vorangestellt werden.

Hier sind die sogenannten Wahrheitstabellen der Booleschen Algebra noch einmal genau aufgeführt:

NOT	AND	OR	EXOR
0 = 1	0 0 = 0	0 0 = 0	0 0 = 0
1 = 0	0 1 = 0	0 1 = 1	0 1 = 1
	1 0 = 0	1 0 = 1	1 0 = 1
	1 1 = 1	1 1 = 1	1 1 = 0

1.3 Der Aufbau einer BASIC-Zeile

Der COMMODORE 64 verfügt über einen recht komfortablen BASIC-Interpreter, der auf dem des COMMODORE PET 2001 aufbaut. Dieser Interpreter wurde von der Firma Microsoft entwickelt und findet auch in der 3000er Serie von COMMODORE Verwendung. Da es sich auch hier um das BASIC mit der Versionsnummer 2.0 handelt, entsprechen sich die Funktionen der beiden Interpreter. Doch nun zur Funktionsweise des Interpreters.

Nach dem Einschalten befindet sich der Computer in einer Eingabewarteschleife, sofern kein Modul angeschlossen ist. Wie sich schon aus dem Wort entnehmen läßt, wartet der Rechner hier auf die Eingabe einer BASIC-Zeile, die mit RETURN abgeschlossen werden muß. Danach verzweigt er in eine Schleife, die für die Umwandlung in den Interpretercode zuständig ist. Dabei wird überprüft, ob es sich bei dem ersten Zeichen dieser Zeile um eine Zahl handelt. Ist dies nicht der Fall, so wird die Zeile im Direktmodus abgearbeitet. Handelt es sich jedoch um eine Zahl, wird diese als Zeilennummer angesehen. Existiert diese Zeilennummer schon, wird die betreffende BASIC-Zeile im Speicher gelöscht und der Rechner kehrt in die Eingabewarteschleife zurück. Normalerweise folgt nach der Zeilennummer jedoch ein Text, der dann vom BASIC-Interpreter in den Interpretercode umgewandelt und abgespeichert wird. Bei dieser Umwandlung wird jeder BASIC-Befehl in eine Codenummer übersetzt. Die entsprechenden Werte kann der Computer aus einer Tabelle ablesen, die wir auf einer der nächsten Seiten abgedruckt haben. Diese Werte, mit denen der Interpreter arbeitet, werden auch Tokens genannt. Texte, die etwa in Anführungsstrichen stehen, werden nicht in solche Tokens umgewandelt, sondern in normalem ASCII-Code (American Standart Code for Information Interchange) im Hexadezimalsystem abgelegt. Zum besseren Verständnis haben wir einige Beispielzeilen mit dem dazugehörigen Interpretercode abgedruckt, die wir noch weiter dokumentieren wollen.

```
BASIC-Zeile       Interpretercode

10 PRINT"HALLO"   0E 08 0A 00 99 22 48 41 4C 4C 4F 22 00
                   080E  0010 PR. "  H  A  L  L  O  "

20 REM TOKEN      1A 08 14 00 8F 20 54 4F 4B 45 4E 00
                   081A  0020 REM    T  O  K  E  N

30 END            20 08 1E 00 80 00 00 00
                   0820  0030 END
```

Fangen wir mit Zeile 10 an: Die ersten beiden Bytes stehen mit der BASIC-Zeile in keinem direkten Zusammenhang und können deshalb etwas verwirren. Es handelt sich hierbei um die Anfangsadresse der jeweils nächsten BASIC-Zeile. Diese Adresse liegt als LOW- und HIGH-Byte vor. Die nächste Zeile fängt in unserem Beispiel also bei $080E an. Hierbei sind wir davon ausgegangen, daß der BASIC-Start bei $0801 (2049) liegt, was nach dem Einschalten des Rechners normalerweise der Fall ist. Die nächsten beiden Bytes geben die Zeilennummer der BASIC-Zeile an. Schauen wir uns diese beiden Bytes an, so sehen wir, daß es sich um die Zeilennummer $000A handelt, die in das Dezimalsystem umgerechnet der Zahl 10 entspricht. Nun folgt der erste BASIC-Befehl, der in ein Token umgewandelt wurde. Der Befehl PRINT ist hier in den Codewert $99 (153) umgewandelt worden. Danach folgt die Zahl $22 (34), die dem Hochkommazeichen, auch Anführungsstriche genannt, entspricht. Der in Hochkommazeichen eingeschlossene Text wird nicht in Tokens umgewandelt, sondern als ASCII-Code abgelegt. Das Wort 'HALLO' bleibt also in seiner vollen Länge von fünf Bytes erhalten. Jetzt kommt wieder der Token für das Hochkommazeichen und im Anschluß die Zahl Null. Diese Null wird nach jeder BASIC-Zeile vom Interpreter automatisch angehängt, um das Ende der Zeile zu markieren.

In der zweiten Zeile wiederholt sich dieses Prinzip. Die ersten Bytes zeigen auf den Anfang der dritten Zeile, die sich in $081A (2074) anschließt. Der Codewert $8F (143) ist der Token für den BASIC-Befehl REM.

Der Text hinter REM wird, wie der in den Hochkommazeichen auch, als ASCII-Code abgelegt. Wie bei der ersten Zeile wird an das Ende eine Null angehängt.

Auch die dritte Zeile ist nach demselben System aufgebaut, weist jedoch zu den ersten beiden einen Unterschied auf. Die ersten beiden Bytes zeigen nicht auf den Anfang der nächsten BASIC-Zeile, da diese nicht existiert.

Die Bytes zeigen hinter das Ende der letzten Zeile. An diese Stelle setzt der Interpreter zu der Null für das Ende der Zeile noch zwei weitere Nullen. Hier befinden sich jetzt also drei Nullen, die für den Interpreter das Ende des gesamten BASIC-Programms markieren.

Bei der Umwandlung in den Klartext, also einem LIST, wandelt der Interpreter die Tokens in ASCII-Codes um. Diese Methode hat zwei wesentliche Vorteile.

Zum einen wird für ein Programm weniger Speicherplatz benötigt, zum anderen muß das Programm nicht bei jedem Ablauf erst umgewandelt werden, was zu einem schnelleren Programmablauf führt.

Hier nun die Tabelle der Befehlsworte und ihrer Tokens: Die Adresse hinter den Tokens gibt den Einsprung der jeweiligen Routinen im BASIC-Interpreter an, soweit dieses möglich ist.

Befehl	Token	Adresse	Befehl	Token	Adresse
END	$80 128	$A831	SPC(	$A6 166	-
FOR	$81 129	$A742	THEN	$A7 167	-
NEXT	$82 130	$AD1D	NOT	$A8 168	$AED4
DATA	$83 131	$A8F8	STEP	$A9 169	-
INPUT#	$84 132	$ABA5	+	$AA 170	$B86A
INPUT	$85 133	$ABBF	-	$AB 171	$B853
DIM	$86 134	$B081	*	$AC 172	$BA2B
READ	$87 135	$AC06	/	$AD 173	$BB12
LET	$88 136	$A9A5	^	$AE 174	$BF7B

Befehl	Token	Adresse	Befehl	Token	Adresse
GOTO	$89 137	$A8A0	AND	$AF 175	$AFE9
RUN	$8A 138	$A871	OR	$B0 176	$AFE6
IF	$8B 139	$A928	Größer	$B1 177	-
RESTORE	$8C 140	$A81D	=	$B2 178	-
GOSUB	$8D 141	$A883	Kleiner	$B3 179	-
RETURN	$8E 142	$A8D2	SGN	$B4 180	$BC39
REM	$8F 143	$A93B	INT	$B5 181	$BCCC
STOP	$90 144	$A82F	ABS	$B6 182	$BC58
ON	$91 145	$A94B	USR	$B7 183	$0310
WAIT	$92 146	$B82D	FRE	$B8 184	$B37D
LOAD	$93 147	$E168	POS	$B9 185	$B39E
SAVE	$94 148	$E156	SQR	$BA 186	$BF71
VERIFY	$95 149	$E165	RND	$BB 187	$E097
DEF	$96 150	$B3B3	LOG	$BC 188	$B9EA
POKE	$97 151	$B824	EXP	$BD 189	$BFED
PRINT#	$98 152	$AA80	COS	$BE 190	$E264
PRINT	$99 153	$AAA0	SIN	$BF 191	$E26B
CONT	$9A 154	$A857	TAN	$C0 192	$E2B4
LIST	$9B 155	$A69C	ATN	$C1 193	$E30E
CLR	$9C 156	$A65E	PEEK	$C2 194	$B80D
CMD	$9D 157	$AA86	LEN	$C3 195	$B77C
SYS	$9E 158	$E12A	STR$	$C4 196	$B465
OPEN	$9F 159	$E1BE	VAL	$C5 197	$B7AD
CLOSE	$A0 160	$E1C7	ASC	$C6 198	$B78B
GET/GET#	$A1 161	$AB7B	CHR$	$C7 199	$B6EC
NEW	$A2 162	$A642	LEFT$	$C8 200	$B700
TAB(	$A3 163	-	RIGHT$	$C9 201	$B72C
TO	$A4 164	-	MID$	$CA 202	$B737
FN	$A5 165	$B3F4	GO	$CB 203	-

Als Ergänzung muß noch hinzugefügt werden, daß das höchstwertige Bit, also Bit 7, der Tokens immer gesetzt ist. Daraus ergibt sich zwangsläufig, daß die Werte der Tokens immer größer als 127 sind.

Neben den Tokens finden noch folgende Werte im Interpreter Verwendung:

$00	Der Nullcode markiert Zeilen- oder Programmenden
$20 (32) bis 5F (95)	Diese Werte sind identisch mit dem ASC II-Code
$FF (255)	Codewert für die Zahl Pi

Betrachten wir die Wertebereiche der Codes, stellen wir fest, daß die Bereiche von (1) bis $1F (31), $60 (96) bis $7F (127) und $CC (204) bis $FE (254) vom Betriebssystem nicht benutzt werden. Sie eignen sich damit hervorragend, um eigene Befehle in BASIC einzubinden. Zu diesem Thema erfahren Sie weiteres in Kapitel 1.5

1.4 Ablage von Variablen und Arrays

Die Variablen des Commodore-BASIC werden in drei Arten unterteilt, die sich in bestimmten Adressbereichen des BASIC-Speichers befinden. Hierbei wird zwischen einfachen, also nicht-indizierten, Variablen, indizierten Variablen und den Inhalten von Stringvariablen unterschieden.

Die nichtindizierten Variablen werden unmittelbar hinter dem BASIC-Programm abgelegt. Der Anfang dieses Blocks ist in einem Zeiger in der Zero-Page in den Adressen $2D/2E (45/46) festgelegt. Da sich der Block direkt hinter dem Programm befindet, ist es klar, daß diese Variablen beim Einfügen eines neuen Programmteils gelöscht werden. Hinter diesem Adressbereich schließt sich der Block mit den indizierten Variablen an, dessen Beginn in den Adressen $2F/30 (47/48) festgelegt ist und der gleichzeitig das Ende des ersten Blocks markiert. Das Ende des zweiten Variablenbereichs wird durch den Zeiger in $31/32 (49/50) gekennzeichnet. Die Inhalte der Stringvariablen werden am Ende des BASIC-Speichers abgelegt und auf der folgenden Seite noch näher erläutert.

Als erstes schreiten wir zur Erklärung der nichtindizierten Variablen, die sich in vier Variablentypen gliedern. Dieses sind die REAL-Variablen, die INTEGER-Variablen, die STRING-Variablen und die Variablen von selbstdefinierten Funktionen mittels DIM. Die Einträge bestehen aus jeweils sieben Bytes, die den Variablennamen und den Variablenwert beziehungsweise den Zeiger auf den Variablenwert beinhalten.

Type	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Real	Name Bit 7 = 0	Name Bit 7 = 0	Exponent	Variablen wert	Variablen wert	Variablen wert	Variablen wert
Integer	Name Bit 7 = 0	Name Bit 7 = 1	Wert (Low)	Wert (High)	-	-	-
String	Name Bit 7 = 1	Name Bit 7 = 0	Länge	Adresse (Low)	Adresse (High)	-	-
FN	Name Bit 7 = 1	Name Bit 7 = 1	Adresse (Low)	Adresse (High)	Wert (Low)	Wert (High)	-

Abb. 1.4.1: Format der Variablentypen

Die Bytes 0 und 1 eines jeden Eintrages beinhalten den Namen und den Typ der Variablen. Der Typ wird durch die beiden siebten Bits gekennzeichnet, die entsprechend gesetzt sind, wie Sie aus der Abbildung 1.4.1 ersehen können. Bei der REAL-Variable enthält das Byte 2 den Exponenten und die restlichen Bytes den Variablenwert. Bei INTEGER-Variablen ist es noch einfacher, da dort die Bytes 2 und 3 das LOW- und HIGH-Byte der entsprechenden Zahl beinhalten. Da Strings in den restlichen fünf Bytes meistens nicht genug Platz finden, werden dort nur die Stringlänge und die jeweilige Startadresse abgelegt. Ähnlich verhält es sich bei den selbstdefinierten Funktionen. Hier werden die Adresse der Funktion und der eigentlichen Variable hintereinander abgelegt.

Die STRINGS werden in einem Bereich am Ende des BASIC-RAMs abgespeichert, auf den der Zeiger in $37/38 (55/56) zeigt. Von dort an werden die Variablen nach unten hin angebaut. Die untere Grenze dieses Bereichs wird in $33/34 (51/52) festgehalten. Zu beachten wäre noch, daß bei Stringoperationen die Zwischenergebnisse abgespeichert und die Endergebnisse

angehängt werden. Bei Platzmangel werden die Zwischenergebnisse gelöscht, was auch durch den Befehl FRE (0) erzwungen werden kann.

Bei dimensionierten (indizierten) Variablen werden für die Einträge nicht mehr grundsätzlich sieben Bytes zur Verfügung gestellt, sondern nur noch die Anzahl der benötigten Bytes. Dies entspricht zum Beispiel zwei Bytes beim Integerformat oder drei Bytes bei Strings. Jedem Feld geht ein Arrayheader voraus, der in Tabelle 1.4.2 erklärt wird.

Zum Schluß noch eine Bemerkung, die Zeit sparen hilft: Es ist auf jeden Fall sinnvoll, die Variablen vor den Arrays zu deklarieren, da beim Einfügen von Variablen alle Arrays verschoben werden müssen.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Name Bit 7 = 1	Name Bit 7 = 1	Feld- länge (Low)	Feld- länge (High)	Anzahl der Dimen- sonen	Spalten- länge (Low)	Spalten- länge (High)

Abb. 1.4.2: Dimensionierte Variablen

1.5 Wie erweitert man BASIC?

Eigene Maschinenroutinen lassen sich außer der Realisierung durch USR- und SYS-Funktion noch eleganter direkt in den BASIC-Interpretercode einbinden. Sehen wir uns dazu die Stelle im Interpreter an, die ein BASIC-Statement holt und ausführt. Hier ist der entsprechende Auszug aus dem ROM-Listing:

```
A7E1 6C 08 03  JMP ($0308); zeigt normalerweise auf $A7E4
A7E4 20 73 00  JSR $0073  ; Zeichen aus BASIC-Text holen
A7E7 20 ED A7  JSR $A7ED  ; Statement ausführen
A7EA 4C AE A7  JMP $A7AE  ; zurück zur Interpreterschleife
```

An dieser Stelle können wir nun eingreifen. In der Adresse $0308/$0309 (776/777) steht der Zeiger für die Ausführung eines BASIC-Befehls, den wir auf eine eigene Routine umstellen können. In dieser Routine können wir dann überprüfen, ob es sich um unseren selbstdefinierten Befehl handelt und entsprechend verzweigen. Eine übliche Methode ist es, eigene Befehlserweiterungen durch ein vorangestelltes Sonderzeichen, zum Beispiel ein Ausrufezeichen, zu kennzeichnen, so könnte

```
100 !PRINT
```

eine eigene modifizierte Druckroutine aufrufen. Unsere Routine prüft dann auf das Ausrufezeichen. Wird es gefunden, kann in die eigene Routine verzweigt werden, ansonsten wird die Routine des BASIC-Interpreters aufgerufen. Ein entsprechender Programmausschnitt könnte so aussehen:

```
ÜBERPRÜFEN JSR  $0073   ; CHRGET, nächstes Zeichen holen
           CMP  #$21    ; mit Sonderzeichen vergleichen
           BEQ  FOUND   ; Verzweigung zur eigenen Routine
           JSR  $0079   ; CHRGOT, Flags wieder setzen
           JMP  $A7E7   ; Interpreterbefehl ausführen
FOUND      JSR  $0073   ; CHRGET, nächstes Zeichen holen
           JSR  COMMAND ; eigenen Befehl ausführen
           JMP  $A7E4   ; zurück zur Interpreterschleife,
                          nächstes Zeichen holen
```

Der Zeiger in $0308/$0309 muß beim Initialisieren der Befehlserweiterung auf die Anfangsadresse (das Label ÜBERPRÜFEN) des obigen Beispielprogramms gesetzt werden. Will man mehrere Befehle implementieren, so kann man noch eine Routine zur Unterscheidung der Befehlsworte einbauen, die die verschiedenen Befehlserweiterungen selektiert und wiederum entsprechend vergleicht.

Im folgenden finden Sie einige Anregungen zur Verwirklichung eigener Routinen. Beim ersten Beispiel handelt es sich um eine Hardcopyroutine. Die Hardcopy-Funktion hat den Zweck, den

Bildschirminhalt auf den Drucker mit der Gerätenummer 4 zu kopieren und kann mit SYS 36864 direkt aufgerufen werden. Es ist jedoch sinnvoller, ein kleines Starterprogramm nach dem obigen Muster zu schreiben, um die Routine als BASIC-Erweiterung nutzen zu können.

Hardcopy Funktion:

```
9000 A9 04    LDA #$04
9002 85 BA    STA $BA    ; Gerätenummer des Druckers
9004 A9 7E    LDA #$7E
9006 85 B8    STA $B8    ; logische Filenummer
9008 A9 00    LDA #$00   ; LOW-Byte des Bildschirms
900A A0 04    LDY #$04   ; HIGH-Byte des Bildschirms
900C 85 71    STA $71    ; als Zeiger merken
900E 84 72    STY $72
9010 85 B7    STA $B7    ; kein Filenamen
9012 85 B9    STA $B9    ; Sekundäradresse gleich 0
9014 20 C0 FF JSR $FFC0  ; Drucker File öffnen
9017 A6 B8    LDX $B8    ; logische Filenummer des Drucker
9019 20 C9 FF JSR $FFC9  ; Drucker als Ausgabegerät
901C A2 19    LDX #$19   ; Anzahl der Bildschirmzeilen (24)
901E A9 0D    LDA #$0D   ; neue Zeile
9020 20 D2 FF JSR $FFD2  ; an Drucker
9023 20 E1 FF JSR $FFE1  ; Stoptaste abfragen
9026 F0 2E    BEQ $9056  ; gedrückt, dann beenden
9028 A0 00    LDY #$00
902A B1 71    LDA ($71),Y; Zeichen vom Bildschirm holen
902C 85 67    STA $67
902E 29 3F    AND #$3F
9030 06 67    ASL $67
9032 24 67    BIT $67    ; Bildschirmkode
9034 10 02    BPL $9038  ; in ASCII-Kode umwandeln
9036 09 80    ORA #$80
9038 70 02    BVS $903C
903A 09 40    ORA #$40
903C 20 D2 FF JSR $FFD2  ; und zum Drucker schicken
903F C8       INY
9040 C0 28    CPY #$28   ; Zeile zu Ende ?
```

```
9042 D0 E6    BNE $902A
9044 98       TYA
9045 18       CLC          ; ja, Zeiger auf nächste
9046 65 71    ADC $71      ; Zeile setzen
9048 85 71    STA $71
904A 90 02    BCC $904E
904C E6 72    INC $72
904E CA       DEX          ; schon alle Zeilen ausgegeben ?
904F D0 CD    BNE $901E
9051 A9 0D    LDA #$0D
9053 20 D2 FF JSR $FFD2    ; neue Zeile
9056 20 CC FF JSR $FFCC    ; Ausgabe wieder auf Bildschirm
9059 A9 7E    LDA #$7E
905B 4C C3 FF JMP $FFC3    ; Druckdatei schließen und fertig
```

Es folgt ein Ladeprogramm in BASIC:

```
100 POKE 56,9*16 : CLR : FOR I = 36864 TO 36957
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169,  4,133,186,169,126,133,184,169,  0,160,  4
130 DATA 133,113,132,114,133,183,133,185, 32,192,255,166
140 DATA 184, 32,201,255,162, 25,169, 13, 32,210,255, 32
150 DATA 225,255,240, 46,160,  0,177,113,133,103, 41, 63
160 DATA   6,103, 36,103, 16,  2,  9,128,112,  2,  9, 64
170 DATA  32,210,255,200,192, 40,208,230,152, 24,101,113
180 DATA 133,113,144,  2,230,114,202,208,205,169, 13, 32
190 DATA 210,255, 32,204,255,169,126, 76,195,255
200 IF S <> 12023 THEN PRINT "FEHLER IN DATAS !!" : END
210 PRINT "OK !"
```

Hier noch ein Programm, daß es Ihnen ermöglicht, die beiden Programme durch Aufrufen mit !H für Hardcopy und !R für Renew als BASIC-Erweiterung zu benutzen:

```
        JSR $0073 ; nächstes Zeichen holen
        CMP #$21  ; mit Sonderzeichen vergleichen
        BEQ FOUND ; gleich: eigene Routine
        JSR $0079 ; CHRGOT, Flags wieder setzen
        JMP $A7E7 ; Interpreterbefehl ausführen
FOUND   CMP #$52  ; Vergleich auf R
        BNE TEST  ; ungleich: dann Hardcopy
        JMP $CF00 ; Renew
        JMP $9000 ; Hardcopy aufrufen
```

Das folgende Beispiel stellt eine RENEW-Funktion dar. Das Programm kann dann nützlich sein, wenn man versehentlich ein Programm mit NEW gelöscht hat. Das Programm findet das Ende des gelöschten Programms und setzt die BASIC-Zeiger wieder auf die alten Werte, sofern danach keine neuen Programmzeilen eingegeben oder Variablen benutzt wurden. Die Startadresse ist hier 12*4096+15*256 gleich 52992.

Die RENEW-Funktion holt ein gelöschtes Programm wieder zurück.

```
CF00 A5 2B    LDA $2B    ; BASIC-Programmstart
CF02 A4 2C    LDY $2C
CF04 85 22    STA $22    ; als Zeiger speichern
CF06 84 23    STY $23
CF08 A0 03    LDY #$03
CF0A C8       INY
CF0B B1 22    LDA ($22),Y; sucht Ende der ersten Zeile
CF0D D0 FB    BNE $CF0A  ; (Nullbyte)
CF0F C8       INY
CF10 98       TYA
CF11 18       CLC
CF12 65 22    ADC $22    ; Offset addieren
CF14 A0 00    LDY #$00
CF16 91 2B    STA ($2B),Y; als Zeiger auf nächste
CF18 A5 23    LDA $23
CF1A 69 00    ADC #$00   ; Zeile speichern
CF1C C8       INY
```

```
CF1D 91 2B    STA ($2B),Y
CF1F 88       DEY         ; enthält jetzt null
CF20 A2 03    LDX #$03
CF22 E6 22    INC $22
CF24 D0 02    BNE $CF28   ; Programmende gleich
CF26 E6 23    INC $23     ; drei Nullbytes suchen
CF28 B1 22    LDA ($22),Y
CF2A D0 F4    BNE $CF20
CF2C CA       DEX
CF2D D0 F3    BNE $CF22
CF2F A5 22    LDA $22
CF31 69 02    ADC #$02
CF33 85 2D    STA $2D
CF35 A5 23    LDA $23    ; Zeiger auf Programmende setzen
CF37 69 00    ADC #$00
CF39 85 2E    STA $2E
CF3B 4C 63 A6 JMP $A663 ; CLR und ready.
```

Hier wieder ein Ladeprogramm in BASIC. Dieses Programm muß natürlich zuerst geladen und gestartet werden, bevor man sein eigenes BASIC-Progamm einlädt oder schreibt. Ansonsten würde man mit dem Nachladen des Beispielprogramms sein eigenes zerstören.

```
100 FOR I = 52992 TO 53053
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 165, 43,164, 44,133, 34,132, 35,160,  3,200,177
130 DATA  34,208,251,200,152, 24,101, 34,160,  0,145, 43
140 DATA 165, 35,105,  0,200,145, 43,136,162,  3,230, 34
150 DATA 208,  2,230, 35,177, 34,208,244,202,208,243,165
160 DATA  34,105,  2,133, 45,165, 35,105,  0,133, 46, 76
170 DATA  99,166
180 IF S <> 7000 THEN PRINT "FEHLER IN DATAS !!" : END
190 PRINT "OK !"
```

1.6 Übergabe von BASIC-Parametern über USR und SYS

Welcher BASIC-Programmierer hat nicht schon einmal daran gedacht, Maschinenroutinen in sein Programm einzubinden, sei es, um ein Programm zu beschleunigen oder um Routinen ausnutzen zu können, die in BASIC nicht verfügbar sind. Eine gute Möglichkeit dazu stellen die Befehle SYS und USR dar.

Mit SYS kann ein Maschinenprogramm an einer beliebigen Stelle des Speichers aufgerufen werden. Natürlich können auch fertige Routinen des Betriebssystems verwendet werden. Startet man zum Beispiel die Routine BILDSCHIRM-RESET ab der Adresse $E518 (58648), werden der Videocontroller und die Bildschirmzeiger initialisiert, die Farbe für den Bildschirm neu gesetzt, die Cursorblinkzeit eingestellt, der Bildschirm gelöscht und der Cursor auf die Position HOME gesetzt. Die Routine muß mit

```
SYS 58648
```

aufgerufen werden. Eine erweiterte Variante des SYS-Befehls werden wir später noch besprechen.

Für Funktionen mit einem Argument bietet sich die USR-Funktion an. Wie funktioniert nun die USR-Funktion? Sie kann genauso wie alle anderen Funktionsaufrufe des Interpreters, zum Beispiel die SIN-Funktion zur Berechnung von Variablen oder auch in einem PRINT-Statement, verwendet werden. Die Besonderheit dabei ist, daß Parameter an das Maschinenprogramm übergeben werden können. Der Befehl

```
X = USR (10)
```

übergibt zum Beispiel die Zahl 10 an den Fließkommaakkumulator, kurz FAC genannt. Damit der Interpreter weiß, an welcher Stelle des Speichers sich die eigene Routine befindet, muß die Startadresse in die Adressen $0311/0312 (785/786) geschrieben werden. An dieser Stelle befindet sich der sogenannte USR-Vektor, über den der Interpreter beim Aufruf des USR-Befehls zur eigenen Routine verzweigt. Normalerweise ist dieser Vektor auf $B248 gerichtet, wo sich der Einsprung für

die Fehlermeldung 'illegal quantity' befindet. Möchte man seine Routine ab der Adresse $C000 starten, dann sieht das folgendermaßen aus:

```
POKE 785,0 : POKE 786,12*16
```

Es ist auch möglich, die Werte aus dem Maschinenprogramm wieder in das BASIC-Programm zu Übergeben. Zur Veranschaulichung des USR-Befehls wollen wir eine Routine zur Berechnung der Quadratwurzel einer Zahl schreiben. Der BASIC-Interpreter stellt eine solche Funktion zwar bereits zur Verfügung, unsere Routine soll jedoch schneller und genauer werden, da wir keine Potenzierung benutzen wollen, die den Aufruf von LOG und EXP erfordert, sondern stattdessen eine Iteration durchführen. Als Startwert nehmen wir dazu das Argument und halbieren den Exponent, was bereits eine gute Schätzung des Wurzelwertes ist. Die Iterationsvorschrift lautet: X(N+1) = (X(N) + A/X(N)) / 2, wobei A das Argument und X(N) und X(N+1) der alte und der neue Schätzwert sind. Durch Ausprobieren zeigt sich, daß sich das Ergebnis nach vier Iterationen nicht mehr ändert.

```
C800 JSR $BC2B ; Vorzeichen testen
C803 BEQ $C839 ; Wert gleich 0, fertig
C805 BPL $C80A ; positiv, dann in Ordnung
C807 JMP $B248 ; negativ, 'ILLEGAL QUANTITY'
C80A JSR $BBC7 ; FAC nach Akku #4 übertragen
C80D LDA $61
C80F SEC
C810 SBC #$81  ; Exponent normalisieren
C812 PHP
C813 LSR       ; Exponent halbieren
C814 CLC
C815 ADC #$01
C817 PLP
C818 BCC $C81C
C81A ADC #$7F  ; Exponent wiederherstellen
C81C STA $61
C81E LDA #$4   ; 4 Iterationen
```

```
C820 STA $67
C822 JSR $BBCA ; FAC nach Akku #3
C825 LDA #$5C
C827 LDY #$00  ; Zeiger auf Akku #4
C829 JSR $BB0F ; durch FAC dividieren
C82C LDA #$57
C82E LDY #$00  ; Zeiger auf Akku #3
C830 JSR $B867 ; zu FAC addieren
C833 DEC $61   ; FAC / 2 (Exponent minus 1)
C835 DEC $67   ; Zähler erniedrigen
C837 BNE $C822 ; noch eine Iteration
C839 RTS       ; Rücksprung ins BASIC
```

Bevor wir unsere neue USR-Funktion aufrufen, müssen wir, wie bereits erwähnt, die Startadresse festlegen. Dazu wird das LOW-Byte der Adresse nach $311 (785) und das HIGH-Byte nach $312 (786) gepoket. Für unsere Funktion sähe das so aus:

```
POKE 785,0 : POKE 786, 12*16+8
```

Das folgende Ladeprogramm in BASIC enthält diese Pokes bereits.

```
100 FOR I = 51200 TO 51257
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32, 43,188,240, 52, 16,  3, 76, 72,178, 32,199
130 DATA 187,165, 97, 56,233,129,  8, 74, 24,105,  1, 40
140 DATA 144,  2,105,127,133, 97,169,  4,133,103, 32,202
150 DATA 187,169, 92,160,  0, 32, 15,187,169, 87,160,  0
160 DATA  32,103,184,198, 97,198,103,208,233, 96
170 IF S <> 6211 THEN PRINT "FEHLER IN DATAS !!" : END
180 POKE 785,0 : POKE 786,200 : PRINT "OK !"
```

Jetzt läßt sich mit ? USR(A) unsere Routine aufrufen. Vergleicht man die Ausführungszeit unserer Routine mit der SQR-Routine des Interpreters, so ist unsere mit ca. 12 Millisekunden gegenüber ca. 52 Millisekunden etwa viermal schneller als die

des Interpreters. Jetzt wollen wir uns noch ein etwas komplizierteres Beispiel ansehen.

Oft steht man vor der Aufgabe, eine Zahlenreihe zu addieren. Dies ist zum Beispiel bei der Ermittlung des Durchschnitts oder anderen statistischen Berechnungen der Fall. Wir nehmen an, daß die Daten in einem Array , also einer dimensionierten Variablen, zur Verfügung stehen.

```
10 DIM A(1000)
. . . . . .     Berechnungen oder Einlesen der Daten
100 S = 0
110 FOR I = 0 TO 1000 : S = S + A(I) : NEXT
120 PRINT S
```

Unsere USR-Funktion soll nun die BASIC-Zeilen 100 und 110 ersetzen. Wir wollen dafür

```
100 S = USR(A)
```

schreiben. Der Parameter A steht dabei für den Arraynamen. Wie wir später sehen werden, läßt sich durch Ändern zweier Maschinenbefehle auch das Produkt der Arrayelemente berechnen.

```
033C JSR $AD8D  ; Variable numerisch?
033F LDX $2F
0341 LDA $30    ; Zeiger auf Beginn der Arraytabelle
0343 STX $5F
0345 STA $60    ; laufender Zeiger
0347 CMP $32
0349 BNE $034F
034B CPX $31    ; Ende der Arraytabelle?
034D BEQ $036C
034F LDY #$00   ; Zeiger setzen
0351 LDA ($5F),Y; erster Buchstabe des Namens
0353 INY        ; Zeiger erhöhen
```

```
0354 CMP $45    ; mit gesuchtem Namen vergleichen
0356 BNE $035E  ; nein, dann nächstes Array testen
0358 LDA $46    ; zweiter Buchstabe
035A CMP ($5F),Y; vergleichen
035C BEQ $0375  ; gefunden
035E INY
035F LDA ($5F),Y
0361 CLC
0362 ADC $5F    ; Offset für nächstes Array addieren
0364 TAX
0365 INY
0366 LDA ($5F),Y
0368 ADC $60
036A BCC $0343
036C LDX #$E2
036E STX $22    ; Zeiger auf Fehlermeldung
0370 LDA #$03
0372 JMP $A445  ; Fehlermeldung ausgeben
0375 INY
0376 LDA ($5F),Y
0378 CLC
0379 ADC $5F
037B STA $24
037D INY
037E LDA ($5F),Y
0380 ADC $60
0382 STA $25
0384 INY
0385 LDA ($5F),Y; Anzahl der Indizes
0387 JSR $B196  ; Zeiger auf erstes Arrayelement
038A STA $5F
038C STY $60    ; Zeiger nach Temp
038E BIT $0E    ; Integerflag testen
0390 BMI $03B1
0392 JSR $BBA2  ; Element in FAC
0395 CLC
0396 BCC $039C  ; Sprung in Schleife
0398 JSR $B867  ; Variable plus FAC
039B CLC
039C LDA $5F
```

```
039E ADC #$5    ; Zeiger auf nächstes Element
03A0 STA $5F
03A2 BCC $03A6
03A4 INC $60
03A6 LDY $60
03A8 CMP $240   ; Ende des Arrays?
03AA BCC $0398
03AC CPY $25
03AE BCC $0398
03B0 RTS        ; ja, fertig
03B1 JSR $03D5  ; Integervariable nach FAC
03B4 JSR $BC0C  ; FAC nach ARG
03B7 CLC
03B8 LDA $5F
03BA ADC #$2    ; Zeiger auf nächstes Arrayelement
03BC STA $5F
03BE BCC $03C2
03C0 INC $60
03C2 CMP $24    ; Ende des Arraybereichs?
03C4 BCC $03CC
03C6 LDA $60
03C8 CMP $25
03CA BCS $03B0
03CC JSR $03D5  ; Integervariable nach FAC holen
03CF JSR $B86F  ; FAC + ARG
03D2 JMP $03B4
03D5 LDY #$00
03D7 LDA ($5F),Y
03D9 TAX
03DA INY
03DB LDA ($5F),Y
03DD TAY
03DE TXA
03DF JMP $B391  ; nach Fließkomma
03E2 41 52 52 TAB       .ASC "ARRAY NOT FOUN"
03E5 41 59 20 4E 4F 54 20 46 50 55 4E
03F0 C4                 .BYTE "D" + $80
```

Wieder das dazugehörige Ladeprogramm in BASIC

```
100 FOR I = 828 TO 1008
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,141,173,166, 47,165, 48,134, 95,133, 96,197
130 DATA  50,208,  4,228, 49,240, 29,160,  0,177, 95,200
140 DATA 197, 69,208,  6,165, 70,209, 95,240, 23,200,177
150 DATA  95, 24,101, 95,170,200,177, 95,101, 96,144,215
160 DATA 162,226,134, 34,169,  3, 76, 69,164,200,177, 95
170 DATA  24,101, 95,133, 36,200,177, 95,101, 96,133, 37
180 DATA 200,177, 95, 32,150,177,133, 95,132, 96, 36, 14
190 DATA  48, 31, 32,162,187, 24,144,  4, 32,103,184, 24
200 DATA 165, 95,105,  5,133, 95,144,  2,230, 96,164, 96
210 DATA 197, 36,144,236,196, 37,144,232, 96, 32,213,  3
220 DATA  32, 12,188, 24,165, 95,105,  2,133, 95,144,  2
230 DATA 230, 96,197, 36,144,  6,165, 96,197, 37,176,228
240 DATA  32,213,  3, 32,111,184, 76,180,  3,160,  0,177
250 DATA  95,170,200,177, 95,168,138, 76,145,179, 65, 82
260 DATA  82, 65, 89, 32, 78, 79, 84, 32, 70, 79, 85, 78
270 DATA 196
280 IF S <> 20399 THEN PRINT "FEHLER IN DATAS !!" : END
290 POKE 785, 3*16+12 : POKE 786, 3 : PRINT "OK !"zln1
```

Das Programm kann sowohl Arrays mit reellen Zahlen als auch Integer-Arrays verarbeiten. Wird ein Array nicht gefunden, so wird die Fehlermeldung 'array not found error' ausgegeben.Da die Logik zum Errechnen des Produkts der Arrayelemente gleich ist, kann man eine Produktfunktion erhalten, indem wir die Aufrufe zur Addition durch die Multiplikationsroutine ersetzen. Dazu muß ab Adresse $0398 20 28 BA (JSR $BA28) stehen und ab Adresse $03CF steht 20 2B BA (JSR $BA2B). Vom BASIC aus kann dies mit POKE 921,40 : POKE 922, 186 : POKE 976, 43 : POKE 977, 186 geschehen.

Um unsere Routine, die diesmal im Bandpuffer liegt, benutzen zu können, müssen wir wieder die Startadresse in $0311/0312 poken, was im BASIC-Programm schon geschehen ist.

```
POKE 785, 3*16+12 : POKE 786, 3
```

Berechnen Sie zum Vergleich einmal die Summe mit einer BASIC-Schleife und dann mit unserer Routine - der Zeitunterschied ist gewaltig.

Sollen mehr als ein Parameter übergeben werden, so ist die USR-Funktion nicht mehr geeignet. Hier bietet sich eine erweiterte Variante des SYS-Befehls an. Normalerweise führt der SYS-Befehl nur das Maschinenprogramm ab der angegebenen Adresse aus und übergibt keine weiteren Parameter. Unsere Routine soll nun einen oder beliebig viele Parameter an das Maschinenprogramm übergeben. Neben der Routine zur Formelauswertung stehen noch eine Reihe weiterer Einsprungpunkte und Unterroutinen zur Verfügung, die zum Beispiel Parameter in Klammern auswerten oder auf ein nachfolgendes Komma prüfen. Auch läßt sich der Variablentyp, String oder numerisch, testen. Bei numerischen Variablen ist zusätzlich noch eine Bereichsüberprüfung möglich. Die wichtigsten Routinen sind unten zusammengestellt. Weitere Einzelheiten entnehmen Sie bitte unserem ROM-Listing.

Adresse	Beschreibung:
AD8A	Argument auswerten und auf numerisch prüfen
AD8D	auf numerisch prüfen
AD8F	auf String prüfen
AD9E	Argumentauswertung, beliebiger Ausdruck
AEF1	Argument in Klammern auswerten
AEF7	prüft auf Klammer zu
AEFA	prüft auf Klammer auf
AEFD	prüft auf Komma
B79E	holt Byte, (0 bis 255) in X-Register
0073	holt nächstes Zeichen aus BASIC-Text

Bei Bereichsüberschreitung wird 'ILLEGAL QUANTITY' ausgegeben, falscher Variablentyp ergibt 'TYPE MISMATCH'. Die Umwandlung der verschiedenen Formate miteinander ist mit folgenden Routinen möglich:

Adresse	Beschreibung
B1BF	wandelt FAC nach Integer
B395	wandelt 16-Bit Integerzahl in A/X nach Fließkomma
B3A2	wandelt Byte in Y nach Fließkomma
BC9B	wandelt FAC nach 16-Bit Zahl
BCF3	wandelt Ziffernstring nach Fließkomma
BDDD	wandelt FAC in Ziffernstring

Jetzt wollen wir uns noch ein Beispiel für einen SYS-Aufruf mit Parameterübergabe ansehen. Will man von BASIC aus eine Bildschirmausgabe an eine bestimmte Position machen, so muß man mit der Cursorsteuerung nach HOME die entsprechende Anzahl an Cursor-right-und Cursor-down-Zeichen drucken. Dies ist umständlich und verbraucht viel Speicherplatz. Einfacher geht es mit einer selbstgeschriebenen Maschinenroutine. Der Aufruf soll folgende Syntax haben:

SYS PR, Spalte, Zeile, Text

Dabei ist PR die Startadresse der Routine, Zeile und Spalte bestimmen die Cursorposition, an die die Variablen oder Ausdrücke des Textes wie beim normalen PRINT-Befehl ausgegeben werden. In unserem Beispiel ist PR = 49152 ($C000).

```
C000 JSR $AEFD  ; prüft auf Komma
C003 JSR $B79E  ; holt Spaltenwert nach X
C006 TXA        ; Spaltenwert in Akku
C007 PHA        ; Spaltennummer merken
C008 JSR $AEFD  ; prüft auf Komma
C00B JSR $B79E  ; holt Zeilenwert
C00E PLA        ; Spaltenwert holen
C00F TAY        ; Spaltenwert nach Y
C010 CLC        ; Flag für Cursor setzen
C011 JSR $FFF0  ; setzt Cursor
C014 JSR $AEFD  ; prüft auf Komma
C017 JMP $AAA4  ; weiter mit PRINT-Befehl
```

Hier wieder das BASIC-Programm:

```
100 FOR I = 49152 TO 49177
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,253,174, 32,158,183,138, 72, 32,253,174, 32
130 DATA 158,183,104,168, 24, 32,240,255, 32,253,174, 76
140 DATA 164,170
150 IF S <> 3566 THEN PRINT "FEHLER IN DATAS !!" : END
160 PRINT "OK !"
```

Wenn man zu Anfang des Programms der Variablen PR die Startadresse $C000 der Routine zuweist, läßt sich mit dem folgenden Befehl der Text "Beispiel" ab der 24. Spalte der 20. Zeile ausgeben.

```
10 PR = 12*4096
100 SYS PR, 24, 20, "Beispiel"
```

1.7 Sprungvektoren und Autostart

Der C64 verfügt im Gegensatz zu vielen anderen Computern über ein ROM (Read Only Memory), in dem das Betriebssystem des Rechners fest abgelegt ist. Das hat den Vorteil, daß das Betriebssystem nach dem Einschalten nicht erst eingeladen werden muß. Andererseits bringt gerade diese Eigenschaft einen großen Nachteil für den Anwender mit sich, der das ROM nicht abändern und seinen Bedürfnissen anpassen kann. Um dem ein wenig abzuhelfen, bietet der C64 eine Reihe von sogenannten Sprungvektoren an, die zwar vom Betriebssystem benutzt werden, aber nicht im ROM, sondern am Anfang des Arbeitsspeichers abgelegt sind. Diese Vektoren liegen ab der Adresse $0300 (768) und verweisen auf die jeweiligen Routinen innerhalb des Betriebssystems. An dieser Stelle kann der Programmierer eingreifen, indem er die Vektoren auf seine eigenen Routinen stellt. Es lassen sich jedoch auch andere Effekte erreichen, die äußerst nützlich sein können.

Hier soll eine schematische Darstellung verdeutlichen, wie die Sprungvektoren vom Betriebssystem benutzt werden.

Abb. 1.7.1: Funktion der Sprungvektoren

Nachdem die Routine direkt oder aus einer anderen Routine heraus angesprungen wurde, wird über einen indirekten Sprung über die Vektoren zur eigentlichen Routine verzweigt. Diese Routine befindet sich im allgemeinen direkt hinter dem indirekten Sprungbefehl. Die ersten Vektoren sind die des BASIC-

Interpreters, die die nachfolgend aufgeführten Funktionen erfüllen. Zum besseren Verständnis ist es ratsam, die Routinen im ROM-Listing nachzuschlagen.

Adresse	Vektor	Beschreibung
$0300/0301	(768/769) $E38B	Vektor für BASIC-Warmstart; wird nach END sowie beim Auftreten eines Fehlers angesprungen (Fehlernummer im Akku)
$0302/0303	$A483 (770/771)	Vektor für Eingabe einer Zeile; Rechner bleibt in der Eingabewarteschleife, bis RETURN erfolgt
$0304/0305	$A57C (772/773)	Vektor für Umwandlung in den Interpretercode
$0306/0307	$A71A (774/775)	LIST-Vektor; wird bei Umwandlung in den Klartext angesprungen.
$0308/0309	$A7E4 (776/777)	Vektor für BASIC-Befehlsadresse holen; zeigt an die Stelle des Interpreters, die den BASIC-Befehl ausführt
$030A/030B	$AE86 (778/779)	Vektor wird angesprungen, wenn ein Element eines Ausdrucks berechnet werden soll
$0311/0312	$B248 (785/786)	USR-Vektor; steht normalerweise auf 'ILLEGAL QUANTITY'

Die Vektoren des Betriebssystems stehen ab $0314/0315

Adresse	Vektor	Beschreibung
$0314/0315	$EA31 (788/789)	IRQ-Vektor; wird jede 1/60 Sekunde angesprungen
$0316/0317	$FE66 (790/791)	BRK-Vektor
$0318/0319	$FE47 (792/793)	NMI-Vektor; wird beim Drücken der RESTORE-Taste benutzt
$031A/031B	$F34A (794/795)	OPEN-Vektor
$031C/031D	$F291 (796/797)	CLOSE-Vektor
$031E/031F	$F20E (798/799)	CHKIN-Vektor
$0320/0321	$F250 (800(801)	CKOUT-Vektor

```
$0322/0323  $F333 (802/803) CLRCH-Vektor
$0324/0325  $F157 (804/805) INPUT-Vektor; normalerweise auf Ein-
                            gabe von der Tastatur
$0326/0327  $F1CA (806/807) OUTPUT-Vektor; normalerweise auf Aus-
                            gabe auf den Bildschirm
$0328/0329  $F6ED (808/809) STOP-Vektor
$032A/032B  $F13E (810/811) GET-Vektor
$032C/032D  $F32F (812/813) CLALL-Vektor
$032E/032F  $FE66 (814/815) Warmstartvektor
$0330/0331  $F4A5 (816/817) LOAD-Vektor
$0332/0333  $F5ED (818/819) SAVE-Vektor
```

Mit diesen Vektoren lassen sich nun einige 'Kunststücke' vollbringen. Man kann zum Beispiel die LIST-Vektoren so umstellen, daß diese auf ein RTS zeigen. Das bewirkt, daß nach einem LIST sofort wieder ins BASIC zurückgesprungen wird, ohne daß überhaupt etwas gelistet wird. Der Befehl wird also einfach ignoriert. Wir haben noch eingige andere Ideen für Sie aufgeschrieben:

POKE 774,	POKE 775,	Adresse	Routine
226	252	$FCE2 (64738)	RESET
68	166	$A644 (42564)	NEW
7	168	$A807 (43015)	SYNTAX ERROR
160	240	ein $02-Wert	Absturz des Systems

Natürlich lassen sich auch die anderen Vektoren auf diese Routinen 'umbiegen'. Wirkungsvoll wäre es zum Beispiel noch beim SAVE-Vektor. Das Umstellen dieses Vektors läßt es zu, mit einem Programm wie üblich zu arbeiten, ohne es jedoch danach wieder abspeichern zu können. Dieser Trick stellt also einen kleinen Kopierschutz dar.

Für den BASIC-Anwender kann es auch von Nutzen sein, den RESTORE-Vektor abzuändern, wodurch eine Unterbrechung des Programms mittels RUN/STOP-RESTORE unmöglich wird.

Auch hier kann wieder zu einer eigenen Routine verzweigt oder die Funktion ganz ausgeschaltet werden. Dies kann zum Beispiel durch POKE792,193: POKE 793,254 erzielt werden.

Nun aber zu einer der interessantesten Anwendungen dieser Vektoren, dem Autostart. Wie Sie bestimmt wissen, kann man mit LOAD"PROGRAMMNAME",8,1 ein Programm in einen bestimmten Bereich des Speichers laden, wenn es zuvor in diesem Bereich abgesichert wurde. Dieses kann man am einfachsten mit einem Maschinensprachemonitor, wie zum Beispiel dem PROFI-MON, machen. Werden nun beim Laden eines Programms die Sprungvektoren überschrieben, werden dadurch alle Zeiger umgestellt und der Rechner stürzt ab. Wurde jedoch vor dem Abspeichern nur ein Zeiger umgeändert und die anderen in ihrem Zustand gelassen, lädt der Computer ordnungsgemäß, und nur eine Routine wird anders ausgeführt. Man kann nun einen Vektor umstellen, der nur manchmal benutzt wird, wie zum Beispiel den LIST-Vektor, oder einen, über den ständig verzweigt wird. Dazu bietet sich die Eingabewarteschleife ($0302/0303) an, die fast ständig auf eine Eingabe von der Tastatur wartet. Diesen Vektor kann man nun auf den eigenen Programmstart stellen, so daß nach dem Laden nicht mehr der Cursor erscheint, sondern direkt ein Programmstart erfolgt. Es ist sinnvoll, das Programm in den Cassettenpuffer ab $033C (828) zu legen, da sonst der Bildschirm mit abgespeichert werden muß.

Auf diese Weise kann aber nur ein Maschinenprogramm gestartet werden, da zu einer bestimmten Adresse gesprungen wird. Soll ein BASIC-Programm gestartet werden, so muß eine kleine Routine zwischengeschaltet werden, die folgendermaßen aussieht:

```
033C JSR $A659 ;CHRGET-Zeiger auf Programmstart und CLR
033F JMP $A7AE ;in die Interpreterschleife springen
```

Eine andere Möglichkeit ist es, über den Vektor $0326/0327 zu verzweigen, da dieser bei jeder Ausgabe auf den Bildschirm angesprungen wird und dementsprechend schwer zu unterdrücken ist.

Eine weitere Alternative bietet der sogenannte Stapelautostart. Der Stapel, auch Stack genannt, befindet sich im Bereich von $0100 (256) bis $01FF (511) und beinhaltet unter anderem die Absprungadressen des Programms beim Aufrufen von Unterprogrammen. Da auch das Laden von einem Unterprogramm aus geschieht, kann der Stapel dabei mit dem Einsprung unseres Programms überschrieben werden. Dabei muß die Startadresse in Low- und High-Byte abgelegt werden, zu der noch 1 hinzuaddiert wird. Beim Rücksprung holt sich der Rechner nun die Adresse aus dem Stapel, die ja mittlerweile abgeändert wurde, und verzweigt dementsprechend.

Das folgende Programm ermöglicht es Ihnen, ein gewöhnliches Programm, das mit RUN gestartet wird, mit einem Autostart zu versehen.

```
5 N=49152
10 READX:IFX=-1THEN30
20 S=S+X:POKE N,X:N=N+1:GOTO 10
30 IFS<>22926 OR N<>49339 THEN PRINT"FEHLER IN  DATA S":END
40 SYS49152
101 DATA 169,0,141,32,208,141,33,208,169,5,141,134,2,162,0,189,148
,192,201
102 DATA 32,240,7,32,210,255,232,76,15,192,32,130,192,162,8,160,1,
32,186,255
103 DATA 162,0,160,207,134,187,132,188,169,0,133,157,32,213,255,16
5,144,201
104 DATA 64,208,196,162,0,189,149,192,240,7,32,210,255,232,76,62,1
92,32,130
105 DATA 192,162,0,169,32,157,0,4,232,208,250,162,64,160,3,142,38,
3,140,39
106 DATA 3,162,0,189,170,192,157,64,3,232,224,16,208,245,162,0,160
,3,134,251
107 DATA 132,252,169,251,166,174,164,175,32,216,255,76,64,3,162,0,
134,183
108 DATA 32,207,255,157,0,207,232,230,183,201,13,208,243,96,147,13
,80,82,79
```

```
109 DATA 71,82,65,77,77,78,65,77,69,58,32,40,78,69,85,41,0,162,202
,160,241
110 DATA 142,38,3,140,39,3,32,89,166,76,174,167,32,-1
```

1.8 Die Adressen der BASIC-Routinen

Im folgenden sind die Einsprungadressen der BASIC-Routinen aufgeführt. Sie eignen sich zur Verwendung in eigenen BASIC- und Assemblerprogrammen, wie in Kapitel 6.1 beschrieben wird. Zum genaueren Verständnis schauen Sie sich die Routinen bitte auch im ROM-Listing an. Nun noch ein Wort zu den Routinen beim VC 20.

Der BASIC-Interpreter des Commodore 64 ist mit dem des VC 20 identisch. Er ist lediglich in der Adresslage verschoben. Die Umrechnung einer Adresse des Commodore 64 in die entsprechende Adresse des VC 20 geschieht folgendermaßen: Bei Adressen von $A000 bis $BFFF wird einfach $2000 dazuaddiert, aus $A860 wird die Adresse $C860 im VC 20. Bei Adressen von $E000 bis $E37A wird von der Commodore-64-Adresse der Wert 3 abgezogen. Aus $E30E wird die VC 20 Adresse $E30B.

Adresse	Beschreibung
A000	Startvektor
A002	NMI-Vektor
A004	'cbmbasic'
A00C	Adressen der BASIC-Befehle minus 1
A052	Adressen der BASIC-Funktionen
A080	Hierarchiecodes und Adressen der BASIC-Operatoren
A09E	Liste der BASIC-Befehlsworte
A19E	BASIC-Fehlermeldungen
A328	Adressen der Fehlermeldungen
A364	Meldungen des BASIC-Interpreters
A38A	Stapelsuchroutine für FOR-NEXT und GOSUB
A3B8	Blockverschieberoutine
A3FB	prüft auf Platz im Stapel
A408	schafft Platz im Speicher
A435	Ausgabe von 'out of memory'

A437	Fehlermeldung ausgeben
A469	Break-Einsprung
A474	Ready-Einsprung
A480	Eingabe-Warteschleife
A49C	Löschen und Einfügen von Programmzeilen
A533	BASIC-Programmzeilen neu binden
A560	holt eine Zeile in den Eingabepuffer
A571	Ausgabe von 'string too long'
A579	Umwandlung einer Zeile in Interpretercode
A613	Startadresse einer BASIC-Zeile suchen
A642	BASIC-Befehl NEW
A65E	BASIC-Befehl CLR
A68E	Programmzeiger auf BASIC-Start setzen
A69C	BASIC-Befehl LIST
A717	Interpretercode in Befehlswort umwandeln
A742	BASIC-Befehl FOR
A7AE	Interpreterschleife, führt BASIC-Befehle aus
A7ED	führt einen BASIC-Befehl aus
A81D	BASIC-Befehl RESTORE
A82C	bricht Programm bei gedrückter Stop-Taste ab
A82F	BASIC-Befehl STOP
A831	BASIC-Befehl END
A857	BASIC-Befehl CONT
A871	BASIC-Befehl RUN
A883	BASIC-Befehl GOSUB
A8A0	BASIC-Befehl GOTO
A8D2	Basic-Befehl RETURN
A8F8	BASIC-Befehl DATA
A906	sucht nächstes Statement
A909	sucht nächste Zeile
A928	BASIC-Befehl IF
A93B	BASIC-Befehl REM
A94B	BASIC-Befehl ON
A96B	sucht Adresse einer BASIC-Zeile
A9A5	BASIC-Befehl LET
AA80	BASIC-Befehl PRINT#
AA86	BASIC-Befehl CMD
AAA0	BASIC-Befehl PRINT
AB1E	String ausgeben
AB3E	Leerzeichen bzw. Cursor right ausgeben

AB4D	Fehlerbehandlung bei Eingabe
AB7B	BASIC-Befehl GET
ABA5	BASIC-Befehl INPUT#
ABBF	BASIC-Befehl INPUT
AC06	BASIC-Befehl READ
ACFC	'?extra ignored' und '?redo from start'
AD1D	BASIC-Befehl NEXT
AD8A	FRMNUM: holt Ausdruck und prüft auf numerisch
AD8D	prüft auf numerisch
AD8F	prüft auf String
AD99	Ausgabe von 'type mismatch'
AD9E	FRMEVL holt und wertet beliebigen Ausdruck aus
AE83	arithmetischen Ausdruck holen
AEA8	Fließkommakonstante Pi
AED4	BASIC-Befehl NOT
AEF1	holt Ausdruck in Klammern
AEF7	prüft auf 'Klammer zu'
AEFA	prüft auf 'Klammer auf'
AEFD	prüft auf 'Komma'
AEFF	prüft auf Zeichen im Akku
AF08	Ausgabe von 'syntax error'
AF28	holt Variable
AFE6	BASIC-Befehl OR
AFE9	BASIC-Befehl AND
B016	Vergleichsoperationen
B081	BASIC-Befehl DIM
B113	prüft auf Buchstabe
B194	berechnet Zeiger auf erstes Arrayelement
B1A5	Fließkommakonstante -32768
B1AA	FAC nach Integer wandlen
B245	Ausgabe von 'bad subscript'
B248	Ausgabe von 'illegal quantity'
B34C	berechnet Arraygröße
B37D	BASIC-Funktion FRE
B39E	BASIC-Funktion POS
B3A6	Test auf Direkt-Modus
B3AB	Ausgabe von 'illegal direct'
B3AE	Ausgabe von 'undef'd function'
B3B3	BASIC-Befehl DEF
B3E1	FN-Syntax prüfen

B3F4 BASIC-Funktion FN
B465 BASIC-Funktion STR$
B475 Stringverwaltung, Zeiger auf String berechnen
B487 String einrichten
B526 Garbage Collection, nichtgebrauchte Strings löschen
B63D Stringverknüpfung '+'
B6A3 Stringverwaltung FRESTR
B6EC BASIC-Funktion CHR$
B700 BASIC-Funktion LEFT$
B72C BASIC-Funktion RIGHT$
B737 BASIC-Funktion MID$
B77C BASIC-Funktion LEN
B782 Stringparameter holen
B78B BASIC-Funktion ASC
B79B Holt Byte-Ausdruck (0 bis 255)
B7AD BASIC-Funktion VAL
B7EB Holt Adresse(0 bis 65535) und Byte-Wert(0 bis 255)
B7F7 FAC nach Adreßformat wandeln (Bereich 0 bis 65535)
B80D BASIC-Funktion PEEK
B824 BASIC-Befehl POKE
B82D BASIC-Befehl WAIT
B849 FAC = FAC + 0.5
B850 Minus FAC = Konstante (A/Y) - FAC
B853 Minus FAC = ARG - FAC
B867 Plus FAC = Konstante (A/Y) - FAC
B86A Plus FAC = ARG + FAC
B97E Ausgabe von 'overflow'
B9BC Fließkommakonstanten für LOG
B9EA BASIC-Funktion LOG
BA28 Multiplikation FAC = Konstante (A/Y) * FAC
BA2B Multiplikation FAC = ARG * FAC
BA8C ARG = Konstante (A/Y)
BAE2 FAC = FAC * 10
BAF9 Fließkommakonstante 10
BAFE FAC = FAC / 10
BB0F FAC = Konstante (A/Y) / FAC
BB12 FAC = ARG / FAC
BB8A Ausgabe von 'division by zero'
BBA2 FAC = Konstante (A/Y)
BBC7 Akku#4 = FAC

BBCA Akku#3 = FAC
BBD0 Variable = FAC
BBFC FAC = ARG
BC0C ARG = FAC
BC1B FAC runden
BC2B Vorzeichen von FAC holen
BC39 BASIC-Funktion SGN
BC58 BASIC-Funktion ABS
BC5B Konstante (A/Y) mit FAC vergleichen
BC9B Umwandlung FAC nach Integer
BCCC BASIC-Funktion INT
BCF3 Umwandlung ASCII nach Fließkomma
BDB3 Fließkommakonstanten für Fließkomma nach ASCII
BDC2 Ausgabe der Zeilennummer bei Fehlermeldung
BDCD Positive Integerzahl (0 bis 65535) ausgeben
BDDD FAC nach ASCII-Format wandeln
BF11 Fließkommakonstante 0.5
BF16 Binärzahlen für Umwandlung FAC nach ASCII
BF71 BASIC-Funktion SQR
BF78 Potenzierung FAC = Konstante (A/Y) hoch FAC
BF7B Potenzierung FAC = ARG hoch FAC
BFBF Fließkommakonstanten für EXP
BFED BASIC-Funktion EXP

E043 Polynomberechnung
E059 Polynomberechnung
E08D Fließkommakonstanten für RND
E097 BASIC-Funktion RND
E107 Ausgabe von 'break'
E10C BSOUT: ein Zeichen ausgeben
E112 BASIN: ein Zeichen empfangen
E118 CKOUT: Ausgabegerät festsetzen
E11E CHKIN: Eingabegerät festsetzen
E124 GETIN: ein Zeichen holen
E12A BASIC-Befehl SYS
E156 BASIC-Befehl SAVE
E165 BASIC-Befehl VERIFY
E168 BASIC-Befehl LOAD
E1BE BASIC-Befehl OPEN
E1C7 BASIC-Befehl CLOSE

E1D4	Parameter für LOAD und SAVE holen
E219	Parameter für OPEN holen
E264	BASIC-Funktion COS
E26B	BASIC-Funktion SIN
E2B4	BASIC-Funktion TAN
E2E0	Fließkommakonstanten für SIN und COS
E30E	BASIC-Funktion ATN
E33E	Fließkommakonstanten für ATN
E37B	BASIC-NMI-Einsprung
E394	BASIC-Kaltstart
E3A2	Kopie der CHRGET-Routine
E3BA	Anfangswert für RND-Funktion
E3BF	RAM für BASIC initialisieren
E447	Tabelle der BASIC-Vektoren
E453	BASIC-Vektoren laden

1.9 Fließkommaarithmetik

Maschinensprache bietet gegenüber BASIC den Vorteil der hohen Geschwindigkeit. Dafür hat sie jedoch den unübersehbaren Nachteil, daß sie schwer zu handhaben ist. Dies zeigt sich besonders bei komplizierteren Berechnungen. Addition und Subtraktion lassen sich noch auf recht einfache Weise durchführen, selbst Multiplikation und Division sind mit ein wenig Mehraufwand realisierbar. Kompliziert wird es aber, wenn man solche Rechnungen nicht nur mit ganzen Zahlen (Integer-Zahlen), sondern auch mit gebrochenen Zahlen, Zahlen mit Nachkommastellen also, durchführen will. Wie läßt sich zum Beispiel in Maschinensprache der Umfang eines Kreises mit dem Radius R berechnen? In BASIC stellt das kein Problem dar. Der entsprechende Befehl lautet einfach: U=2*Pi*r . 2*r stellt auch in Maschinensprache kein Problem dar, dagegen die Multiplikation mit Pi ein ziemlich großes.

Die Lösung dieses Problems trägt den Namen "Fließkommaarithmetik". Fließkommaarithmetik ermöglicht beispielsweise dem BASIC-Interpreter, solche Berechnungen wie die oben genannte mit der uns bekannten Genauigkeit auszuführen. Wenn wir solche Formeln in Maschinensprache ausrechnen wollen,

brauchen wir uns demnach keine eigenen Programmteile dafür zu schreiben, sondern wir können auf die Unterroutinen des Interpreters zurückgreifen. Es ist dem Verständnis dieser Routinen dienlich, wenn man den Aufbau von Fließkommazahlen kennt. Wer sich hier schon auskennt oder wer kein Interesse an ein wenig Theorie hat, kann den folgenden Absatz überspringen.

Format von Fließkommazahlen

Der Aufbau von Fließkommazahlen läßt sich am einfachsten darstellen, wenn man sich überlegt, auf welche verschiedenen Arten sich ein und dieselbe gebrochene Zahl in BASIC darstellen läßt. Die Zahl 1.234567 läßt sich auch noch als 0.1234567E1, 0.01234567E2 oder 12.34567E-1 schreiben. Der Teil vor dem E heißt "Mantisse", der Teil dahinter "Exponent". Das E trägt die Bedeutung von "*10^".

Die Art der Darstellung ist, wie man sieht, nicht eindeutig festgelegt. Da der Computer allerdings Fließkommazahlen auch als Mantisse und Exponent getrennt abspeichert, muß man eine einheitliche Darstellung finden. Man einigt sich darauf, den Exponenten so zu wählen, daß die Mantisse kleiner als Eins wird, jedoch keine führenden Nullen hinter dem Komma, beziehungsweise dem Dezimalpunkt in BASIC, besitzt. Das Komma wird also durch Änderung des Exponenten so verschoben, daß die am weitesten links stehende Ziffer der Mantisse direkt rechts neben ihm steht. Daher auch die Bezeichnung "Fließkomma": Das Komma "fließt" entlang der Zahl.

Der Computer arbeitet natürlich nicht mit einem Exponenten, der eine Potenz von 10 darstellt, sondern, da ja intern jegliche Arithmetik binär abläuft, mit einem Exponenten zur Basis 2. Beispielsweise wird die Zahl 123.625 folgendermaßen dargestellt:

$$123.625=1*2^6+1*2^5+1*2^4+1*2^3+0*2^2+1*2^1+1*2^0+1*2^{-1}+0*2^{-2}+1*2^{-3}$$

Die binäre Darstellung von 123.625 ist also 1111011.101, die binäre Exponentialdarstellung demnach 0.1111011101E111. Der Exponent 111, dezimal 7, ergibt sich dadurch, daß er genau der

Anzahl Stellen entspricht, um die das Komma in der Mantisse nach links gerückt ist.

Der BASIC-Interpreter des C-64 benutzt für eine Fließkommazahl fünf Bytes. Das erste Byte beinhaltet den Exponenten, die restlichen vier die Mantisse. Hier zeigt sich auch der große Vorteil der Fließkommadarstellung: Man kann, ohne die Anzahl der benutzten Bytes zu verändern, einerseits relativ große Zahlen mit geringer Genauigkeit, andererseits kleinere Zahlen mit entsprechend größerer Genauigkeit darstellen.

Die 32 Bits der 4 Byte der Mantisse entsprechen den 32 Stellen dieser Zahl. Da man auch negative Fließkommazahlen benutzen möchte, muß ein Bit als Vorzeichenbit dienen. Man verwendet dabei das erste Bit der Mantisse, welches normalerweise immer den Wert 1 hat, da die Zahl ja keine führenden Nullen mehr enthalten darf. Wenn dieses Bit den Wert 0 hat, handelt es sich um eine positive, beim Wert 1 um eine negative Fließkommazahl. Bei Berechnungen im ROM wird das Bit zuerst ausgelesen, zwischengespeichert und durch eine Eins ersetzt. Nach Abschluß einer Rechenoperation wird es wieder an derselben Stelle eingefügt.

Die Mantissen zweier vom Betrag her gleicher, jedoch vom Vorzeichen her unterschiedlicher Zahlen unterscheiden sich nur im Vorzeichenbit, im Gegensatz zu negativen Integer-Zahlen, die meistens in der Zweier-Komplementform verwendet werden. Der Exponent dagegen wird bis auf eine Abweichung in Zweier-Komplementform dargestellt. Die Abweichung besteht darin, daß aus Gründen vereinfachter Rechenoperationen Bit 7 invertiert ist, beziehungsweise daß zur "normalen" Darstellung des Exponenten 128 addiert wird.

Wenn Sie bis hierhin alles verstanden haben, dann wird Ihnen vielleicht aufgefallen sein, daß es mit dem bisher beschrieben Aufbau einer Fließkommazahl nicht möglich ist, die Zahl Null darzustellen. Da die erste Ziffer der Mantisse zwingend den Wert Eins hat, läßt sich auch durch einen noch so kleinen Exponenten nicht die Null erreichen. Daher vereinbart man, daß eine Zahl mit dem Exponenten Null selbst den Wert Null hat. Der

Wert der Mantisse spielt dabei keine Rolle, obwohl man ihn üblicherweise auch auf Null setzt.

Unterroutinen des BASIC-Interpreters

Wie schon gesagt, brauchen wir uns nicht unbedingt mit dem etwas komplizierten Aufbau von Fließkommazahlen zu befassen, wenn wir mit ihnen rechnen wollen, da uns der BASIC-Interpreter alle dafür notwendigen Routinen zur Verfügung stellt. Alle diese Routinen benutzen den sogenannten "Fließkomma-Akku", abgekürzt FAC. Es handelt sich dabei um die zu einem Fließkommaregister zusammengefaßten 5 Bytes von $61 bis $65. Nahezu sämtliche Fließkommaoperationen beziehen sich auf dieses Register. Dabei wird meistens noch ein Hilfsregister verwendet, nämlich die Bytes $69 bis $6D, die auch als Fließkomma-Akku 2 oder ARG bezeichnet werden. Es existieren daneben auch noch Akkus 3 und 4 ab $87 und $92, diese werden allerdings seltener gebraucht.

Wie benutzt man nun erwähnte Routinen? Man muß zuerst unterscheiden, ob man eine Rechenoperation durchführt, die zwei Operatoren erfordert, oder eine Funktion wie zum Beispiel INT oder SIN benutzt, die keinen weiteren Operator benötigen. Letztere werden durch ihren Aufruf einfach auf den Inhalt des FACs angewendet, das Ergebnis wird dort auch wieder gespeichert. Die Einsprünge aller BASIC-Funktionen sind in Tabelle 1 aufgelistet.

Operationen wie "+", "-", "*" oder "/", sowie das Laden des FACs mit einer Zahl finden zwischen einer Fließkommavariablen oder -konstanten und dem FAC statt. Dazu wird die Adresse dieser Zahl im Akku und Y-Register des Prozessors an die entsprechende Routine übergeben, und zwar das Low-Byte in A und das High-Byte in Y. Das Übertragen des FAC-Inhalts in eine Variable erfolgt fast genauso, jedoch wird hier das Low-Byte nicht in A, sondern im X-Register angegeben. Bei den Operationen "-" und "/" muß außerdem noch die Reihenfolge beachtet werden. Es wird nämlich nicht, wie man erwarten könnte, die

Variable vom FAC, sondern der FAC von der Variable abgezogen. Entsprechendes gilt für die Division. Die Liste der Interpreterroutinen befindet sich in Tabelle 2.

Weiterhin existieren Routinen, um Integerzahlen in Fließkommazahlen, Strings in Fließkommazahlen und natürlich beides umgekehrt umzuwandeln. Bei der Wandlung Integer - Fließkomma befindet sich die 16-Bit-Integerzahl in A und Y. Hier enthält jetzt allerdings Y das Lowbyte und A das Highbyte. Wenn der FAC zu groß oder zu klein ist, so daß er sich nicht umwandeln läßt, kommt es zu einer Fehlermeldung des Interpreters.

Bei der Umwandlung des FACs in einen String werden die ASCII-Zeichen ab $0100 abgelegt. Das Ende des Strings wird durch ein Null-Byte kenntlich gemacht. Die Routine liefert in A/Y die Adresse $0100 zurück, so daß zur Ausgabe des Strings direkt im Anschluß an die Umwandlung die Stringausgabe-Routine $AB1E aufgerufen werden kann. Will man einen String in eine Fließkommazahl umwandeln, so muß man die Adresse des Strings in $22/$23 übergeben und seine Länge in A. Tabelle 3 enthält die Adressen aller Umwandlungsroutinen.

Um die Speicherstellen, die man sich als Variable einrichtet, mit einem Wert vorzubesetzen, hat man drei Möglichkeiten:

a) Man gibt die Zahlen als String vor, wandelt sie um und speichert sie in seinen Variablen.

b) Wenn es sich um Integer-Zahlen handelt, kann man die Umwandlung Integer nach Fließkomma benutzen und entsprechend wie bei a) verfahren.

c) Wenn möglich, kann man auch auf die im ROM schon vorhandenen Konstanten zurückgreifen. Die wichtigsten sind in Tabelle 4 aufgeführt.

Eine vollständige Liste aller Interpreterroutinen finden Sie in Kapitel 6. Dort finden Sie auch eine Bezugnahme auf die Akkus 2, 3 und 4, die aber für die Anwendung der Routinen meistens

außer Acht gelassen werden können. Die Akkus 3 und 4 können allerdings von Ihnen als Hilfsregister benutzt werden, wenn Sie beispielsweise die Rechenreihenfolge bei Subtraktion oder Division umkehren wollen.

Schnittstelle zu BASIC

Durch die USR-Funktion ist eine einfache Möglichkeit gegeben, zwischen BASIC- und Maschinenprogrammen Fließkommawerte in beiden Richtungen auszutauschen. Bei ihrem Gebrauch wird ein Maschinenprogramm gestartet, das ab der Adresse beginnt, auf die der Vektor $0311/$0312 zeigt. Vorher wird allerdings das Argument der USR-Funktion ausgewertet und das Ergebnis in den FAC übertragen. Wenn das Maschinenprogramm mit RTS beendet ist, wird der jetzige Inhalt des FAC dem BASIC-Programm als Wert der Funktion übergeben.

Im Anschluß an die Tabellen folgen zwei Beispielprogramme, die die Verwendung der Fließkomma-Routinen demonstrieren.

Tabelle 1 - BASIC-Funktionen

Adresse	Beschreibung
$AED4	FAC =NOT(FAC)
$B9EA	FAC =LOG(FAC)
$BC39	FAC =SGN(FAC)
$BC58	FAC =ABS(FAC)
$BCCC	FAC =INT(FAC)
$BF71	FAC =SQR(FAC)
$BFED	FAC =EXP(FAC)
$E097	FAC =RND(FAC)
$E264	FAC =COS(FAC)
$E26B	FAC =SIN(FAC)
$E2B4	FAC =TAN(FAC)
$E30E	FAC =ATN(FAC)

Tabelle 2 - Operationen mit zwei Zahlen

Adresse	Beschreibung
$BBA2	FAC =(A/Y)
$BBD4	(X/Y) =FAC
$B850	FAC =(A/Y)-FAC
$B867	FAC =(A/Y)+FAC
$BA28	FAC =(A/Y)*FAC
$BB0F	FAC =(A/Y)/FAC

Tabelle 3 - Umwandlung verschiedener Datenformate

Adresse	Beschreibung
$B1AA	Y/A =INT(FAC) (mit Vorzeichen)
$B391	FAC =Integerzahl Y/A (mit Vorzeichen)
$B7B5	String ab ($22/$23) nach FAC wandeln
$B7F7	Y/A =INT(FAC) (ohne Vorzeichen)
$BDDD	FAC nach String ab Adresse $0100 wandeln

Tabelle 4 - Fließkommakonstanten im ROM

Adresse	Beschreibung
$AEA8	Pi
$B1A5	-32768
$B9BC	1
$B9D6	SQR(2)/2
$B9DB	SQR(2)
$BAF9	10
$BF11	0.5
$E2E0	Pi/2
$E2E5	2*Pi
$E2EA	0.25

Beispielprogramm 1:

Eingabe eines Radius R von der Tastatur und Berechnung des zugehörigen Kreisumfangs.

```
C000  A2 00      LDX #$00     ;Zähler für Zeichenanzahl
C002  20 CF FF   JSR $FFCF    ;Zeichen von Tastatur
C005  C9 0D      CMP #$0D     ;=RETURN?
C007  F0 07      BEQ $C010    ;wenn ja, weiter
C009  9D 00 C1   STA $C100,X  ;wenn nein, abspeichern
C00C  E8         INX          ;Zähler erhöhen
C00D  4C 02 C0   JMP $C002    ;und Schleife fortsetzen
C010  20 D2 FF   JSR $FFD2    ;RETURN ausgeben
C013  8A         TXA          ;Stringlänge nach A
C014  A2 00      LDX #$00     ;Stringadresse nach $22/23
C016  A0 C1      LDY #$C1
C018  86 22      STX $22
C01A  84 23      STY $23
C01C  20 B5 B7   JSR $B7B5    ;String nach FAC wandeln
C01F  A9 E5      LDA #$E5     ;$E2E5 ist Adresse von
C021  A0 E2      LDY #$E2     ;2*Pi
C023  20 28 BA   JSR $BA28    ;mal FAC
C026  20 DD BD   JSR $BDDD    ;FAC nach String wandeln
C029  20 1E AB   JSR $AB1E    ;String ausgeben
C02C  A9 0D      LDA #$0D     ;2 mal RETURN ausgeben
C02E  20 D2 FF   JSR $FFD2
C031  20 D2 FF   JSR $FFD2
C034  4C 00 C0   JMP $C000    ;zurück zum Anfang
```

Beispielprogramm 2:

Berechnung der Formel:

USR(X)=INT(SQR((X*3.2+4/7)/0.35))+0.5)

Der USR-Vektor muß zuvor auf $C000 eingestellt werden. Dies geschieht durch POKE 785,0:POKE 786,192. Danach läßt sich die Funktion direkt von BASIC aus aufrufen.

Das Programm benutzt zwei Hilfsregister für Fließkommazahlen in $C100-$C104 und $C105-$C109. Die Zahlen 3.2 und 0.35 sind als Strings ab C06D abgelegt.

```
C000  A2 00     LDX #$00
C002  A0 C1     LDY #$C1
C004  20 D4 BB  JSR $BBD4   ;x aus FAC nach $C100
C007  A9 03     LDA #$03    ;Länge des Strings 3.2
C009  A2 6D     LDY #$6D    ;Adresse des Strings 3.2
C00B  A0 C0     LDY #$C0
C00D  86 22     STX $22     ;nach $22/23
C00F  84 23     STY $23
C011  20 B5 B7  JSR $B7B5   ;String nach FAC wandeln
C014  A9 00     LDA #$00
C016  A0 C1     LDY #$C1
C018  20 28 BA  JSR $BA28   ;FAC=3.2 * x
C01B  A2 00     LDX #$00
C01D  A0 C1     LDY #$C1
C01F  20 D4 BB  JSR $BBD4   ;FAC nach $C100
C022  A0 04     LDY #$04    ;Integerzahl 4
C024  A9 00     LDA #$00    ;in Y/A
C026  20 91 B3  JSR $B391   ;nach FAC wandeln
C029  A2 05     LDX #$05
C02B  A0 C1     LDY #$C1
C02D  20 D4 BB  JSR $BBD4   ;FAC nach $C105
C030  A0 07     LDY #$07    ;Integerzahl 7
C032  A9 00     LDA #$00    ;in Y/A
C034  20 91 B3  JSR $B391   ;nach FAC wandeln
C037  A9 05     LDA #$05;
C039  A0 C1     LDY #$C1
C03B  20 0F BB  JSR $BB0F   ;FAC=4 / 7
C03E  A9 00     LDA #$00
C040  A0 C1     LDY #$C1
C042  20 67 B8  JSR $B867   ;FAC=FAC + 3.2 * x
C045  A2 0A     LDX #$00
C047  A0 C1     LDY #$C1
C049  20 D4 BB  JSR $BBD4   ;FAC nach $C100
C04C  A9 04     LDA #$04    ;Länge des Strings 0.35
```

```
C04E  A2 70      LDX #$70    ;Adresse des Strings 0.35
C050  A0 C0      LDY #$C0
C052  86 22      STX $22     ;nach $22/23
C054  84 23      STY $23
C056  20 B5 B7   JSR $B7B5   ;String nach FAC wandeln
C059  A9 00      LDA #$00
C05B  A0 C1      LDY #$C1
C05D  20 0F BB   JSR $BB0F   ;FAC=(x*3.2 + 4/7) / FAC
C060  20 71 BF   JSR $BF71   ;FAC=SQR(FAC)
C063  A09 11     LDA #$11    ;ROM-Konstante 0.5
C065  A3.20 BF   LDY #$BF    ;aus $BF11
C067  20 67 B8   JSR $B867   ;zu FAC addieren
C06A  4C CC BC   JMP $BCCC   ;FAC=INT(FAC)
C06D 33 2E 32 30 2E 33 35    ;3.2 0.35
```

Nach Abschluß der Routine befindet sich daß Ergebnis im FAC und wird daher beim USR-Aufruf als Funktions-Ergebnis an das BASIC-Programm übergeben.

1.10 Der Virus-Killer

Unter Computerviren versteht man Programme, die irgendwo im letzten Winkel des Speichers verborgen liegen und Schaden anrichten. Sie bringen zum Beispiel den Rechner zum 'Absturz', oder sie beschreiben die im Laufwerk befindliche Diskette.

Um diese 'Plage' aus dem Rechner zu verscheuchen, muß der gesamte Speicher des C64 gelöscht werden, wobei auch das unter dem Kernal- und BASIC-ROM befindliche RAM gelöscht werden muß, um jede Möglichkeit in Betracht zu ziehen, den Virus zu zerstören.

Aber um an das unter dem Betriebssystem befindliche RAM heranzukommen, muß das ROM ausgeblendet werden. Das kann man durch das Umstellen des Prozessorports in der Zeropage erreichen, indem man die Bits 0 und 1 auf LOW stellt, also löscht.

Anschließend wird der gesamte Speicher mit Nullen überschrieben und damit gesäubert. Dazu ist dieses kleine Maschinenprogramm notwendig:

```
033C SEI          Interrupt verhindern
033D LDA #$00     ROM-Berreich
033F STA $01      ausblenden
0341 LDX #$00     Zähler 1 setzen
0343 LDY #$F6     Zähler 2 setzen
0345 LDA #$00     Füllwert
0347 STA $0400,X  Speicher löschen
034A INX          Zähler 1 erhöhen
034B BNE $0347    Wenn 255 erhöht, dann
034D INC $0349    High-Byte erhöhen
0350 DEY          Zähler 2 erniedrigen
0351 BNE $0347    Wenn Zähler 2=0, dann
0353 LDA #$37     ROMs wieder
0355 STA $01      einblenden
0357 JMP $FCE2    RESET
```

Es folgt das gleiche Programm als BASIC-Loader:

```
100 FOR I=1 TO 30 STEP 15:FOR J=0 TO 14:READ A$: B$=RIGHT$(A$,1)
105 A=ASC(A$)-48:IF A>9 THEN A=A-7
110 B=ASC(B$)-48:IF B>9 THEN B=B-7
120   A=A*16-B:C=(C+A)AND255:POKE827+I+J,A:NEXT:READ   A:IF   C=A   THEN
C=0:NEXT:END
130 PRINT "FEHLER IN ZEILE:";PEEK(63)+PEEK(64)*256:STOP
300 DATA 78,A9,00,85,01,A2,00,A0,F6,A9,00,9D,00,04,E8, 17
301 DATA D0,FA,EE,49,03,88,D0,F4,A9,37,85,01,4C,E3,FC, 244
```

Das kleine Maschinenprogramm liegt im Kassettenpuffer und wird mit SYS 828 gestartet. Das hat den Vorteil, daß der Kassettenpuffer nach dem RESET automatisch gelöscht wird und der Speicher danach absolut 'sauber' ist.

1.11 Der BASIC-Kompaktor

Der BASIC-Packer ist ein nützliches Utility, um Speicherplatz auf der Diskette zu sparen, da man mit diesem Programm sich im Speicher befindliche BASIC-Programme komprimieren kann, so daß sie weniger Speicherplatz benötigen.

Unter Komprimieren ist in dem Fall nicht das Zusammenpressen gemeint, sondern BASIC-Zeilen, in denen wenige Befehle vorkommen, werden mit der vorherigen verbunden. Das heißt, zwei Befehle, die vorher in zwei Zeilen gestanden haben, werden einfach in einer Zeile zusammengefaßt. Zum Beispiel so:

Vorher:
```
10 PRINT
20 PRINT
30 PRINT
```

Nacher:
```
10 PRINT:PRINT:PRINT
```

Dieses Programm ist aus Geschwindigkeitsgründen vollkommen in Maschinensprache geschrieben, und es faßt in nur wenigen Sekunden bis zu 245 Zeichen pro Zeile zusammen.

Programmzeilen, in denen GOTO-, GOSUB- oder THEN-Befehle vorkommen, werden nicht gebunden, weil das zu Fehlern im Programmablauf führen könnte. Auch Zeilen, in denen REM-Befehle vorkommen, bleiben unverändert, weil die nachfolgenden Befehle als REM-Text anerkannt würden.

Außerdem müssen alle PRINT- und OPEN-Befehle mit Anführungszeichen abgeschlossen sein, da sonst vom Kompaktor ein "SYNTAX ERROR" ausgegeben wird.

Wenn nun ein BASIC-Programm gepackt werden soll, muß entweder der folgende BASIC-Lader gestartet werden, oder man lädt das Maschinenprogramm direkt in den Speicher und startet es anschießend mit SYS 49152. Vorher muß jedoch NEW eingegeben werden, um die Zeiger wieder richtig zu stellen, da das Maschinenprogramm ab der Adresse $C000 (49152) in den Speicher geladen wird.

Wenn man den BASIC-Lader benuzt, muß dieser erst gestartet werden. Anschließend muß NEW eingegeben werden, um dem Programm Platz zu schaffen, das gepackt werden soll. Das zu packende Programm kann jetzt von Diskette oder Datasette in den Speicher geladen werden. Nach dem Ladevorgang wird der Packer ebenfalls mit SYS 49152 gestartet. Nachdem das Programm komprimiert wurde, kann es ganz normal mit SAVE wieder gespeichert werden.

Das kompimierte Programm ist auf jeden Fall einige Bytes kürzer als vorher. So kann man viel Speicherplatz auf Diskette sparen.

Vorsicht: Man sollte nicht versuchen, ein komprimiertes Programm anschließend zu verändern, weil viele Zeilen Überlänge haben und somit bei bei Abänderung einer Zeile evtl. Teile einer Zeile abgeschnitten werden können.

Hier nun das Maschinenlisting des Packer-Programms und der dazugehörige BASIC-Loader:

```
C000 LDA §$36     BASIC-Interpreter
C002 STA $01      ausblenden
C004 JSR $C0A7    Tabelle angesprungen.
                  Zeilen erstellen
C007 LDA $2B      BASIC-Anfang LOW-Byte
C009 SEC          Carry für Subtraktion
C00A SBC §$01     minus eins
C00C STA $AB      Zeilspeixher LOW-Byte
C00E STA $A9      Programm LOW-Byte
C010 LDA $2C      BASIC-Anfang HIGH
C012 SBC §$00     minus Übertrag
C014 STA $AC      Zielspeicher HIGH
C016 STA $AA      Programm HIGH
C018 LDY §$01     Linkzeiger setzen
C01A LDA ($A9),Y  Linkbyte LO holen
C01C INY          Programmzeiger erhöhen
C01D ORA ($A9),Y  Byte HIGH verknüpfen
```

```
C01F BEQ $C05E   verzweige, wenn Null
C021 INY         Programmzeiger erhöhen
C022 LDA ($A9),Y Zeilennummer LO laden
C024 TAX         nach X schieben
C025 INY         Programmzeiger erhöhen
C026 LDA ($A9),Y Zeilennummer High
C028 JSR $C213   Zeile in Tabelle ?
C02B BEQ $C079   verzweige, wenn ja
C02D LDY §$01    Zeiger auf Linker
C02F LDA ($A9),Y Linker LO-Byte holen
C031 SEC         Carry für Subtraktion
C032 SBC $A9     minus Programmzeiger
C034 SEC         Carry für Subtraktion
C035 SBC §$05    gekürzte Zeilenlänge
C037 CLC         Carry für Addition
C038 ADC $AD     plus Zeilenlänge
C03A BCS $C079   größer als 255 ?
C03C CMP §$F5    gleich 245 ?
C03E BCS $C079   verzweige, wenn ja
C040 STA $AD     neue Zeilenlänge
C042 LDY §$00    Verschiebeschleife=0
C044 LDA §$3A    ASCII ":"
C046 STA ($AB),Y in Zielspeicher
C048 INY         Zeiger erhöhen
C049 LDA $A9     Programmzeiger LO
C04B CLC         Carry für Addition
C04C ADC §$04    plus 4
C04E STA $A9     nach Programmzeiger
C050 BCC $C054   verzweige, wenn kein
                 übertrag
C052 INC $AA     Programmzeiger erhöhen
C054 LDA ($A9),Y Programmbyte holen
C056 BEQ $C090   nächste Zeile ?
C058 STA ($AB),Y in Zielspeicher
C05A INY         Zähler erhöhen
C05B JMP $C054   Sprung zum Anfang
C05E TAY         Zielbereich=0
C05F STA ($AB),Y Programmende=0
C061 INY         Zeiger erhöhen
C062 CPY §$03    schon 3 Nullen ?
```

```
C064 BNE §C05F   verzweige, wenn ja
C066 TYA         Y-Register nach AKKU
C067 CLC         Carry für Addition
C068 ADC §AB     Programmende berechnen
C06A STA §2D     in Programmzeiger
C06C LDA §AC     Programmende HIGH
C06E ADC §$00    Übertrag addieren
C070 STA $2E     in Programmzeiger
C072 LDA §$37    BASIC-Interpreter
C074 STA $01     einschalten
C076 JMP $E1AB   CLR, Rücksprung
C079 LDY §$00    Zähler auf Null
C07B LDA ($A9),Y fünf Programmbytes
C07D STA ($AB),Y verschieben
C07F INY         Zähler erhöhen
C080 CPY §$05    fünf Bytes verschoben?
C082 BNE $C07B   verzweige, wenn nein
C084 LDA ($A9),Y Programmbyte holen
C086 BEQ $C08E   nächste Zeile erreicht
C088 STA ($AB),Y Programmbyte speichern
C08A INY         Zähler erhöhen
C08B JMP $C084   zum Schleifenanfang
C08E STY $AD     Zeilenlänge speichern
C090 TYA         Zähler nach Akku
C091 CLC         Carry für Addition
C092 ADC $A9     Programmzeiger LOW
C094 STA $A9     berechnen
C096 BCC $C09A   kein Übertrag ?
C098 INC $AA     Programmzeiger HIGH
C09A TYA         Zähler nach Akku
C09B CLC         Carry für Addition
C09C ADC $AB     Zielzeiger berechnen
C09E STA $AB     und speichern
C0A0 BCC $C0A4   kein Übertrag ?
C0A2 INC $AC     Zielzeiger HIGH
C0A4 JMP $C018   nächste Zeile
C0A7 LDA $2B     BASIC-Anfang LO
C0A9 SEC         Carry für Subtraktion
C0AA SBC §$01    minus 1
C0AC STA $AB     Programmzeiger LO
```

```
C0AE LDA $2C      BASIC-Anfang HIGH
C0B0 SBC §$00     minus 1
C0B2 STA $AC      Programm Zeiger HIGH
C0B4 LDY §$00     LOW-Byte von $A000
C0B6 LDA §$A0     HIGH-Byte von $A000
C0B8 STY $A5      Tabellenzeiger LO
C0BA STA $A6      Tabellenzeiger HIGH
C0BC LDY §$03     erste Zeilennummer
C0BE LDA ($AB),Y  Zeilennummer LO
C0C0 TAX          nach X-Reg.
C0C1 INY          Programmzeiger erhöhen
C0C2 LDA ($AB),Y  Zeilennummer HIGH
C0C4 JSR $C193    in Tabelle
C0C7 INY          Programmzeiger erhöhen
C0C8 LDA ($AB),Y  Programmbyte holen
C0CA BNE $C0F6    noch nicht Zeilenende?
C0CC TYA          Zeiger nach Akku
C0CD CLC          Carry für Addition
C0CE ADC $AB      Programmzeiger
C0D0 STA $AB      berechnen
C0D2 BCC $C0D6    kein übertrag ?
C0D4 INC $AC      Programmzeiger HIGH
C0D6 LDY §$01     Zeiger auf Linker
C0D8 LDA ($AB),Y  Linker LO
C0DA INY          Programmzeiger erhöhen
C0DB ORA ($AB),Y  Akku mit Linker HIGH
C0DD BNE $C0E6    noch nicht Ende ?
C0DF STA $A7      Tabellenzeiger LO
C0E1 LDA §$A0     HIGH-Byte von $A000
C0E3 STA $A8      Tabellenzeiger HIGH
C0E5 RTS          Rücksprung
C0E6 LDA $02      IF-Flag testen
C0E8 BEQ $C0F1    noch nicht gesetzt ?
C0EA LDA §$00     IF-Flag
C0EC STA $02      löschen
C0EE JMP $C0BC    Zeile eintragen
C0F1 LDY §$05     Zeile überspringen
C0F3 JMP $C0C8    nächstes Programmbyte
C0F6 CMP §$8D     GOSUB-Token ?
C0F8 BEQ $C121    verzweige, wenn ja
```

```
COFA CMP §$89     GOTO-Token ?
COFC BEQ $C121    verzweige, wenn ja
COFE CMP §$CB     GO-Token ?
C100 BEQ $C11A    verzweige, wenn ja
C102 CMP §$8B     IF-Token ?
C104 BEQ $C17A    verzweige, wenn ja
C106 CMP §$A7     THEN-Token ?
C108 BEQ $C121    verzweige, wenn ja
C10A CMP §$22     Anführungszeichen ?
C10C BEQ $C182    verzweige, wenn ja
C10E CMP §$8F     REM-Token ?
C110 BEQ $C17A    verzweige, wenn ja
C112 CMP §$91     ON-Token ?
C114 BEQ $C17A    verzweige, wenn ja
C116 INY          Programmzeiger erhöhen
C117 JMP $C0C8    nächstes Byte testen
C11A INY          Programmzeiger erhöhen
C11B LDA ($AB),Y  Programmbyte holen
C11D CMP §$20     Leerzeichen ?
C11F BEQ $C11A    verzweige, wenn ja
C121 INY          GOTO überspringen
C122 LDA ($AB),Y  Programmbyte holen
C124 CMP §$20     Leerzeichen ?
C126 BEQ $C121    verzweige, wenn ja
C128 LDA §$37     BASIC-Interpreter
C12A STA $01      einschalten
C12C STY $AE      Y-REG. speichern
C12E TYA          Zeiger nach Akku
C12F CLC          Carry für Addition
C130 ADC $AB      Programmzeiger LO
C132 STA $22      berechnen
C134 LDA $AC      Programmzeiger HIGH
C136 ADC §$00     Übertrag addieren
C138 STA $23      und speichern
C13A LDA ($AB),Y  Programmbyte holen
C13C CMP §$30     kleiner als ASCII"0"?
C13E BCC $C148    verzweige, wenn ja
C140 CMP §$3A     größer als ASCCI"9" ?
C142 BCS $C148    verzweige, wenn ja
C144 INY          Zeiger erhöhen
```

```
C145 JMP $C13A    Schleifenanfang
C148 TYA          Y-REG. auf
C149 PHA          Stapel retten
C14A SEC          Carry für Subtraktion
C14B SBC $AE      Ziffernfolge berechnen
C14D BEQ $C171    gleich Null ?
C14F JSR $B7B5    Ziffernstring in
                  Fließkommazahl wandeln
C152 JSR $B7F7    Fließkommazahl in
                  Interger wandeln
C155 PLA          Y-Reg. vom Stapel
C156 TAY          zurückholen
C157 LDA §$36     BASIC-Interpreter
C159 STA $01      wieder abschalten
C15B LDX $14      Zeilennummer LO
C15D LDA $15      Zeilennummer HIGH
C15F JSR $C193    in Tabelle eintragen
C162 LDA ($AB),Y  nächstes Programmbyte
C164 INY          Programmzeiger erhöhen
C165 CMP §$20     Leerzeichen?
C167 BEQ $C162    verzweige, wenn ja
C169 DEY          Programmzeiger -1
C16A CMP §$2C     ASCII "," ?
C16C BEQ $C121    verzweige, wenn ja
C16E JMP $C0C8    nachstes Programmbyte
C171 PLA          Y-REG. vom Stapel
C172 TAY          zurückholen
C173 LDA §$36     BASIC-Interpreter
C175 STA $01      abschalten
C177 JMP $C0C8    nächstes Programmbyte
C17A LDA §$01     IF-Flag, REM oder ON
C17C STA $02      setzen
C17E INY          Token überspringen
C17F JMP $C0C8    nächstes Programmbyte
C182 INY          ASCII " überspringen
C183 LDA ($AB),Y  Programmbyte holen
C185 BEQ $C18F    Zeile zu Ende ?
C187 CMP §$22     zweites " erreicht ?
C189 BNE $C182    verzweige, wenn nein
C18B INY          Programmzeiger erhöhen
```

```
C18C JMP $C0C8   nächstes Programmbyte
C18F DEY         Programmzeiger -1
C190 JMP $C17A   IF-Flag setzen
C193 STX $A9     Zeilennummer LOW
C195 STA $AA     Zeilennummer HIGH
C197 STY $AE     Y-REG. speichern
C199 LDY §$00    LOW-Byte von $A000
C19B STY $A7     nach Tabellenzeiger LO
C19D LDA §$A0    HIGH-Byte von $A000
C19F STA $A8     nach Tabellenzeiger
C1A1 CPY $A5     Tab.-Zeiger vergl.
C1A3 LDA $A8     Tabellenzeiger HIGH
C1A5 SBC $A6     Tabellenende HIGH
C1A7 BEQ $C1CD   verzweige, wenn Ende
C1A9 SEC         Carry für Subtraktion
C1AA LDA ($A7),Y Zeilennummer LOW
C1AC SBC $A9     mit neuer vergleichen
C1AE TAX         Differenz speichern
C1AF INY         Tabellenzeiger erhöhen
C1B0 LDA ($A7),Y Zeilennummer HIGH
C1B2 SBC $AA     mit neuer vergleichen
C1B4 INY         Programmzeiger LOW
C1B5 BNE $C1B9   kein Übertrag ?
C1B7 INC $A8     Programmzeiger HIGH
C1B9 BCC $C1A1   verzweige, wenn größer
C1BB CMP §$00    HIGH-Byte gleich ?
C1BD BNE $C1C5   verzweige, wenn nein
C1BF TXA         LOW-Byte gleich ?
C1C0 BNE $C1C5   verzweige, wenn nein
C1C2 LDY $AE     Y-REG. zurückholen
C1C4 RTS         Rücksprung
C1C5 DEY         Zeiger LOW verringern
C1C6 CPY §$FF    Übertrag ?
C1C8 BNE $C1CC   verzweige, wenn nein
C1CA DEC $A8     Programmzeiger HIGH
C1CC DEY         Programmzeiger LOW
C1CD STY $A7     Y-REG. in Programmz.
C1CF LDA $A5     Tabellenende LOW
C1D1 STA $FB     Verschieberreg.LOW
C1D3 LDA $A6     Tabellenende HIGH
```

```
C1D5 STA $FC      Verschiebereg. HIGH
C1D7 LDA $A7      Tabellenzeiger LOW
C1D9 CMP $FB      Verschiebereg. LOW
C1DB LDA $A8      Tabellenzeiger HIGH
C1DD SBC $FC      Verschiebereg. HIGH
C1DF BCS $C1FD    verzweige, wenn größer
C1E1 DEC $FB      Verschiebereg. LOW
C1E3 LDA $FB      und holen
C1E5 CMP §$FF     Übertrag ?
C1E7 BNE $C1EB    verzweige, wenn nein
C1E9 DEC $FC      Verschiebereg. HIGH
C1EB LDY §$00     Verschiebereg. LOW
C1ED LDA ($FB),Y Zeilennummer LOW
C1EF LDY §$02     Verschiebereg. LOW
C1F1 STA ($FB),Y Zeilennummer LOW
C1F3 DEY          Verschiebereg. HIGH
C1F4 LDA ($FB),Y Zeilennummer HIGH
C1F6 LDY §$03     Schleifenanfang
C1F8 STA ($FB),Y Zeiger LOW
C1FA JMP $C1D7    Zeilennummer LOW
C1FD LDY §$00     in Tabelle schreiben
C1FF LDA $A9      Zeiger HIGH
C201 STA ($FB),Y Zeilennummer HIGH
C203 INY          Zeiger auf HIGH
C204 LDA $AA      Zeilennummer HIGH
C206 STA ($FB),Y in Tabelle schreiben
C208 INC $A5      Tabellenende LOW
C20A INC $A5      Tabellenende LOW
C20C BNE $C1C2    kein übertrag
C20E INC $A6      Tabellenende HIGH
C210 JMP $C1C2    Rücksprung
C213 STY $AE      Y-REG. speichern
C215 STX $FB      Zeilennummer LOW
C217 STA $FC      Zeilennummer HIGH
C219 LDA $A7      Tabellenzeiger LOW
C21B CMP $A5      Tabellenende LOW
C21D LDA $A8      Tabellenzeiger HIGH
C21F SBC $A6      Tabellenende HIGH
C221 BCS $C23D    Ende erreicht ?
C223 LDY §$00     Zeiger auf Null
```

```
C225 LDA $FB     Zeilennummer LOW
C227 CMP ($A7),Y Tabellenwert vergl.
C229 INY         Zeiger erhöhen
C22A LDA $FC     Zeilennummer HIGH
C22C SBC ($A7),Y Tabellenwert
C22E BCC $C23D   nicht in Tabelle ?
C230 INC $A7     Tabellenzeiger LOW
C232 INC $A7     Tabellenzeiger LOW
C234 BNE $C238   kein Übertrag ?
C236 INC $A8     Tabellenzeiger HIGH
C238 LDY $AE     Y-REG. zurückholen
C23A LDA §$00    Zero-Flag setzen
C23C RTS         Rücksprung
C23D LDY $AE     Y-REG. zurückholen
C23F LDA §$01    Zero-Flag löschen
C241 RTS         Rücksprung
```

```
100 FORI=1TO578STEP15:FORJ=0TO14:READA$:B$=RIGHT$(A$,1)
105 A=ASC(A$)-48:IFA>9THENA=A-7
110 B=ASC(B$)-48:IFB>9thEnB=B-7
120 A=A*16+B:C=(C+A)AND255:POKE49151+I+J,A:NEXT:READA:IFC=ATHENC=
0:NEXT:END
130 PRINT"FEHLER IN ZEILE:";PEEK(63)+PEEK(64)*256:STOP
300 DATA A9,36,85,01,20,A7,C0,A5,2B,38,E9,01,85,AB,85,  147
301 DATA A9,A5,2C,E9,00,85,AC,85,AA,A0,01,B1,A9,C8,11,  151
302 DATA A9,F0,3D,C8,B1,A9,AA,C8,B1,A9,20,13,C2,F0,4C,  245
303 DATA A0,01,B1,A9,38,E5,A9,38,E9,05,18,65,AD,B0,3D,  254
304 DATA C9,F5,B0,39,85,AD,A0,00,A9,3A,91,AB,C8,A5,A9,  174
305 DATA 18,69,04,85,A9,90,02,E6,AA,B1,A9,F0,38,91,AB,  147
306 DATA C8,4C,54,C0,A8,91,AB,C8,C0,03,D0,F9,98,18,65,  117
307 DATA AB,85,2D,A5,AC,69,00,85,2E,A9,37,85,01,4C,AB,  39
308 DATA E1,A0,00,B1,A9,91,AB,C8,C0,05,D0,F7,B1,A9,F0,  181
309 DATA 06,91,AB,C8,4C,84,C0,84,AD,98,18,65,A9,85,A9,  183
310 DATA 90,02,E6,AA,98,18,65,AB,85,AB,90,02,E6,AC,4C,  130
311 DATA 18,C0,A5,2B,38,E9,01,85,AB,A5,2C,E9,00,85,AC,  229
312 DATA A0,00,A9,A0,84,A5,85,A6,A0,03,B1,AB,AA,C8,B1,  95
313 DATA AB,20,93,C1,C8,B1,AB,D0,2A,98,18,65,AB,85,AB,  45
314 DATA 90,02,E6,AC,A0,01,B1,AB,C8,11,AB,D0,07,85,A7,  168
315 DATA A9,A0,85,A8,60,A5,02,F0,07,A9,00,85,02,4C,BC,  172
```

```
316 DATA C0,A0,05,4C,C8,C0,C9,8D,F0,27,C9,89,F0,23,C9, 212
317 DATA CB,F0,18,C9,8B,F0,74,C9,A7,F0,17,C9,22,F0,74, 81
318 DATA C9,8F,F0,68,C9,91,F0,64,C8,4C,C8,C0,C8,B1,AB, 30
319 DATA C9,20,F0,F9,C8,B1,AB,C9,20,F0,F9,A9,37,85,01, 46
320 DATA 84,AE,98,18,65,AB,85,22,A5,AC,69,00,85,23,B1, 172
321 DATA AB,C9,30,90,08,C9,3A,B0,04,C8,4C,3A,C1,98,48, 226
322 DATA 38,E5,AE,F0,22,20,B5,B7,20,F7,B7,68,A8,A9,36, 38
323 DATA 85,01,A6,14,A5,15,20,93,C1,B1,AB,C8,C9,20,F0, 107
324 DATA F9,88,C9,2C,F0,B3,4C,C8,C0,68,A8,A9,36,85,01, 98
325 DATA 4C,C8,C0,A9,01,85,02,C8,4C,C8,C0,C8,B1,AB,F0, 181
326 DATA 08,C9,22,D0,F7,C8,4C,C8,C0,88,4C,7A,C1,86,A9, 148
327 DATA 85,AA,84,AE,A0,00,84,A7,A9,A0,85,A8,C4,A5,A5, 176
328 DATA A8,E5,A6,F0,24,38,B1,A7,E5,A9,AA,C8,B1,A7,E5, 20
329 DATA AA,C8,D0,02,E6,A8,90,E6,C9,00,D0,06,8A,D0,03, 68
330 DATA A4,AE,60,88,C0,FF,D0,02,C6,A8,88,84,A7,A5,A5, 54
331 DATA 85,FB,A5,A6,85,FC,A5,A7,C5,FB,A5,A8,E5,FC,B0, 54
332 DATA 1C,C6,FB,A5,FB,C9,FF,D0,02,C6,FC,A0,00,B1,FB, 37
333 DATA A0,02,91,FB,88,B1,FB,A0,03,91,FB,4C,D7,C1,A0, 21
334 DATA 00,A5,A9,91,FB,C8,A5,AA,91,FB,E6,A5,E6,A5,D0, 99
335 DATA B4,E6,A6,4C,C2,C1,84,AE,86,FB,85,FC,A5,A7,C5, 84
336 DATA A5,A5,A8,E5,A6,B0,1A,A0,00,A5,FB,D1,A7,C8,A5, 108
337 DATA FC,F1,A7,90,0D,E6,A7,E6,A7,D0,02,E6,A8,A4,AE, 253
338 DATA A9,00,60,A4,AE,A9,01,60,00,00,00,00,00,00,00, 101
```

2. Der Aufstieg zu Assembler

Dieses Kapitel wendet sich an Einsteiger in die Maschinensprache-Programmierung des Commodore-Computers C64.

Maschinensprache wird oftmals als "schwierig", "abstrakt" und so weiter bezeichnet. Dieses Kapitel soll beweisen, daß Maschinensprache ebenso problemlos wie jede andere Programmiersprache zu erlernen ist. "Theoretische Abhandlungen" werden Sie relativ selten entdecken. Dieses Kapitel ist praxisorientiert, das heißt, die verschiedenenen Befehle der Maschinensprache werden anhand ausführlich erläuterter Demoprogramme dargestellt. Sie werden niemals mit einem Befehl, der nur theoretisch behandelt wurde, "allein gelassen". Immer wird Ihnen zugleich auch die praktische Anwendung des Befehls mit mehreren Beispielprogrammen vorgeführt.

Der Zweck dieses Kapitels besteht darin, Leser, die bereits einigermaßen mit BASIC vertraut sind, auf möglichst einfache Weise in die "Maschinenprogrammierung" einzuführen.

Möglichst einfach bedeutet, daß Ihnen zu Beginn nicht der Reihe nach alle Befehle dieser Sprache erläutert werden, sondern zuerst jene, die problemlos anzuwenden sind und mit denen bereits effektive Programme erstellt werden können. Anschließend werden die wichtigsten Routinen des Betriebssystems vorgestellt, die Ihnen "fix und fertig programmiert" viele Hilfsprogramme zur Verfügung stellen, die Sie in eigenen Programmen verwenden können.

Nach diesen Kapiteln werden Sie bereits in der Lage sein, sehr interessante Programme zu erstellen. Diesem "einführenden" Teil folgt der zweite Hauptteil, in dem nach einer kurzen Wiederholung des im ersten Teil erlernten Stoffes systematisch alle Befehlsklassen erläutert werden, die zur Erstellung komplexer Maschinenprogramme benötigt werden.

Dieser zweite Hauptteil ist wie folgt aufgebaut:

- Befehlserläuterung anhand von Demoprogrammen: Die Wirkungsweise und Funktion des jeweiligen Befehls wird erklärt und der Befehl selbst zur Vertiefung in mehreren kleinen Programmen angewendet.
- Vertiefung durch Übungsprogramme: jedem in sich abgeschlossenen Kapitel folgt ein Abschnitt, in dem ausschließlich Programme erstellt werden. Diese zum Teil recht anspruchsvollen Programme sollen Ihnen anschaulich zeigen, wie die erläuterten Befehle sinnvoll einzusetzen sind und Ihnen die benötigten Programmiertechniken vermitteln.

Im Gegensatz zu höheren Programmiersprachen wie BASIC ist die Kenntniß der vorhandenen Befehle allein sinnlos. Um in Maschinensprache programmieren zu können, müssen Sie wissen, wie Ihr Rechner intern aufgebaut ist, welche wichtigen Bereiche er enthält, wie die Routinen des Betriebssystems genutzt werden und welche Speicherbereiche für die Ablage von Daten und Programmen vorhanden sind.

Im weiteren Verlauf werden Sie feststellen, daß es bereits mit relativ geringen und schnell erlernten Kenntnissen über das "Innenleben" Ihres Rechners möglich ist, sehr interessante Programme zu entwickeln.

Der vierte Hauptteil erläutert ein ganz hervorragendes Hilfsmittel, das uns der Rechner "kostenlos" zur Verfügung stellt, den eingebauten BASIC-Interpreter.

Die Erläuterung der wichtigsten Routinen des BASIC-Interpreters erschließt Ihnen zwei "Welten": BASIC und Maschinensprache. Maschinenprogramme stehen nicht mehr im "freien Raum", sondern können zusammen mit BASIC-Programmen eingesetzt werden, um zum Beispiel Eingabe- oder Sortierroutinen zu erstellen.

Dieser Teil soll Ihnen alles vermitteln, was Sie benötigen, um in Maschinensprache Routinen zu schreiben, die ein BASIC-Programm nutzen und anwenden kann.

Noch ein Hinweis: Am Ende dieses Buches finden Sie das komplette Listing eines "Monitor-Programms", eines sehr nützlichen Werkzeugs bei der Programmierung.

Ohne Ihr Wissen haben Sie ziemlich sicher bereits das eine oder andere Maschinensprache-Programm eingegeben. In fast allen Heimcomputer-Zeitschriften finden sich Unmengen von Listings, die zum großen Teil aus sogenannten "Data-Zeilen" bestehen. Diese Data-Zeilen enthalten fast immer Maschinensprache-Programme im Folgenden kurz "Maschinenprogramme" genannt).

Diese Programme demonstrieren häufig den großen Vorteil von Maschinensprache, die enorme Geschwindigkeit. Die erstellten Programme sind meist mehrere hundertmal so schnell wie BASIC-Programme und somit immer noch extrem viel schneller als selbst compilierte BASIC-Programme.

Ein weiterer Vorteil: ein Maschinenprogramm ist immer erheblich kürzer als das entsprechende BASIC-Programm. Mit keiner anderen Programmiersprache kann ein derart kompakter und effizienter Programmcode erstellt werden.

Einige Probleme - zum Beispiel sogenannte "Interrupt-Routinen" - lassen sich sogar nur in Maschinensprache lösen. Sie sehen also, es gibt genug Gründe, um sich für diese Programmiersprache zu interessieren. Das es nur in Maschinensprache möglich ist, auf Heimcomputern "wirklich professionelle" Programme zu schreiben, beweist die Werbung für viele Programme, in denen diese als "zu 100% in Maschinensprache geschrieben" gerühmt werden.

Ein kleiner Nachteil ist, daß Maschinensprache-Befehle nicht ganz so komfortabel sind wie BASIC-Befehle. Zum Beispiel müssen zur Division oder Multiplikation von zwei Zahlen eigene Unterprogramme erstellt werden, entsprechende Befehle sind nicht vorhanden. Dieser Nachteil ist jedoch nicht sehr gravierend, da Sie wohl kaum eine Buchhaltung in Maschinensprache schreiben, sondern eher Videospiele oder eine kleine Textverar-

beitung. Gerade zeitkritischen Anwendungen dieser Art sind optimale Einsatzgebiete für Maschinensprache.

Zudem werden bereits wir im Verlaufe dieses Buches die sogenannten "Betriebssystem-" und "BASIC-Interpreter"-Routinen kennenlernen, fest eingebaute Unterprogramme, die das Erstellen eigener Programme erheblich erleichtern.

Diese Routinen stellen eine Art "Unterprogramm-Bibliothek" dar und enthalten fertige Programmteile für die verschiedensten Anwendungen und Problemstellungen. Die Nutzung dieser Unterprogramme versetzt Sie sehr schnell in die Lage, effektive Maschinenprogramme zu schreiben.

Sicherlich kennen Sie Hilfsmittel zur Programmierung in BASIC wie zum Beispiel Befehlserweiterungen. Zur Programmierung in Maschinensprache benötigen Sie einen sogenannten "Monitor", ein Programm, das die Befehlseingabe ungemein erleichtert.

Monitor-Programme gibt es in Hülle und Fülle, zum Beispiel als Listings in den verschiedensten Zeitschriften. Die Arbeitsweise und Bedienung dieser Programme ist standardisiert, die benötigten Eingaben unterscheiden sich nur in seltenen Fällen.

Ein Monitor ist Voraussetzung für die Eingabe der in diesem Buch verwendeten Programmbeispiele. Im Anhang dieses Buches finden Sie ein Monitor-Programm, das Sie verwenden können.

Einführung in Assembler

Der Grundkurs vermittelt Ihnen bereits alles Notwendige zur Erstellung kleiner Assembler-Programme, die Grundlagen zum Umgang mit einem Monitor-Programm, die wichtigsten Befehle und Adressierungsarten, und erläutert die wichtigsten Unterprogramme, die uns das Betriebssystem zur Verfügung stellt.

Die ersten "Gehversuche"

Für den Fall, daß Sie noch nie - auch nicht in Form von Data-Zeilen - Kontakt mit einem Maschinenprogramm hatten, sollten Sie das weiter unten abgebildete Demoprogramm eingeben.

Das eigentliche Maschinenprogramm ist in der Data-Zeile am Ende des BASIC-Programms enthalten. Seine recht einfache Aufgabe besteht darin, die oberen Bildschirmzeilen mit dem Buchstaben A zu füllen.

```
100 REM *** 'A' AUSGEBEN ***
110 FOR I=49152 TO 49152+10
120 : READ A:POKE I,A
130 NEXT
140 PRINT CHR$(147):REM SCREEN LOESCHEN
150 SYS 49152:REM PROGRAMM STARTEN
160 END
170 :
180 DATA 169,1,162,0,157,0,4,202,208,250,96
```

BASIC-Programmierer (und an die wendet sich ja dieses Buch) sind sicherlich über die enorme Geschwindigkeit erstaunt, mit der die 256 Zeichen ausgegeben werden. Irgendeine Verzögerung ist nicht festzustellen, das Programm arbeitet für das menschliche Auge praktisch in "Nullzeit".

Zum Vergleich ein BASIC-Programm, das analog zum Maschinenprogramm arbeitet:

```
100 REM *** 'A' AUSGEBEN IN BASIC ***
110 PRINT CHR$(147);:REM SCREEN LOESCHEN
120 FOR I=1 TO 256
130 : PRINT"A";
140 NEXT
```

Zugegeben: auch das BASIC-Programm ist nicht allzu langsam (circa eine Sekunde zur Ausgabe aller A's). Der Unterschied zwischen diesem und dem Maschinenprogramm ist dennoch erstaunlich.

Um Ihnen den Geschwindigkeitsunterschied zu verdeutlichen: eine vor Jahren von mir in BASIC geschriebene Sortierroutine (Bubble-Sort) benötigte zur Sortierung von 1000 Strings etwa drei Stunden. Die gleiche Routine in Maschinensprache sortiert 1000 Strings in zehn Sekunden!

2.1 Maschinensprache und Assembler

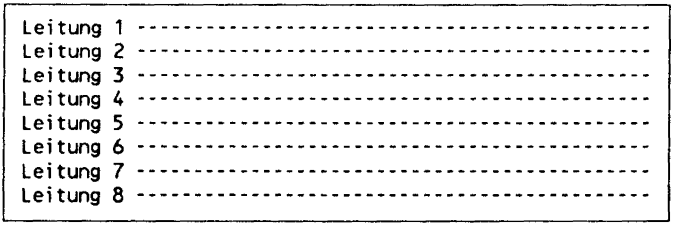
Der Begriff "Maschinensprache" fiel bereits ziemlich oft, ohne daß eine nähere Erläuterung erfolgte. Um dieses Versäumnis nachzuholen, benötigen wir ein wenig "trockene" Theorie.

Die Maschinensprache ist die einzige Sprache, die unser Computer wirklich versteht. Wie wir noch sehen werden, versteht er BASIC nicht, BASIC ist eine Fremdsprache für ihn.

Die eigentliche Maschinensprache besteht nicht aus Befehlswörtern, sondern aus "Zuständen" von Leitungen. Entweder fließt Strom oder es fließt kein Strom. Diese beiden Zustände einer Leitung kann unser Rechner unterscheiden (verstehen) und darauf reagieren.

Der eigentliche Rechner besteht aus der sogenannten "CPU" ("Central Processing Unit", "Zentraleinheit"). Diese Zentraleinheit versteht Befehle, die sich aus der Kombination der Zustände mehrerer Leitungen ergeben.

Diese Erläuterung klingt kompliziert, ist es jedoch nicht. Stellen wir uns acht Leitungen vor, die mit dem "Gehirn" unseres Computers - der CPU - verbunden sind.



Jede dieser acht Leitungen besitzt zwei mögliche Zustände: Strom fließt/kein Strom fließt. Diese beiden Zustände kennzeichnen wir mit je einer Zahl, eins (Strom fließt) und null (kein Strom fließt).

Insgesamt ergeben sich 256 mögliche Kombinationen von Nullen und Einsen, zum Beispiel:

```
        00000000
oder    00000001
oder    00000010
oder    10010010
oder    11101110
oder    11111111
...
...
```

Jede dieser Kombinationen wird von der CPU als ein Befehl interpretiert. Wie ein von uns erteilter Befehl die Kombination dieser Leitungszustände herstellt, interessiert uns nicht weiter.

Wichtig für uns ist jedoch, daß diese maximal 256 Befehle ähnlich wie BASIC-Befehle Namen besitzen, die wir bei der Programmierung eingeben.

Ein Beispiel für einen solchen Befehl ist der Befehl LDA, eine Abkürzung für "Load Accumulator". Diese Befehle werden "Assembler-Befehle" genannt und mit ihnen werden wir im folgenden programmieren. Anstelle des Ausdrucks "Maschinensprache-Programm" sollten wir daher besser von "Assembler-Programm" sprechen, da sich das Programm aus einzelnen von uns eingegebenen Assembler-Befehlen zusammensetzt.

Die Frage ist nun, wie Assembler-Programme eingegeben werden. Mit dem eingebauten BASIC-Interpreter ist die Eingabe nicht möglich, wie Sie sofort feststellen, wenn Sie zum Beispiel den Assembler-Befehl LDA eingeben und RETURN drücken. Zur Eingabe verwenden wir das erwähnte "Monitor-Programm".

Bitte beachten Sie bei der Benutzung eines Monitors für den C64 folgendes:

1. Die vielen Beispielprogramme sind im Bereich $C000 geschrieben. Viele Monitor-Programme liegen aber genau in diesem Bereich. Sie benötigen also entweder einen Monitor, der von vornherein in einem anderen Bereich liegt, oder müssen den Monitor von $C000 in einen anderen Bereich verlegen (was nicht ganz einfach ist) oder Sie schreiben die Beispielprogramme in einem anderen Bereich, beispielsweise $6000. Den hinten abgedruckten Maschinensprache-Monitor können Sie problemlos benutzen, weil er im Bereich ab $9000 liegt.
2. Der hinten abgedruckte Monitor gibt diese Zahlen nicht aus und versteht das "A" für Assemble nicht. Bei diesem Monitor geben Sie neue Befehle ein, in dem Sie die gewünschte Speicherstelle entweder mit "D" Disassemblieren und den gewünschten Befehl an die Stelle des vorhandenen Schreiben. Sie können auch einfach ein Komma vor die gewünschte Speicherstelle und dahinter den Befehl schreiben. Wir wollen diesen Unterschied kurz an einem Beispiel zeigen:

Beim abgedruckten und vielen anderen Monitoren des C64 geben Sie ein:

```
,C000 LDA $0300
```

Nach RETURN zeigt der Bildschirm:

```
,C000 LDA $0300
,C003
```

(Vor beiden Zeilen steht noch ein Punkt, den der Monitor automatisch ausgibt.)

Wenn Sie also den abgedruckten oder einen ähnlichen Monitor verwenden, so ändern Sie die Eingaben bitte entsprechend um.

Geben Sie bitte folgenden Befehl ein (und bestätigen Sie ihn wie üblich mit RETURN):

```
A C000 LDA #$01
```

Die eingegebene Zeile wird sofort in ein anderes Format umgewandelt werden:

```
A C000 A9 01    LDA #$01
A C002
```

Der Cursor befindet sich nun hinter der "Zahl" C002 und das Programm wartet auf eine weitere Eingabe. Geben Sie bitte der Reihe nach ebenso wie LDA #$01 die folgenden Befehle ein:

```
LDX #$00
STA $0400,X
DEX
BNE $C004
RTS
```

Nach jeder Eingabe wird in der nächsten Zeile eine "Zahl" vorgegeben und auf den nächsten Befehl gewartet. Betrachten Sie diese Vorgabe als eine Art AUTO-Befehl, der Ihnen nach Eingabe der ersten Zeilennummer eines BASIC-Programms alle weiteren Zeilennummern automatisch vorgibt.

Nachdem Sie den letzten Befehl RTS eingaben, drücken Sie bitte RETURN, ohne einen weiteren Befehl einzugeben, um die "Zeilenvorgabe" abzubrechen. Der Bildschirm zeigt nun folgendes "Programm":

```
A C000 A9 01    LDA #$01
A C002 A2 00                LDX #$00
A C004 9D 00 04 STA $0400,X
A C007 CA                   DEX
A C008 D0 FA    BNE $C004
A C00A 60                   RTS
A C00B
```

Sollten Sie einen Fehler begangen haben, gehen Sie bitte mit dem Cursor in die betreffende Zeile und korrigieren Sie den Assembler-Befehl wie in einem BASIC-Programm. Wichtig: korrigieren Sie bitte weder die "Zeilennummer" noch die folgenden zweistelligen "Zahlen", sondern immer nur den von Ihnen eingegebenen Assembler-Befehl!

Geben Sie anschließend den Buchstaben X ein und drücken Sie RETURN. Die Eingabe X führt zum "Verlassen" des Monitors und der BASIC-Interpreter meldet sich wie üblich mit READY.

Sie haben nun ohne den Umweg über die Data-Zeilen mit dem Monitor jenes Demoprogramm eingegeben, das 256mal das Zeichen A auf dem Bildschirm ausgab und damit Ihr erstes Assembler-Programm geschrieben. Starten Sie dieses Programm bitte mit dem gleichen Aufruf, den auch das BASIC-Programm verwendete.

```
SYS 49152
```

Sie können dieses Programm übrigens ebenso wie jedes BASIC-Programm mit diesem "SYS-Befehl" beliebig oft starten.

2.2 Dualsystem und Hexadezimalsystem

Vielleicht wundern Sie sich darüber, daß ich Zeichenfolgen wie C000 oder 03F7 als Zahlen bezeichne. Es sind tatsächlich Zahlen, jedoch keine Dezimal- sondern sogenannte "Hexadezimalzahlen". Unglücklicherweise können wir bei der Assembler-Programmierung mit unserem gewohnten Dezimalsystem nur wenig anfangen. Üblich ist die Verwendung des dualen und - wie bereits der Monitor zeigt - vor allem des hexadezimalen Systems.

Im Folgenden werde ich zuerst direkt auf das duale und das hexadezimale Zahlensystem eingehen. Anhand konkreter Beispiele werden die Unterschiede zum Dezimalsystem erläutert.

Das Dualsystem ist die Grundlage jeglicher "Computerei". Für jede Stelle einer Dualzahl steht nur eine von zwei verschiedenen

Ziffern zur Verfügung. Diese Ziffern sind 0 und 1 (zum Vergleich: im Dezimalsystem stehen bekanntlich zehn Ziffern zur Verfügung, 0,1,...,8,9).

Warum das Dualsystem sehr gut zur Zahlenverarbeitung durch einen Computer geeignet ist, wird deutlich, wenn wir uns erinnern, daß die CPU nur zwei verschiedene Leitungszustände unterscheiden kann, "Strom fließt" und "Strom fließt nicht", die durch eben jene beiden Ziffern 0 und 1 beschrieben werden können.

Eine Stelle einer Dualzahl, die wie erwähnt aus der Ziffer 0 oder der Ziffer 1 bestehen kann, nennt man ein "Bit". Acht solcher Bits bilden ein "Byte". Die Bits einer Dualzahl sind von rechts nach links durchnumeriert, wobei das erste Bit (ganz rechts) die Nummer null erhält (Bit 0, Bit 1,...,Bit 7).

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0

Die Tabelle enthält vier Bytes, das heißt, vier aus je acht Bits bestehende Dualzahlen. Die Frage ist nun, welchen Wert diese Dualzahlen besitzen. Der Wert hängt offensichtlich von der Kombination aus Nullen und Einsen ab. Enthält ein Bit die Ziffer 0, nennt man es "gelöscht". Ein "gesetztes" Bit wird durch die Ziffer 1 gekennzeichnet.

Um den Wert einer Dualzahl zu ermitteln, genügt es offensichtlich nicht, einfach alle gesetzten Bits zu addieren. Ebenso wie im Dezimalsystem ist der Wert einer Ziffer davon abhängig, an welcher Stelle sich die Ziffer in der Zahl befindet.

Um beim Dezimalsystem zu bleiben: mit jeder Stelle weiter links verzehnfacht sich der Wert einer Ziffer. Die Ziffer 5 besitzt an der ersten Stelle den Wert fünf, an der zweiten Stelle den Wert 50, an der dritten Stelle den Wert fünfhundert und so weiter.

Im Unterschied zum Dezimalsystem verdoppelt sich der Wert einer Ziffer im Dualsystem, wenn die Ziffer (0 oder 1) um eine Stelle nach links verschoben wird.

Die erste Dualzahl der Tabelle (00000000) besitzt natürlich den Wert null.

Die zweite Dualzahl (00000001) besitzt den Wert eins, die dritte Dualzahl (00000010) den Wert zwei und die vierte Dualzahl (00000100) den Wert vier.

Wenn man dieses Schema der Verdoppelung fortsetzt, ist leicht zu erkennen, daß bei nur einem gesetzten Bit der größte mit einer achtstelligen Dualzahl darstellbare Wert die Zahl 128 ist (10000000).

Sind mehrere Bits eines Bytes gesetzt, ergibt sich der Gesamtwert der Zahl, indem - ebenso wie im Dezimalsystem - die Werte aller Ziffern - abhängig von der Stelle in der Zahl - addiert werden.

Die Dezimalzahl 123 besitzt den Wert einhundertdreiundzwanzig, da die Ziffer 3 an der ersten Stelle den Wert drei besitzt, die Ziffer 2 an der zweiten Stelle den Wert zwanzig und die Ziffer 1 an der dritten Stelle den Wert einhundert. Addiert man diese Zahlen, ergibt sich der Gesamtwert.

Das Problem an dieser Analogie ist natürlich, daß wir aufgrund unseres ständigen Umgangs mit dem Dezimalsystem diese Berechnungen nicht mehr bewußt vornehmen, sondern die Ziffernfolge 1, 2, 3 (123) bereits als die Zahl einhundertdreiundzwanzig zu sehen glauben (was nicht stimmt, sondern nur bedeutet, daß die erforderlichen Berechnungen unbewußt ablaufen).

Versuchen Sie bitte, sich diese Berechnung des Wertes einer Zahl bewußt zu machen, indem Sie versuchen, eine Ziffer nicht sogleich als Zahl zu betrachten, sondern als Zeichen, dessen zahlenmäßiger Wert von seiner Position abhängt.

Gelingt Ihnen dies, ist die Übertragung des Schemas auf Dualzahlen ein Kinderspiel. Beispielsweise besitzt die Dualzahl 00000011 den dezimalen Wert drei, da die Ziffer 1 an der ersten Stelle den Wert eins besitzt und an der zweiten Stelle den Wert zwei.

Sehr einfach wird die Berechnung des Wertes einer Dualzahl mit Hilfe einer Tabelle, die den "Stellenwert" eines gesetzten Bits an den verschiedenen Stellen angibt.

```
Stellenwerte im Dualsystem
--------------------------

Bit 0 gesetzt => Wert 1
Bit 1 gesetzt => Wert 2
Bit 2 gesetzt => Wert 4
Bit 3 gesetzt => Wert 8
Bit 4 gesetzt => Wert 16
Bit 5 gesetzt => Wert 32
Bit 6 gesetzt => Wert 64
Bit 7 gesetzt => Wert 128
```

Zur Berechnung schauen Sie sich an, welche Bits gesetzt sind und addieren die betreffenden Stellenwerte.

Die Dualzahl 00010011 entspricht somit der Dezimalzahl 19 (1+2+16) und die Dualzahl 11111111 der Dezimalzahl 255 (1+2+4+8+16+32+64+128). Sie sehen, mit einem Byte - also einer achtstelligen Dualzahl - lassen sich durch Kombinationen gesetzter und gelöschter Bits alle Zahlen zwischen null (alle Bits gelöscht: 00000000) und 255 darstellen (alle Bits gesetzt: 11111111).

Das Hexadezimalsystem ist gewöhnungsbedürftiger als das Dualsystem, da es äußerst ungewohnte Ziffern enthält. Nicht nur 0 und 1 wie das Dualsystem oder 0, 1.., 8, 9 wie das Dezimalsystem, sondern die Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Die Ziffern A..F besitzen die "Ziffernwerte" zehn bis 15 (A=10, B=11, C=12, D=13, E=14, F=15). Das Hexadezimalsystem besitzt

einen wesentlichen Vorteil gegenüber dem Dual- und dem Dezimalsystem. Auch sehr grosse Zahlen können mit wenigen Ziffern dargestellt werden. Üblicherweise werden wir mit vierstelligen Hexadezimalzahlen arbeiten (Beispiel: C000, A000, 1AB5).

Im Hexadezimalsystem versechzehnfacht sich der Wert einer Ziffer mit jeder Stelle weiter links. Das heißt, die Ziffer 1 besitzt an der ersten Stelle (0001) den Wert eins, an der zweiten Stelle (0010) den Wert 16, an der dritten Stelle (0100) den Wert 256 und an der vierten Stelle (1000) bereits den Wert 4096.

Der Wert einer Hexadezimalzahl ergibt sich wiederum aus der Addition der Werte der einzelnen Ziffern unter Berücksichtigung ihrer Stellung in der Zahl. Um Ihnen die Berechnung zu erleichtern, folgt eine Tabelle der Stellenwerte des Hexadezimalsystems.

Stelle	Stellenwert
1	1
2	16
3	256
4	4096

Bei der Berechnung des Wertes einer Hexadezimalzahl gehen Sie so vor:

1. Multiplizieren Sie den Ziffernwert mit dem jeweiligen Stellenwert, um den Wert der Ziffer an der betreffenden Stelle zu ermitteln.
2. Addieren Sie alle ermittelten Werte.

Beispiele (Die Berechnungen erfolgen immer von rechts nach links!):

```
1. 0011 => 1*1  +  1*16 + 0*256  + 0*4096  =    17
2. 1100 => 0*1  +  0*16 + 1*256  + 1*4096  =  4112
3. 1111 => 1*1  +  1*16 + 1*256  + 1*4096  =  4369
4. 00FF => 15*1 + 15*16 + 0*256  + 0*4096  =   255
5. FF00 => 0*1  +  0*16 + 15*256 + 15*4096 = 65280
```

6. FFFF => 15*1 + 15*16 + 15*256 + 15*4096 = 68880
7. ABCD => 13*1 + 12*16 + 11*256 + 10*4096 = 43981

Sie sehen, mit einer vierstelligen Hexadezimalzahl lassen sich Zahlen darstellen, zu deren Darstellung im Dezimalsystem bereits sechs Stellen benötigt werden.

Beim Umgang mit verschiedenen Zahlensystem stellt sich das Problem der Unterscheidung. Wie erkennen wir, ob eine bestimmte Zahl eine Dual-, Dezimal-, oder aber eine Hexadezimalzahl ist? Dual- und Hexadezimalzahlen besitzen als Kennzeichnung ein Sonderzeichen, das der Zahl vorangestellt wird. Dualzahlen werden mit dem Zeichen "%", Hexadezimalzahlen mit dem Zeichen "$" markiert.

Dezimalzahlen	Dualzahlen	Hexadezimalzahlen
0	%00000000	$0000
16	%00010000	$0010
255	%11111111	$00FF

Sie brauchen übrigens nicht alle Hexadezimalzahlen auswendig zu lernen. Fast jeder Monitor kann unterschiedliche Zahlenformate umrechnen. Schauen Sie dazu bitte in die entsprechende Anleitung des Monitors.

2.3 Der Hauptspeicher

Jeder Computer besitzt einen "Arbeits-" oder auch "Hauptspeicher" (im Unterschied zu "Massenspeichern" wie Cassetten oder Disketten), der alle benötigten Daten und Programme aufnimmt. Wie und wo diese Speicherung vorgenommen wird, ist für uns uninteressant, wenn wir in BASIC programmieren.

Bei der Assembler-Programmierung ist es jedoch unumgänglich, in Grundzügen über den prinzipiellen Aufbau eines Hauptspeichers und vor allem über den Aufbau Ihres Rechners Bescheid zu wissen.

2.4 Speicherorganisation

Um den Begriff "Speicherorganisation" zu klären, ist ein kurzer Ausflug in die BASIC-Programmierung angebracht. Wenn Sie - zum Beispiel ein BASIC-Programm erstellen, wird dieses Programm und alle die Variablen, die es verwendet, im Rechnerspeicher abgelegt.

Aus dem bisher Gesagten geht hervor, daß die CPU dieses Programm nicht versteht, BASIC ist eine "Fremdsprache" für unseren Rechner. Sie alle kennen den Begriff "BASIC-Interpreter". Dieser Interpreter ist ein Assembler-Programm, das unsere BASIC-Befehle "interpretiert" und in Anweisungen der für die CPU verständlichen Assembler-Sprache umsetzt.

Das Assembler-Programm namens "BASIC-Interpreter" übernimmt auch die Aufgabe, unseren BASIC-"Text" in bestimmten Speicherbereichen abzulegen. Während der Duchführung eines BASIC-Programms werden fast immer Variablen benötigt (A=10, B%=23), denen Werte zugewiesen wird.

Wo diese Variablen gespeichert werden, ist wiederum dem BASIC-Interpreter überlassen und für den Benutzer uninteressant. BASIC ist eine sogenannte "problemorientierte" Sprache, im Gegensatz zur "hardwareorientierten" Sprache Assembler.

Da uns kein hilfreiches Programm zur Verfügung steht, das sich um die Speicherverwaltung kümmert, müssen wir selbst bei der Assembler-Programmierung angeben, wo unser Programm und die benötigten Variablen abzulegen sind.

Um einen geeigneten Ort zu bestimmen, müssen wir jedoch grundlegende Kenntnisse über die Speicherorganisation unseres Rechners besitzen.

Stellen Sie sich den Hauptspeicher eines Rechners bitte wie ein Hochhaus vor, das in einzelne Etagen unterteilt ist. Jede dieser Etagen ist eine "Speicherzelle", in der man eine Zahl ablegen kann.

Jede Speicherzelle kann eine beliebige achstellige Dualzahl aufnehmen, also ein Byte (eine Zahl zwischen null und 255). Der besitzet 65536 dieser Speicherzellen.

Sie sehen, aufgrund der großen Anzahl der Speicherzellen ist die Maßeinheit Byte ziemlich "unhandlich". Die Speicherkapazität eines Rechners wird üblicherweise in "Kilobyte" (Kb) gemessen (Ihr Gewicht messen Sie ebenfalls nicht in Gramm, sondern in Kilogramm).

Ein Kilobyte entspricht nun nicht - wie anzunehmen wäre - 1000, sondern 1024 Byte.

Jede dieser Speicherzellen kann mit einer beliebigen Zahl zwischen null und 255 - einem Byte - beschrieben werden. Um ein Byte in eine Speicherzelle zu schreiben, geben wir die "Hausnummer" (Etage in der "Hochhaus"-Analogie) der gewünschten Speicherzelle an, die sogenannte "Adresse".

Die Adressierung der Speicherzellen beginnt mit der Adresse null. Die höchste Adresse ist daher die letzte Speicherzelle, nämlich die Adresse 65536.

Merken Sie sich bitte: der Hauptspeicher unseres Rechners ist in Speicherzellen unterteilt, die über ihre Adresse (Hausnummer) angesprochen werden und beliebige Zahlen zwischen null und 255 aufnehmen können.

Der Bildschirmspeicher

Nicht nur Assembler, sondern auch BASIC erlaubt es uns, beliebige Werte in eine angegebene Speicherzelle zu schreiben, und zwar mit dem POKE-Befehl.

Wie Sie wissen, besitzt der POKE-Befehl das Format: POKE (ADRESSE),(ZAHL). Diesen POKE-Befehl werden wir nun verwenden, um ein "maschinennahes" Programm zu schreiben.

Wir füllen den Bildschirm mit A's, jedoch ohne den PRINT-Befehl zu verwenden.

Da Sie alle BASIC beherrschen, kennen Sie sicherlich die "Bildschirmcode-Tabelle", die sich im Handbuch Ihres Rechners eine Seite vor der "ASCII-Tabelle" befindet. Jedes Zeichen auf dem Bildschirm entspricht einer Zahl in den Speicherzellen des sogenannten "Bildschirmspeichers" oder "Video-RAM's".

Der Bildschirmspeicher besteht aus 1000 Speicherzellen. Jede dieser Speicherzellen "gehört" zu einer bestimmten Bildschirmposition und enthält den Bildschirmcode des Zeichens, das sich an dieser Position befindet.

Dieser Bildschirmspeicher beginnt beim C64 mit der Adresse 1024. Die Speicherzelle mit der Hausnummer 1024 enthält den Bildschirmcode des Zeichens, das sich an der HOME-Position des Cursors befindet, also an Spalte eins von Zeile eins.

Der Bildschirmspeicher ist fortlaufend organisiert. Die Speicherzelle 1025 enthält daher den Code des Zeichens an der Position 2/1 (Spalte zwei/Zeile eins) und so weiter.

Mit diesem Wissen und dem POKE-Befehl können wir bereits beliebige Zeichen "maschinennah" auf dem Bildschirm ausgeben. Mit POKE 1024,(BILDSCHIRMCODE) verändern Sie das Zeichen an Position 1/1 (Spalte eins/Zeile eins).

Mit einer Schleife können wir den Bildschirm problemlos mit dem Zeichen A füllen.

```
100 VR=1024:REM STARTADRESSE VIDEO-RAM
110 FOR I=VR TO VR+1000
120 : POKE I,1:REM BILDSCHIRMCODE VON 'A'
130 NEXT
```

Dieses Programm ist "maschinennah", da es direkt bestimmte Speicherzellen anspricht. Es arbeitet analog unserem ersten echten Assembler-Programm, das - wie wir noch sehen werden

- den Bildschirm auf die gleiche Art und Weise (Veränderung des Bildschirmspeichers) mit A's füllte.

2.5 Die Speicheraufteilung

Im letzten Kapitel sahen wir, wie der Speicher unseres Rechners organisiert ist (Speicherzellen mit einer "Breite" von einem Byte) und wo sich der Bildschirmspeicher befindet.

1. Die Zeropage

Eine "Page" oder "Seite" ist ein Speicherbereich, der 256 Speicherzellen umfaßt. Die erste Seite (Page null oder "Zeropage") umfaßt den Bereich von Speicherzelle null bis Speicherzelle 255, die zweite Seite beginnt bei Speicherzelle 256 und endet bei Zelle 511 und so weiter. Man spricht auch vom "Seitenkonzept" der Speicheraufteilung (Unterteilung in Blöcke von je 256 Byte Umfang).

Die Zeropage ist für die Assembler-Programmierung außerordentlich wichtig. Verschiedene Assembler-Befehle können nur mit Hilfe dieses Speicherbereichs verwendet werden. In diesem speziellen Bereich befinden sich zudem für uns wichtige Informationen. Beispielsweise geben zwei dieser Speicherzellen immer Auskunft über die aktuelle Cursorposition (Spalte und Zeile).

2. Der Stack

Der sogenannte "Stack" umfaßt ebenfalls eine Seite, das heißt 256 Speicherzellen. Er wird von der CPU benutzt, um sich beim Aufruf eines Unterprogramms zu merken, von wo der Aufruf kam, analog einem BASIC-Programm, bei dem das Programm nach dem Befehl RETURN mit jenem Befehl fortgesetzt wird, der dem Unterprogrammaufruf GOSUB folgt. Der Stack wird außerdem oft zur kurzzeitigen Speicherung von Daten verwendet, da sich der Rechner selbständig darum kümmert, wo die Daten abgelegt werden und somit für uns jeglicher "Verwaltungsaufwand" entfällt.

3. Der Bildschirmspeicher

Der "Bildschirmspeicher" oder das "Video-RAM" besteht aus vier Seiten, also insgesamt 1024 Speicherzellen. Dieser Bereich enthält die Bildschirmcodes aller auf dem Bildschirm vorhandenen Zeichen.

Die erste Zelle des Bildschirmspeichers '1024' enthält den Bildschirmcode des Zeichens, das sich an der Position 1/1 (Spalte eins/Zeile eins) befindet, die zweite Zelle den Code des Zeichens an Position 2/1 (Spalte zwei/Zeile eins) und so weiter.

4. Das BASIC-RAM

Das "BASIC-RAM" ist jener Speicherbereich, in dem der BASIC-Interpreter Programme und Variablen ablegt.

5. BASIC-ROM

Im "BASIC-ROM" befindet sich der BASIC-Interpreter, jenes festeingebaute Programm, das unsere BASIC-Programme interpretiert und für ihre Ausführung sorgt. Ebenso wie alle anderen "ROM"-Bereiche können die Inhalte dieser Speicherzellen gelesen, jedoch nicht beschrieben und damit verändert werden.

6. Das Zeichen-ROM

Das "Zeichen-ROM" enthält den Zeichensatz unseres Rechners. das heißt alle verfügbaren Zeichen in Form des "Punktmusters", aus dem sie aufgebaut sind.

7. Das Kernal

Das "Kernal" ist das sogenannte "Betriebssystem" des Rechners, ein wie der BASIC-Interpreter festeingebautes Assembler-Programm, das für alle Ein-/Ausgabeoperationen (Tastatur abfragen, Zeichen auf Drucker ausgeben etc.) zuständig ist. Ohne dieses Betriebssystem ist unser Rechner eine Ansammlung verschiedener elektronischer Bauteile, mit der wir nichts anfangen können.

Das Betriebssystem enthält viele für die Arbeit mit dem Rechner grundlegende Funktionen, die wir uns in späteren Kapiteln zunutze machen werden.

RAM und ROM

Wie diese Unterteilung zeigt, unterscheidet man "RAM"- und "ROM"-Speichereinheiten. Die ROM-Speicherzellen (ROM = "read only memory", "nur lesbare Speichereinheit") können nicht mit beliebigen Werten beschrieben werden. Sie enthalten fest "eingebrannte" Daten oder Programme, die auch dann erhalten bleiben, wenn der Rechner ausgeschaltet wird.

RAM-Speicherzellen (RAM = "random access memory", "Speicher mit wahlfreiem (Lesen oder Schreiben) Zugriff") verlieren ihre Inhalte sofort, wenn die Stromversorgung unterbrochen wird. Im Gegensatz zum ROM kann der Inhalt von RAM-Speicherzellen jedoch beliebig verändert werden.

Aus diesem Grund ist der RAM-Speicher für uns der interessantere von beiden. Im RAM-Speicher werden wir Assembler-Programme und benötigte Daten ablegen.

Eines der Grundprobleme bei der Assembler-Programmierung ist die Suche nach "freien" Speicherbereichen, die wir für unsere Programme nutzen können. In mehreren Bereichen des Speichers werden ständig benötigte Daten abgelegt, da wir nicht einfach überschreiben dürfen.

So ist zum Beispiel die Zeropage für uns vorläufig "tabu". Wie erläutert, enthält sie Informationen, die das Betriebssystem zu seiner Arbeit benötigt und die wir nicht einfach verändern dürfen.

Ein gut geeigneter Speicherbereich zur Ablage von Assembler-Programmen ist beim C64 der Bereich 49152 bis 53247, der normalerweise unbenutzt ist. In diesem vier Kilobyte (4096 Byte) großen Bereich werden wir unsere Assembler-Programme ablegen. Im folgenden werde ich zur Adreßangabe das Hexadezi-

malsystem benutzen. Hexadezimal ausgedrückt beginnt der Bereich bei Adresse $C000 und endet bei $CFFF.

2.6 Programmieren mit dem Monitor

Wie erläutert setzt dieses Buch die Verwendung eines Monitors zur Programmeingabe voraus. Der folgende Abschnitt zeigt, wie Programme mit einem Monitor eingegeben, editiert, gespeichert und geladen werden.

Die Unterschiede im Format der einzelnen Monitor-Befehle sind von Monitor zu Monitor äußerst gering, so daß sich dieser Abschnitt gleichermaßen auf den im Anhang abgedruckten oder einen beliebigen anderen Monitor bezieht.

Laden und Starten eines Monitor-Programms

Manche für den C64 erhältlichen Monitore müssen mit einem Befehl wie LOAD"NAME",8,1 geladen werden statt nur mit LOAD"NAME",8. Wichtig ist bei diesen Monitoren die Eingabe des Befehls NEW vor dem Starten des Monitors, da Sie ansonsten Schwierigkeiten mit eventuell einzugebenden BASIC-Programmen bekommen.

Wenn Sie einen Monitor erwerben oder abtippen, achten Sie bitte darauf, daß sich dieser nicht im Bereich $C000-$CFFF befindet (siehe Anleitung), da wir diesen Bereich im Folgenden zur Ablage unserer Programme verwenden.

2.6.1 Assembler-Programme mit einem Monitor eingeben

Jeder Monitor besitzt verschiedene Befehle, die uns bei der Erstellung eines Programms unterstützen. Um Assembler-Befehle mit einem Monitor einzugeben, verwenden wir den Befehl "Assemble", der bei jedem Monitor mit dem Buchstaben A abgekürzt wird. Dem Assemble-Befehl folgt die hexadezimale Angabe einer Adresse. Der folgende Assembler-Befehl wird an

eben dieser Adresse vom Monitor im Speicher abgelegt. Das allgemeine Format:

```
A (STARTADRESSE) (ASSEMBLER-BEFEHL)
```

Denken Sie bitte an den Unterschied zum abgedruckten Monitor. Geben Sie bei ihm bitte statt des "A" ein "," ein.

Monitore erwarten wie erläutert Zahleneingaben im Hexadezimalsystem. Da Startadressen immer hexadezimal anzugeben sind, können Sie bei den meisten Monitoren auf die Eingabe des Zeichens "$" vor der Startadresse verzichten.

Jeder Assembler-Befehl besitzt eine bestimmte Länge, der Befehl LDA #$01 zum Beispiel eine Länge von zwei Byte. Diesen Befehl legt der Monitor in den Speicherzellen $C000 und $C001 ab, wenn wir als Startadresse C000 angeben (A C000 LDA #$01). Die Adresse, ab der der nächste Befehl abgelegt wird, steht nun fest ($C000 + $02 = $C002) und muß vom Benutzer nicht mehr eingegeben werden.

Sobald Sie einmalig einen Befehl mit einer Startadresse eingeben, wird Ihnen der Monitor daher die Adresse des jeweils folgenden Befehls vorgeben und Sie müssen nur noch den Befehl selbst eintippen (siehe AUTO-Befehl in BASIC).

Merken Sie sich bitte: ein Assembler-Programm wird mit einem Monitor eingegeben, indem der erste Befehl mit der gewünschten Startadresse des Programms eingegeben wird. Bei allen folgenden Befehlen wird die jeweilige "Befehlsadresse" vom Monitor vorgegeben und es genügt vollkommen, den Befehl selbst einzugeben. Üben Sie diese Programmeingabe bitte mit dem folgenden Programm.

```
Rahmenfarben ändern
      A C000 LDX #$00
      TXA
      STA $D020
      DEX
      JMP $C002
```

Verlassen Sie den Monitor anschließend mit dem Befehl X und starten Sie das Assembler-Programm mit SYS 49152.

Wenn Sie das Programm korrekt eingaben, ergibt sich ein recht netter Effekt. Der Bildschirmrahmen wird von "Wellen" überzogen. Das Programm befindet sich in einer Endlosschleife, die nur mit STOP+RESTORE wieder verlassen werden kann.

Fehler bei der Eingabe korrigieren Sie, indem Sie den betreffenden Befehl übertippen und RETURN drücken, analog der Änderung einer BASIC-Befehlszeile.

In folgenden Kapiteln werden wir Assembler-Programme eingeben, die nicht mehr auf den Bildschirm passen. Zur Durchsicht und Korrektur des Programms benötigen wir daher ein Äquivalent zum LIST-Befehl.

Dieser Monitor-Befehl nennt sich "Dissasemble" und wird mit dem Buchstaben D abgekürzt. Das allgemeine Format (vergleiche LIST (VON)-(BIS)):

```
D (STARTADRESSE) (ENDADRESSE)
```

Geben Sie zum Listen des Programms daher ein:

```
D C000 C00A
```

Mit dem Disassemble-Befehl ist es jederzeit möglich, auch umfangreiche Programme zu listen und zu korrigieren. Ob bei der hexadezimalen Angabe des zu listenden Speicherbereichs das Zeichen "$" benötigt wird oder entfallen kann, ist von Monitor zu Monitor unterschiedlich.

Wenn Sie bei der "Dissasemblierung" eines Speicherbereichs seltsame Assembler-Befehle oder gar Fragezeichen im Anschluß an das eigentliche Programm entdecken, stören Sie sich bitte nicht daran. "Hinter" unserem Programm befinden sich ja ebenfalls Speicherzellen, die - unbestimmte - Werte enthalten. Der Monitor versucht diese Werte als Assembler-Befehle zu inter-

pretieren. Existiert zum Inhalt einer Speicherzelle kein zugehöriger Assembler-Befehl, gibt der Monitor ein Fragezeichen aus.

2.6.2 Speicherung eines Assembler-Programms im Rechner

Bei der Eingabe des letzten Demoprogramms stellten Sie fest, daß der Monitor unmittelbar nach der Eingabe eines Befehls die gesamte Befehlszeile in einem anderem Format darstellt. Der Befehlsadresse folgen mehrere zweistellige Hexadezimalzahlen ohne das Kennzeichen "$" (zumindest bei den meisten Monitoren), die jedoch an Ziffern wie A,B,C,D,E oder F zu erkennen sind. Diesen Hexadezimalzahlen folgt der von Ihnen eingegebene Assembler-Befehl. (Beim hinten abgedruckten Monitor werden diese Zahlen nicht ausgegeben.)

```
Rahmenfarben ändern
      A C000 A2 00      LDX #$00
      A C002 8A               TXA
      A C003 8D 20 D0         STA $D020
      A C006 CA               DEX
      A C00A 4C 02 C0         JMP $C002
```

Da Ihr Rechner prinzipiell nur Zahlen verarbeiten kann, wird auch ein Programm in Zahlenform "codiert" im Speicher abgelegt. Jedem Assembler-Befehl entspricht eine Zahl zwischen null und 255, ein Byte. Dieses Byte stellt der Monitor unmittelbar hinter der Adresse des Befehls in hexadezimaler Form dar. A2 (genauer: $A2) entspricht dem Befehl LDX, 8A dem Befehl TXA und so weiter.

(Sie brauchen diese Zahlen und die zugehörigen Befehle aber nicht auswendigzulernen, denn Sie haben ja einen Monitor, der die Befehle in Zahlen und die Zahlen in die zugehörigen Befehle umrechnet.)

Die meisten Befehle, die wir kennenlernen, benötigen ein "Argument", vergleichbar BASIC-Befehlen wie PEEK(ARGUMENT) oder INT(ARGUMENT). Das Argument ist eine ein oder zwei Byte lange Zahl.

Je nach Länge von Assembler-Befehl plus Argument spricht man von einem "Ein-", "Zwei-" oder "Drei-Byte-Befehl".

Dem Befehl LDX folgt das Argument 00 ($00). LDX #$00 ist daher ein Zwei-Byte-Befehl, JMP $C002 (4C 02 C0) ein Drei-Byte-Befehl mit dem Argument 02 C0.

Wenn Sie sich die zu den Befehlen gehörenden Zahlen mit dem hinten abgedruckten Monitor anschauen möchten, so benutzen Sie dazu bitte den Memory-Befehl "M". Um beispielsweise die Zahlen, die zu

```
,C000 LDA $0300
```

gehören zu sehen, geben Sie "M C000" + RETURN ein und Sie erhalten die Anzeige:

```
:C000 AD 00 03 .....
```

Um zu verdeutlichen, wozu Argumente benötigt werden: der Befehl JMP ist ein "Sprungbefehl", vergleichbar dem BASIC-Befehl GOTO, und setzt wie dieser die Programmverarbeitung an der angegebenen Stelle fort.

Das Sprungziel wird jedoch nicht in Form einer Zeilennummer, sondern einer Befehls-Adresse angegeben. Der Sprung erfolgt zur angegebenen Speicherzelle und das Programm wird mit dem in dieser Zelle enthaltenen Befehl fortgesetzt, in diesem Fall mit dem Befehl an Adresse $C002 (TXA).

Die Adresse $C002 ist das Argument des Befehls JMP. Der Befehl JMP wird in Speicherzelle $C00A abgelegt. Die Hexadezimalzahl $C002 wird in zwei Speicherzellen abgelegt. Der Grund: eine Speicherzelle kann nur Zahlen zwischen null und 255 enthalten, also ein Byte. Zur Darstellung einer größeren Zahl werden mehrere Bytes benötigt.

Mit zwei Bytes kann eine beliebige Zahl zwischen null und 65535 dargestellt werden. Verständlich wird die Aufteilung einer

großen Zahl in zwei Bytes jedoch erst, wenn wir uns die Hexadezimal-Darstellung von Zahlen betrachten.

Die Zahl 49154 wird hexadezimal in der Form $C002 dargestellt. Diese vierstellige hexadezimale Zahl wird in zwei Abschnitte unterteilt. Abschnitt eins enthält die ersten beiden Ziffern (C0), Abschnitt zwei die beiden letzten Ziffern (02).

Jeder dieser Abschnitte wird als eigene Zahl zwischen null ($00) und 255 ($FF) betrachtet. Die vierstellige Hexadezimalzahl wird in zwei Bytes unterteilt.

Man spricht von "höherwertigem" oder "High-Byte" (die beiden linken Ziffern) und "niederwertigem" oder "Low-Byte" (die beiden rechten Ziffern).

Diese beiden Bytes werden in zwei aufeinanderfolgenden Speicherzellen untergebracht und zwar bei allen Commodore-Computern in der etwas gewöhnungsbedürftigen Form "Low-Byte/High-Byte", dem sogenannten "Adreßformat".

In der ersten Speicherzelle wird die niederwertige, in der folgenden Speicherzelle die höherwertige Hälfte der Zahl untergebracht.

Diese Art der Adreß-Speicherung erklärt die Darstellung des Drei-Byte-Befehls JMP $C002 durch die drei Bytes $4C $02 $C0. Das Byte $4C ist der "Befehls-Code" des JMP-Befehls. $02 $C0 ist die Darstellung der Adresse $C002 in der Form Low-Byte/High-Byte.

Merken Sie sich diese Art der Adreß-Darstellung unbedingt. Wir werden viele Befehle kennenlernen, die mit Zwei-Byte-Adressen arbeiten. Bei allen diesen Adressen wird das niederwertige vor dem höherwertigen Byte gespeichert.

Aber keine Sorge: Durch unsere vielen Übungen werden Sie das schnell lernen und ein Monitor nimmt Ihnen diese Arbeit sowieso ab.

2.6.3 Programme speichern und laden

Assembler-Programme nützen uns recht wenig, wenn sie beim Ausschalten des Rechners jedesmal verlorengehen. Mit den BASIC-Befehlen SAVE und LOAD können wir jedoch nur BASIC- und keine Assembler-Programme speichern.

Jeder Monitor besitzt jedoch zwei gleichwertige Befehle, S (für SAVE) und L (für LOAD). Der Befehl S besitzt das Format:

```
S"(NAME)",(GERÄTENR.),(STARTADRESSE),(ENDADRESSE+1)
```

Um ein Assemblerprogramm zu speichern, müssen Sie ebenso wie bei einem BASIC-Programm einen Dateinamen (NAME) angeben. Anschließend folgt die (GERÄTENR.). Wollen Sie das Programm auf Cassette speichern, geben Sie 01 ein, zum Speichern auf Diskette hingegen die Gerätenummer 08.

Vielleicht wundern Sie sich, warum die Eingabe 1 oder 8 nicht genügt. Dies liegt an der bereits erwähnten Eigenheit von Monitoren, auf hexadezimalen Eingaben zu bestehen. 8 entspricht der zweistelligen Hexadezimalzahl $08 und 1 der Zahl $01 (wobei das Dollarzeichen jedoch nicht mit eingegeben werden muß).

Nun geben Sie an, wo sich das abzuspeichernde Programm befindet und wo es endet. Der Monitor wird genau diesen Speicherbereich auf Cassette beziehungsweise Diskette schreiben. Beide Angaben erfolgen wiederum hexadezimal. Ein Beispiel:

```
A C000 EE 20 D0          INC $D020
A C003 4C 00 C0          JMP $C000
A C006
```

Dieses kurze Programm verändert ununterbrochen die Farben des Bildschirmrahmens. Es beginnt bei der im Assemble-Befehl angegebenen Startadresse $C000 und endet bei Adresse $C005, ein Byte vor der nächsten Befehlsadresse $C006.

Beim Speichern können Sie auf das Dollarzeichen bei der Angabe der Start- und der Endadresse verzichten. Beachten Sie je-

doch, daß dem Monitor nicht die echte Endadresse, sondern die (ENDADRESSE+1) anzugeben ist. Im Beispiel wäre dies die Adresse des nächsten Befehls, $C006.

```
Speichern auf Cassette: S"TESTPROG",01,C000,C006
Speichern auf Diskette: S"TESTPROG",08,C000,C006
```

Beide Befehle speichern das Programm unter dem Namen TESTPROG auf Cassette beziehungsweise Diskette. Manche Monitore verlangen übrigens beim Speichern (und Laden) ein Leerzeichen anstelle eines Kommas. Welches dieser beiden Trennzeichen Ihr Monitor erwartet, müssen Sie ausprobieren.

Der Befehl L besitzt folgendes Format:

```
L"(NAME)",(GERÄTENR.)

Laden von Cassette: L"TESTPROG",01
Laden von Diskette: L"TESTPROG",08
```

Die Angabe der Start- und Endadresse entfällt beim Laden. Das Programm TESTPROG wird automatisch nach $C000 geladen, in jenen Bereich, an dem es sich beim Speichern befand.

Sie laden das Programm wie beschrieben, geben beim Speichern jedoch die für Sie gültige Start- und Endadresse (+1) ein.

```
Speichern auf Cassette: S"TESTPROG",01,03F7,03FD
Speichern auf Diskette: S"TESTPROG",08,03F7,03FD
```

Zur Übung empfehle ich Ihnen, den Bildschirminhalt zu speichern und - nach dem Löschen des Bildschirms - wieder zu laden. Dabei müssen Sie den Bereich $0400-$07FF speichern. Beachten Sie bitte, daß die Endadresse plus eins anzugeben ist.

2.7 Befehlsbearbeitung durch die CPU

Die CPU besitzt einen Programmzähler oder "program counter". Der Programmzähler weist immer auf die Adresse, an der sich der nächste zu bearbeitende Befehl befindet.

Da der gesamte Rechnerspeicher "addressiert" werden muß ($0000-$FFFF), ist diese Adresse zwei Byte lang (siehe oben). Der Programmzähler besteht aus zwei "Registern", speziellen Speicherzellen, die wie üblich jeweils ein Byte aufnehmen können und die nieder- beziehungsweise die höherwertige Hälfte der jeweiligen Adresse enthalten.

Nehmen wir an, der Programmzähler weist auf die Adresse $C000, an der sich der erste Befehl unseres Programms befindet (LDX #$00). Über den sogenannten "Datenbus" wird der Inhalt der Speicherzelle $C000 geholt, die Zahl $A2.

Die Zahl wird nun "decodiert" und als Befehl LDX erkannt. Anschließend wird der Programmzähler um eins erhöht und weist danach auf die Adresse $C001. Das "Argument" des Befehls LDX, die Zahl $00 befindet sich an dieser Adresse und wird ebenfalls mit Hilfe des Datenbusses vom Rechnerspeicher in die CPU übertragen.

Ob ein Befehls-Argument ein oder aber zwei Byte umfaßt, erkennt die CPU am Befehlscode. Nachdem das Befehlsargument vom Speicher zur CPU transportiert wurde, wird der Befehl ausgeführt.

Der Programmzähler wird bei jeder Übertragung eines Bytes um den Wert eins erhöht und weist somit nach der Bearbeitung eines Befehls (und seines Argumentes) auf den folgenden Befehl, der auf die gleiche Weise abgearbeitet wird.

Die CPU-Register

Im vorigen Abschnitt lernten wir zwei CPU-Register kennen, die zusammen den Programmzähler bilden. Die verschiedenen

CPU-Register sind die wichtigsten Speicherzellen, die unser Rechner besitzt.

1. Der Akkumulator: Das wichtigste CPU-Register ist der sogenannte "Akkumulator". Fast alle Operationen sind nur mit Hilfe des Akkumulators möglich. Um beispielsweise eine bestimmte Speicherzelle mit einem beliebigen Wert zu beschreiben (analog dem POKE-Befehl), muß dieser Wert zuerst in den Akkumulator "geladen" werden. Anschließend kann der Akkumulatorinhalt in die gewünschte Speicherzelle übertragen werden.
   Außer für den "Datentransfer" ist der Akkumulator für Berechnungen aller Art zuständig. Additionen oder Subtraktionen können nur in diesem speziellen Register stattfinden.
2. Die Indexregister: Die CPU besitzt zwei "Indexregister", die als "X-Register" und "Y-Register" bezeichnet werden. Beide Register werden vorwiegend als Schleifenzähler und zur Bearbeitung von Tabellen verwendet.
3. Das Status-Register: Das Statusregister enthält wie alle übrigen Register ein Byte. Dieses Byte gibt über den "Prozessor-Status" Auskunft, zum Beispiel über das Ergebnis von Subtraktionen oder Additionen. Am Inhalt des Statusregisters können wir zum Beispiel erkennen, ob das Ergebnis einer Addition größer oder kleiner als 255 ist, oder ob eine Programmschleife beendet ist.
4. Der Stapelzeiger: Der Stapel ist ein spezieller, eine Seite (256 Byte) langer Speicherbereich, der zur kurzfristigen Ablage von Informationen geeignet ist und mit Hilfe des "Stapelzeigers" verwaltet wird.

2.8 Die Adressierungsarten

Sie wissen bereits von BASIC-Programmen, daß es möglich ist, Variablen auf verschiedene Weise anzusprechen, zu "adressieren". Bei der direkten Adressierung geben Sie den Namen einer einfachen Variablen an, zum Beispiel PRINT A$.

Die indirekte Adressierung verwendet Arrayvariablen. Welche Variable angesprochen wird, bestimmt der Wert des angegebenen Index (PRINT A$(1):PRINT B%(2)). Auf die Variable wird "indirekt" zugegriffen.

Bei der Assembler-Programmierung steht uns eine Vielzahl verschiedener Addressierungsarten zur Verfügung, deren effektiver Einsatz ungemein wichtig ist.

Daher werde ich darauf verzichten, Ihnen sofort zu Beginn unseres Assembler-Kurses eine Vielzahl von Befehlen zu präsentieren. Das Wissen um die verschiedenen Adressierungsarten ist weitaus wichtiger als die Kenntniß auch des "allerletzten" Befehls.

Im Folgenden werden zwei grundlegende Assembler-Befehle erläutert, LDA und STA. Mit diesen Befehlen werden die meisten Adressierungsarten "durchgespielt". In den folgenden Kapiteln werden nach und nach weitere Befehle eingeführt, an denen zusätzliche Adressierungsmöglichkeiten demonstriert werden.

LDA und STA

Die Assembler-Programmierung besteht vorwiegend darin, Inhalte bestimmter Speicherzellen zu lesen oder zu verändern. Zu diesem Zweck gibt es die Klasse der "Transferbefehle", die für den Datentransport zuständig sind.

Die grundlegendsten und am häufigsten verwendeten Transferbefehle sind LDA und STA. Die "Kürzel" zur Bezeichnung von Assembler-Befehlen, die immer aus drei Buchstaben bestehen, werden als "Mnemonics" bezeichnet und sind so gewählt, daß die Funktion meist direkt aus dem Namen hervorgeht. Die Abkürzung LDA steht für "load accumulator", also "lade den Akkumulator (mit einem Wert)", die Abkürzung STA für "store accumulator", also "übertrage den Inhalt des Akkumulators (in eine Speicherzelle)".

Mit diesen grundlegenden Transferbefehlen ist es möglich, den Inhalt beliebiger Speicherzellen zu lesen und zu verändern. Zu

diesem Zweck existieren verschiedene "Adressierungsarten", die im Folgenden besprochen werden.

Unmittelbare und absolute Adressierung

Bei der einfachsten Adressierungsart, der "unmittelbaren Adressierung", folgt dem Befehl als Argument ein - üblicherweise hexadezimal angegebener - Wert.

Dem Befehl LDA ("lade den Akkumulator") folgt der Wert, mit dem der Akkumulator geladen werden soll. Da der Akkumulator ein Register mit einer "Breite" von acht Bit (einem Byte) ist, ist das Argument eine Zahl zwischen null und 255. Um die unmittelbare Adressierung von anderen Adressierungsarten zu unterscheiden, befindet sich vor dem eigentlichen Wert das Zeichen "#". Beispiele:

```
1.    LDA #$00 : Akkumulator mit dem Wert $00 (=dezimal null) laden.
2.    LDA #$80 : Akkumulator mit dem Wert $80 (=dezimal 128) laden.
3.    LDA #$FF : Akkumulator mit dem Wert $FF (=dezimal 255) laden.
```

Befehle, die mit unmittelbarer Adressierung arbeiten, sind immer Zwei-Byte-Befehle. Der Befehl LDA #$FF besteht aus einem Byte, das den Befehlscode selbst enthält, und einem weiteren Byte, das das Argument $FF enthält.

Erste sinnvolle Programme lassen sich durch Kombination der Befehle LDA und STA erstellen. Mit STA ("übertrage den Inhalt des Akkumulators") wird der aktuelle Akkumulator-Inhalt in die angegebene Speicherzelle kopiert.

Da dem Befehl STA kein Wert, sondern die Adresse einer Speicherzelle als Argument angegeben wird, ist die unmittelbare Adressierung mit STA nicht möglich.

Die einfachste Adressierungart, die STA kennt, ist die "absolute Adressierung", bei der als Argument eine "absolute" Speicheradresse angegeben wird. Beispiele:

1. STA $C000 : Akkumulator-Inhalt in Speicherzelle $C000 (=dezimal 49152) übertragen.
2. STA $C002 : Akkumulator-Inhalt in Speicherzelle $C002 (=dezimal 49154) übertragen.

Da dem Befehlscode des Wortes STA eine Zwei-Byte-Adresse folgt, handelt es sich bei allen Beispielen um Drei-Byte-Befehl.

Mit dem folgenden Programm können Sie die Wirkungsweise der unmittelbaren und der absoluten Adressierung erproben.

```
Unmittelbare und absolute Adressierung
      A C000 A9 01          LDA #$01
      A C002 8D 0A C0       STA $C00A
      A C005 00             BRK
```

Dieses Programm demonstriert zugleich einige bisher unbekannte Möglichkeiten von Monitoren. Starten Sie es bitte nicht auf die übliche Art und Weise (Monitor mit X verlassen, Programm von BASIC aus mit SYS 49152 beziehungsweise SYS 1015 aufrufen).

Jeder Monitor besitzt einen Befehl zum Starten eines Assembler-Programms namens GO, der mit dem Buchstaben G abgekürzt wird.

Dem Befehl selbst folgt die Startadresse des jeweiligen Programms.

```
G (STARTADRESSE)
```

Starten Sie das eingegebene Programm mit G C000. Das Programm arbeitet wie folgt:

1. Der Befehl LDA #$01 lädt den Akkumulator mit dem unmittelbar angegebenen Wert $01.
2. Der Befehl STA $C00A schreibt den momentanen Inhalt des Akkumulators in die absolut angegebene Speicherzelle $C00A.

3. Der bisher unbekannte Assembler-Befehl BRK unterbricht die Bearbeitung des laufenden Programms (analog dem BASIC-Befehl STOP) und bewirkt die Rückkehr "in" den Monitor.

Ob die beiden Befehle tatsächlich wie gewünscht ausgeführt wurden, können Sie mit dem Monitor-Befehl MEMORY (Abkürzung M) überprüfen.

```
M (STARTADRESSE) (ENDADRESSE)
```

Der MEMORY-Befehl zeigt beliebige Speicherbereiche an. Die Inhalte der betreffenden Speicherzellen werden in hexadezimaler Form ausgegeben. Viele Monitore geben diese Inhalte zusätzlich am rechten Bildschirmrand in ASCII-Form aus.

Ebenso wie beim BASIC-Befehl LIST ist die Angabe der Endadresse optional. Wird nur die Anfangsadresse eingegeben, gibt der Monitor eine Bildschirmzeile aus, die die Inhalte der ersten acht Speicherzellen ab der angegebenen Adresse wiedergibt. Um den Inhalt von Speicherzelle $C00A zu überprüfen, geben Sie je nach Rechner ein:

```
M C00A
```

Die erste ausgegebene Hexadezimalzahl entspricht dem Inhalt der angegebenen Speicherzelle. Wurde das Programm korrekt eingegeben, sollte diese Speicherzelle den Wert $01 besitzen.

Die meisten Monitor-Programme verzichten übrigens auf die Ausgabe des Zeichens "$" vor den einzelnen Werten, da die hexadezimale Form wie erläutert von Monitoren meist als selbstverständlich vorausgesetzt wird.

Das Programm zeigt zugleich weitere noch unbekannte Eigenschaften von Monitoren. Nach dem Start des Monitors und der Rückkehr aus einem Assembler-Programm (BRK-Befehl) sehen Sie auf dem Bildschirm die sogenannte "Registeranzeige".

Nach der Ausführung unseres Programms könnten die Register beispielsweise die folgenden Werte enthalten:

```
 PC  IRQ  SR AC XR YR SP
C006 EA31 30 01 00 01 F2
```

- PC : Programmzähler
- IRQ : Zeiger auf Systeminterrupt (IRQ)
- SR : Status-Register
- AC : Akkumulator
- XR : X-Register
- YR : Y-Register
- SP : Stack-Pointer

Diese Register-Anzeige gibt Auskunft über die momentan in den erwähnten Registern enthaltenen Werte. Zusätzlich erhalten wir Auskunft über den "IRQ" und den "Stack-Pointer", die uns jedoch vorläufig nicht weiter interessieren.

Wichtig für uns ist die Anzeige des Akkumulator-Inhalts. Der Akkumulator enthält nach dem BRK-Befehl den Wert $01, gemäß dem Befehl LDA #$01 ("lade den Akkumulator mit dem Wert $01"). Wie wir sehen, wurde der Inhalt des Akkumulators durch den folgenden Befehl STA $C00A nicht beeinflußt.

Transferbefehle, die den Inhalt einer Speicherzelle oder auch eines Register übertragen, verändern niemals den Originalwert. Ein Befehl wie STA kopiert den Inhalt des Akkumulators, der unverändert erhalten bleibt!

Mit diesen beiden Befehlen, LDA und STA, können wir bereits eine Menge anfangen. Zusammengenommen entsprechen Sie dem POKE-Befehl, der ebenfalls eine beliebige Zahl (zwischen null und 255) in eine angegebene Speicherzelle kopiert.

Mit LDA und STA können wir zum Beispiel Zeichen auf dem Bildschirm ausgeben, indem wir den Bildschirmcode des ge-

wünschten Zeichens mit LDA in den Akkumulator laden und den Akkumulatorinhalt mit STA in eine Speicherzelle des Video-RAM kopieren.

Als weiteres Demoprogramm bietet sich die Veränderung der Farben des Bildschirmrahmens an. Alle Commodore-Computer besitzen ein elektronisches Bauteil, einen "Chip", der für die Bildschirmsteuerung zuständig ist. Dieser Chip "VIC" genannt verfügt über spezielle Register - also Speicherzellen -, die wesentliche Parameter enthalten, zum Beispiel die aktuelle Farbe des Bildschirmrahmens.

Beim C64 besitzt dieses spezielle Register die "Hausnummer" $D020. Durch Verändern des Registerinhaltes können wir die Rahmenfarbe beliebig ändern.

```
Rahmenfarbe ändern
      A C000 A9 01            LDA #$01
      A C002 8D 20 0D         STA $D020
      A C005 00               BRK
```

Starten Sie das Programm wie zuvor mit G C000. Die Rahmenfarbe sollte sich ändern, außer die mit dem Wert $01 angegebene Farbe entspricht exakt jener, die der Rahmen momentan besitzt.

Es gibt allerdings auch Monitor-Programme, die die Farben nach einem BRK selber auf bestimmte Werte setzen. Das geht so schnell, daß Sie in einem solchen Fall kaum etwas von der Änderung sehen werden.

An diesem Programm können Sie zugleich das Editieren von Assembler-Programmen mit dem Monitor üben. Um dem Rahmen eine andere Farbe zu geben, ändern Sie den Wert $01 zum Beispiel in $0A. Wenn sich die vom Monitor angezeigte Befehlszeile

```
      A C000 A9 01            LDA #$01
```

noch auf dem Bildschirm befindet, genügt es, den eigentlichen Assembler-Befehl (LDA #$01) zu überschreiben und RETURN

zu drücken. Anschließend starten Sie das Programm erneut mit dem GO-Befehl. Nicht nur STA, sondern auch LDA erlaubt den Einsatz der absoluten Adressierung.

Anstelle eines Wertes kann dem LDA-Befehl eine absolute Speicheradresse angegeben werden (LDA $C000). In diesem Fall wird der Akkumulator mit dem Inhalt der angegebenen Speicherzelle geladen.

Die absolute Adressierung mit dem LDA-Befehls kann ebenfalls mit einem kleinen Assembler-Programm demonstriert werden.

```
Absolute Adressierung mit LDA
      A C000 AD 0A C0           LDA $C00A
      A C003 00                 BRK
```

Verlassen Sie anschließend den Monitor und geben Sie POKE 49162,10 ein. Rufen Sie den Monitor wieder auf und starten Sie das Programm mit SYS 49152.

Die Registeranzeige des Monitors sollte nach dem BRK-Befehl für den Inhalt des Akkumulators den Wert $0A (=dezimal zehn) anzeigen.

Um den Programmablauf zu klären, "listen" Sie bitte das Assembler-Programm mit dem Disassemble-Befehl ab Adresse $C000. Der Ablauf:

1. Der POKE-Befehl schreibt in Speicherzelle 49162 = hexadezimal $C00A den Wert zehn, also die Hexadezimalzahl $0A, bevor das Assembler-Programm gestartet wird.
2. Der erste Assembler-Befehl LDA $C00A kopiert den Inhalt der betreffenden Speicherzelle in den Akkumulator. Der Inhalt der Speicherzelle wird bei diesem Kopiervorgang nicht verändert.
3. Der BRK-Befehl unterbricht das Assembler-Programm und ruft den Monitor auf, dessen Registeranzeige den Wert $0A als aktuellen Akkumulator-Inhalt ausgibt.

Der Akkumulator wurde im Demoprogramm mit Hilfe der absoluten Adressierung mit dem Inhalt der angegebenen Speicherzelle geladen (LDA (ADRESSE)), im Gegensatz zur erläuterten unmittelbaren Adressierung, bei der der Akkumulator mit einem unmittelbar angegebenen Wert geladen wird (LDA #(WERT)).

Am Programm-Listing erkennen Sie, daß die Länge eines Befehls von der Art der Adressierung abhängt. Der Befehl LDA #$01 besitzt eine Länge von zwei, der Befehl LDA $C00A eine Länge von drei Byte.

Der Grund: in beiden Fällen ist der eigentliche Befehlscode genau ein Byte lang. Das Argument ist jedoch bei der unmittelbaren Adressierung ein, bei der absoluten Adressierung zwei Byte lang.

Das Programm-Listing deckt einen weiteren interessanten Unterschied auf. Außer der Befehlslänge hängt auch der Befehlscode von der Art der Adressierung ab. Dem Befehl LDA #$01 entspricht die Bytefolge A9 01, dem Befehl LDA $C00A die Bytefolge AD 0A C0.

Bei der unmittelbaren Adressierung wird der LDA-Befehl mit dem Code $A9 "verschlüsselt", bei der absoluten Adressierung jedoch mit $AD.

Auch dafür ist ein einleuchtender Grund vorhanden. Wie erläutert, holt die CPU bei der Befehlsbearbeitung zuerst den Befehlscode und anschließend das Befehlsargument. Das Problem für die CPU besteht nun darin, anhand des Befehlscodes zu erkennen, ob ein Ein- oder aber ein Zwei-Byte-Argument folgt. Die "Argumentlänge" ist von der Adressierungsart abhängig. Daher werden für einen Befehl verschiedene Befehlscodes benutzt, je nachdem, welche Adressierungsart verwendet wird.

Am Code $A9 erkennt die CPU, daß es sich um einen LDA-Befehl mit unmittelbarer Adressierung handelt und daher ein Ein-Byte-Argument folgt, im Gegensatz zum Code $AD, der einen LDA-Befehl mit absoluter Adressierung kennzeichnet, dem ein Zwei-Byte-Argument folgt.

Die implizite Adressierung

Sie wissen bereits, daß die - bei den verschiedenen Commodore-Rechnern leicht unterschiedliche, jedoch "vollkompatible" - CPU verschiedene Register besitzt. Den Akkumulator lernten wir hinreichend kennen.

Außer dem Akkumulator stehen uns zwei sogenannte "Indexregister" zur Verfügung, das "X-Register" und das "Y-Register".

Diese Register werden vorwiegend zur Schleifenbildung, aber auch zur kurzfristigen Speicherung von Daten verwendet. Jedes dieser Register ist ebenso wie der Akkumulator ein "Acht-Bit-Register" und kann daher eine beliebige Zahl zwischen null und 255 aufnehmen, ein Byte.

Analog den Befehlen LDA und STA existieren Transferbefehle, mit denen Daten in diese Register transportiert oder umgekehrt die Inhalte der Register in eine anzugebende Speicherzelle kopiert werden können.

LDX ("load X-Register", "lade das X-Register") lädt das X-Register mit einem Wert. LDY ("load Y-Register", "lade das Y-Register") besitzt die gleiche Funktion für das Y-Register. Beide Befehle arbeiten analog dem LDA-Befehl, so daß sowohl die unmittelbare (LDX #$FF; LDY #$FF) als auch die absolute (LDX $C00A; LDY $C00A) Adressierung eingesetzt werden kann.

Entsprechend dem Befehl STA, mit dem der Akkumulator-Inhalt in eine beliebige Speicherzelle kopiert wird, stehen auch für die Indexregister X und Y entsprechende Befehle zur Verfügung.

Mit STX ("store X-Register", "übertrage das X-Register") und STY ("store Y-Register", "übertrage das Y-Register") wird der Inhalt eines Indexregisters in eine anzugebende Speicherstelle kopiert.

Das Demoprogramm zum Ändern der Rahmenfarbe kann daher jederzeit so umgeschrieben werden, daß anstelle des Akkumulators eines der Indexregister verwendet wird.

```
Rahmenfarbe ändern
      A C000 A2 01          LDX #$01
      A C002 8E 20 0D       STX $D020
      A C005 00             BRK
```

Der Befehl LDX #$01 lädt das X-Register mit dem unmittelbar (Kennzeichen "#") angegebenen Wert $01. Der Befehl STX $D020 kopiert den Inhalt des X-Registers, das heißt den soeben in dieses Register geladenen Wert $01, in die absolut angegebene Speicherzelle $D020.

Das Programm ist ebenso zu starten wie die vorigen Demoprogramme. Nach dem BRK-Befehl und der Rückkehr in das Monitor-Programm erkennen Sie an der Registeranzeige, daß das X-Register tatsächlich den Wert $01 enthält.

Es sollte Ihnen leichtfallen, die neu kennengelernten Befehle LDX, LDY, STX und STY zu handhaben, da sie sich kaum von den Befehlen LDA und STA unterscheiden.

Wichtig für uns ist jedoch, daß zusätzliche Befehle zum Datenaustausch zwischen den verschiedenen Registern vorhanden sind. Der Inhalt des Akkumulators kann sowohl in das X- als auch in das Y-Register kopiert werden. Die Inhalte von X- und Y-Register lassen sich leider nicht direkt in das jeweils andere Indexregister übertragen.

Sowohl der Inhalt des X- als auch der Inhalt des Y-Registers kann jedoch in den Akkumulator kopiert werden. Bei keiner dieser Transfer-Operationen geht der Inhalt des ursprünglichen Registers verloren!

```
Datenaustausch zwischen den Registern (Akku, X und Y)
-    TAX ("transfer accumulator to X-Register"): der Inhalt des
       Akkumulators wird in das X-Register übertragen.
```

- TAY ("transfer accumulator to Y-Register"): der Inhalt des Akkumulators wird in das Y-Register übertragen.
- TYA ("transfer Y-Register to accumulator"): der Inhalt des Y-Registers wird in den Akkumulator übertragen.
- TXA ("transfer X-Register to accumulator"): der Inhalt des X-Registers wird in den Akkumulator übertragen.

Datenaustausch zwischen Arbeitsspeicher und den CPU-Registern

- LDA ("load accumulator"): der Akkumulator wird mit einem Wert geladen.
- LDX ("load X-Register"): das X-Register wird mit einem Wert geladen.
- LDY ("load Y-Register"): das Y-Register wird mit einem Wert geladen.
- STA ("store accumulator"): der Inhalt des Akkumulators wird in die angegebene Speicherzelle übertragen.
- STX ("store X-Register"): der Inhalt des X-Registers wird in die angegebene Speicherzelle übertragen.
- STY ("store Y-Register"): der Inhalt des Y-Registers wird in die angegebene Speicherzelle übertragen.

Die Transfer-Befehle zum Datenaustausch zwischen Registern verwenden nur eine Adressierungsart, die "implizite" Adressierung. "Implizit" bedeutet, daß das Argument im Befehl selbst bereits enthalten ist.

Für einen Befehl wie TXA wird keine weitere Angabe benötigt, da im Befehl selbst bereits enthalten, welcher Wert in den Akkumulator zu übertragen ist, eben der aktuelle Inhalt des X-Registers.

Bei dieser Befehlsklasse handelt es sich somit um Ein-Byte-Befehle, da dem Befehlscode kein Argument folgt.

Zur Übung schreiben wir ein Programm, das die ersten sechs Speicherzellen des Bildschirms abwechselnd mit den Zeichen A (Bildschirmcode eins) und B (Bildschirmcode zwei) beschreiben soll. Die herkömmliche Lösung:

```
Ausgabe der Zeichen A und B, Version 1
      A C000 A9 01          LDA #$01
      A C002 8D 00 04       STA $0400
      A C005 A9 02          LDA #$02
      A C007 8D 01 04       STA $0401
      A C00A A9 01          LDA #$01
      A C00C 8D 02 04       STA $0402
      A C00F A9 02          LDA #$02
      A C011 8D 03 04       STA $0403
      A C014 A9 01          LDA #$01
      A C016 8D 04 04       STA $0404
      A C019 A9 02          LDA #$02
      A C01B 8D 05 04       STA $0405
      A C01E 00             BRK
```

Löschen Sie bitte den Bildschirm, bevor Sie das Programm wie gewohnt mit G C000 starten, um ein eventuelles Scrollen zu vermeiden.

In der obersten Bildschirmzeile wird die Zeichenfolge ABABAB ausgegeben. Der Programmablauf:

1. Der Akkumulator wird mit dem Wert $01 geladen, dem Bildschirmcode des Zeichens A. Der Inhalt des Akkumulators wird in die erste Speicherzelle des Bildschirmspeicher kopiert. Auf dem Bildschirm erscheint das Zeichen A.
2. Der Akkumulator wird mit dem Wert $02 geladen, dem Bildschirmcode des Zeichens B. Der Inhalt des Akkumulators wird in die zweite Speicherzelle des Bildschirmspeicher kopiert. Auf dem Bildschirm erscheint unmittelbar neben dem A das Zeichen B.

Die restlichen vier Zeichen werden auf die gleiche Weise ausgegeben, in den folgenden Speicherzellen des Bildschirmspeichers.

Der Ablauf dieses Programms bereitet Ihnen wohl kaum Schwierigkeiten. Wenn doch, lesen Sie bitte die Kapitel über die Befehle LDA, STA, die unmittelbare und die absolute Adressierung noch einmal sorgfältig durch.

Viel interessanter ist nun jedoch der Einsatz der neu erlernten Transferbefehle in diesem Programm. Das ständig wiederkehrende LDA #$01 und LDA #$02 kann mit diesen Befehlen erheblich vereinfacht werden.

```
Ausgabe der Zeichen A und B, Version 2
      A C000 A2 01            LDX #$01
      A C002 A0 02            LDY #$02
      A C004 8A               TXA
      A C005 8D 00 04         STA $0400
      A C008 98               TYA
      A C009 8D 01 04         STA $0401
      A C00C 8A               TXA
      A C00D 8D 02 04         STA $0402
      A C010 98               TYA
      A C011 8D 03 04         STA $0403
      A C014 8A               TXA
      A C015 8D 04 04         STA $0404
      A C018 98               TYA
      A C019 8D 05 04         STA $0405
      A C01C 00               BRK
```

Der Programmablauf:

1. Das X- und das Y-Register werden am Programmanfang "initialisiert". Die beiden Register erhalten die im weiteren Programmverlauf immer wieder benötigten Werte $01 (X-Register) und $02 (Y-Register).
2. TXA kopiert den Inhalt des X-Registers ($01) in den Akkumulator, der daraufhin den Wert $01 enthält. Der Inhalt des Akkumulators wird mit STA in Speicherzelle $0400 übertragen, das erste A wird ausgegeben.
3. TYA überträgt den Inhalt des Y-Registers ($02) in den Akkumulator. Der neue Akkumulatorinhalt $02 wird in Speicherzelle $0401 geschrieben, das erste B erscheint auf dem Bildschirm.

Der unter 2. und 3. geschilderte Ablauf wiederholt sich hier nun mehrmals. Beachten Sie unbedingt, daß Transferbefehle wie TXA und TYA den Inhalt der ursprünglichen Speicherzelle (Re-

gister) nicht verändern. Der Inhalt des X- und des Y-Registers bleibt während des gesamten Programmablaufs erhalten.

Nur der Inhalt des Akkumulators wird durch diese Transferbefehle verändert.

Ein Vorteil dieses Programms besteht in der etwas geringeren Tipparbeit bei der Eingabe. Zudem ist das Programm ein wenig kürzer als die erste Version. Version 1 besitzt eine Länge von 31 Byte ($C000-$C01E), Version 2 eine Länge von 29 Byte ($C000-C01C).

Diese Unterschiede sind zweifellos gering. Bedenken Sie jedoch, daß das Programm zwei zusätzliche Befehle zur "Initialisierung" der Indexregister enthält, die bei dem relativ kurzen Programm ins Gewicht fallen, jedoch vernachlässigbar sind, wenn der gesamte Bildschirm mit A's und B's beschrieben wird.

In diesem Fall wird der Unterschied in der Programmlänge erheblich größer und es macht sich viel stärker bemerkbar, daß der Zwei-Byte-Befehl LDA (WERT) durch den Ein-Byte-Befehl TXA beziehungsweise TYA ersetzt wird.

Ein weiterer Unterschied betrifft die Ablaufgeschwindigkeit. Es ist unmittelbar einsichtig, daß der Prozessor gegenüber einem Befehl mit impliziter Adressierung wie TXA einen größeren Arbeitsaufwand erledigt, wenn er außer dem eigentlichen Befehlscode ein folgendes Argument aus dem Speicher lesen muß.

Vereinfacht ausgedrückt: Ein-Byte-Befehle werden schneller verarbeitet als Zwei-Byte-Befehle und diese wiederum schneller als Drei-Byte-Befehle.

Durch effektiven Einsatz der Indexregister können wir ein Programm daher erheblich beschleunigen. Zugegeben, die Ablaufgeschwindigkeit ist bei Assembler-Programmen weit weniger wichtig als bei Programmen, die in einer - erheblich langsameren - höheren Programmiersprache erstellt werden.

Dennoch werden auch Sie mit der Zeit wie jeder Assembler-Programmierer den Ehrgeiz entwickeln, Ihre Programm in Bezug auf Geschwindigkeit und Programmlänge so weit wie möglich zu optimieren.

Kurzadressierung (Zeropage-Adressierung)

Ich erwähnte bereits, daß die erste "Seite" des Rechnerspeichers, also der Bereich $0000-$00FF eine besondere Rolle spielt. Ausschließlich mit dieser "Zeropage" ist die "Kurzadressierung" oder auch "Zeropage"-Adressierung möglich.

Die Besonderheit der Zeropage liegt darin, daß es möglich ist, jede ihrer Speicherzellen mit nur einem Byte zu adressieren ($00-$FF anstelle von $0000-$00FF).

Bei der Zeropage-Adressierung entfällt daher das High-Byte der Adresse. Ein Beispiel:

```
Zeropage-Adressierung
      A C000 A9 01          LDA #$01
      A C002 85 FE          STA $FE
      A C004 00             BRK
```

Es ist nicht unbedingt nötig, dieses Programm zu starten. Es schreibt den Wert $01 in die völlig unbedeutende Speicherzelle $00FE, ohne daß eine Wirkung erfolgt.

Anhand des Programm-Listings erkennen Sie jedoch die Vorteile der Zeropage-Adressierung. Anstelle der absoluten Adresse $00FE geben Sie die Kurzform $FE ein. Für die Adresse wird nur ein Byte benötigt. Der Befehl ist daher kürzer und wird schneller ausgeführt.

Aufgrund dieser Vorteile empfiehlt es sich, Daten, auf die häufig zugegriffen werden muß, möglichst in unbenutzen Speicherzellen der Zeropage abzulegen und bei den entsprechenden Transferbefehlen (LDA, STA, LDX, STX, LDY, STY) die Zeropage-Adressierung einzusetzen.

Unbenutzt und hervorragend zur Zeropage-Adressierung geeignete Bereiche der Zeropage sind vor allem:

```
$FB-FE
```

Indizierte Adressierung

Die "indizierte Adressierung" ist eine der schwierigsten, aber auch flexibelsten Adressierungarten, die uns zur Verfügung steht. Da sich mit dieser Adressierungsart viele Probleme äußerst elegant lösen lassen, wird die indizierte Adressierung sehr häufig in Programmen eingesetzt.

Seit längerem verwenden wir bereits die Indexregister X und Y, ohne daß der Ausdruck "Indexregister" näher erläutert wurde. Der Ausdruck erklärt sich durch den Einsatz dieser Register bei der indizierten Adressierung.

"Indiziert" bedeutet, daß bei dieser Adressierungsart außer der eigentlichen Adresse auch ein "Index" eine Rolle spielt. Mit Indizes arbeiten Sie wahrscheinlich seit langem in Ihren BASIC-Programmen.

Üblicherweise genügt zum Zugriff auf eine bestimmte BASIC-Variable die Angabe des Variablennamens (PRINT A:PRINT F$...), mit einer Ausnahme: zum Zugriff auf eine Arrayvariable ist außer dem Namen des Arrays immer ein zusätzlicher Index anzugeben, wie im folgenden Beispiel:

```
100 FOR I=1 TO 10
110 : PRINT A$(I)
120 NEXT
```

In diesem Beispiel werden der Reihe nach die Inhalte der Arrayvariablen <A$(1)> bis <A$(10)> ausgegeben. Die Schleifenvariable <I> wird als Index benutzt.

Erst die Kombination aus dem Namen des Arrays und dem Index ergibt die komplette "Adresse" der Variablen.

Analog funktioniert die indizierte Adressierung. Die Adresse einer Speicherzelle ergibt sich aus einer angegebenen Adresse und einem zusätzlichen Index, genauer: aus der Adresse plus dem Inhalt eines der Indexregister, wie im folgenden Beispiel:

```
A C000 A2 04          LDX #$04
A C002 BD 00 04       LDA $0400,X
A C005 00             BRK
```

In diesem Beispiel (das Sie nicht einzugeben brauchen, entscheidend ist das dahinterstehende Prinzip) wird zuerst das Indexregister X mit dem Wert $04 geladen (LDX #$04).

Der folgende Befehl LDA $0400,X zeigt den Einsatz der indizierten Adressierung. Die Adresse ergibt sich aus der absolut angegebenen Adresse $0400 plus dem Inhalt des X-Registers ($04). Der Befehle LDA $0400,X lädt somit den Akkumulator mit dem Inhalt der Speicherzelle $0404.

Ein weiteres Beispiel:

```
A C000 A0 FF          LDY #$FF
A C002 B9 00 04       LDA $0400,Y
A C005 00             BRK
```

In diesem Beispiel wird das Y-Register mit $FF geladen. Mit dem Befehl LDA $0400,Y wird auf die Adresse $0400 plus dem Inhalt des Y-Registers zugegriffen, also auf die Speicherzelle $04FF.

Um zu sehen, daß die angesprochene Adresse auch wirklich vom Registerinhalt abhängt, geben Sie bitte zusätzlich folgendes Programm ein:

```
A C000 A9 01          LDA #$01
A C002 A0 00          LDY #$00
A C004 99 00 04       STA $0400,Y
A C007 00             BRK
```

Löschen Sie bitte den Bildschirm und starten Sie das Programm wie gewohnt mit G C000. In der linken oberen Ecke (erste Spalte der obersten Zeile) wird das Zeichen "A" ausgegeben.

Ändern Sie nun den Befehl LDY #$00, zum Beispiel in LDY #$01, und starten Sie das Programm erneut. Das "A" wird in nun in der zweiten Spalte der gleichen Zeile ausgegeben. Statt $0400 wird die Speicherzelle $0401 angesprochen.

Diese Adressierungsart stellt auf den ersten Blick nur eine umständlichere Version der Befehle STA $0400 beziehungsweise STA $0401 dar, mit denen einfacher direkt auf die gewünschten Speicherzellen zugegriffen werden kann.

Der Sinn der indizierten Adressierung liegt vorwiegend im Einsatz innerhalb von Programmschleifen, analog dem zuvor abgebildeten BASIC-Programm.

Diesem Ziel, der Programmierung von Schleifen, werden wir uns nun mit "Riesenschritten" nähern. Die folgenden Seiten liefern Ihnen das zur Schleifenbildung notwendige Wissen. Zusammen mit der indizierten Adressierung werden wir dann bereits in der Lage sein, höchst effektive Assembler-Programme zu erstellen.

Merken Sie sich bitte zur indizierten Adressierung:

1. Die Adresse ergibt sich aus der absolut angegebenen Adresse plus dem Inhalt des verwendeten Indexregisters.
2. Die implizite Adressierung kann nur den Inhalt des Akkumulators auf diese Weise übertragen. Ein Befehl wie STX $0400,X oder STX $0400,Y ist nicht zulässig!

Inkrementier- und Dekrementierbefehle

In Assembler-Programmen ist es oftmals notwendig, den Inhalt bestimmter Speicherzellen oder Register zu erhöhen beziehungsweise zu vermindern.

Zu diesem Zweck stehen uns die sogenannten "Inkrementier-" (Inkrementieren=erhöhen) beziehungsweise Dekrementier-Befehle (Dekrementieren=vermindern) zur Verfügung.

Der Befehl INC (ADRESSE) erhöht den Inhalt der angesprochenen Speicherzelle um eins. DEC (ADRESSE) vermindert den Inhalt der betreffenden Speicherzelle um eins.

Die Befehle lassen sich gut merken, wenn wir wissen, daß INC die Abkürzung für "increment" und DEC die Abkürzung für "decrement" ist.

Mit diesen Befehlen können wir sehr einfach ein Programm schreiben, mit dem wir uns eine Bildschschirmrahmenfarbe nach unserem Geschmack aussuchen können.

```
Rahmenfarbe ändern
      A C000 EE 20 D0          INC $D020
      A C003 00                BRK
      A C004 CE 20 D0          DEC $D020
      A C007 00                BRK
```

Eigentlich handelt es sich hierbei um zwei Programme. Das erste Programm besteht aus dem Befehl INC $D020. $D020 ist jene Speicherzelle, die die Farbe des Bildschirmrahmens in Form einer Zahl enthält. Wird diese Zahl verändert, ändert sich die Rahmenfarbe entsprechend.

Rufen Sie das Programm wie gewohnt mit G C000 auf. Der Befehl INC $D020 erhöht den Inhalt dieser Speicherzelle um eins und die Rahmenfarbe ändert sich. Der BRK-Befehl unterbricht das Programm und führt zur Rückkehr in den Monitor. Jeder neue Aufruf des Programms ändert die Rahmenfarbe.

Dies funktioniert allerdings nur sichtbar, wenn der Monitor nicht anschließend beim BRK eigenständig die Farben zurücksetzt.

Auf diese Weise können Sie problemlos alle Farben ausprobieren. Sollten Sie über das Ziel "hinausgeschossen" sein und Ihnen die vorige Farbe besser gefallen, rufen Sie einfach den zweiten Programmteil mit G C004 auf. Der Befehl DEC $D020 vermindert den Inhalt von $D020 wieder um eins.

Zusammen erlauben Ihnen die Programme, den "Farbkatalog" Ihres Rechners vor- und rückwärts zu durchblättern.

Nicht nur der Inhalt von Speicherzellen, auch die Inhalte der Indexregister X und Y können inkrementiert/dekrementiert werden. Der Befehl INX ("increment X") inkrementiert den Inhalt des X-Registers, und der Befehl INY ("increment Y") inkrementiert das Y-Register. Entsprechend dekrementiert der Befehl DEX ("decrement X") das X-Register und der Befehl DEY ("decrement Y") das Y-Register.

Alle vier Befehle (INX, INY, DEX, DEY) verwenden die implizite Adressierung. Die Adresse des Registers, auf die sich der Befehl beziehen soll, ist bereits im Befehlswort selbst enthalten.

Entsprechende Befehle zur Erhöhung/Verminderung des Akkumulators um eins sind leider nicht vorhanden. Die gleiche Wirkung kann jedoch über einen "Umweg" erzielt werden. Der Inhalt des Akkumulators wird in eines der Indexregister übertragen (TAX oder TAY), das Indexregister in- oder dekrementiert (INX, INY beziehungsweise DEX, DEY) und der neue Inhalt des betreffenden Registers in den Akkumulator zurückgeschrieben (TXA oder TYA):

```
Akkumulator inkrementieren     Akkumulator dekrementieren
  TAX              TAY           TAX              TAY
  INX      oder    INY           DEX      oder    DEY
  TXA              TYA           TXA              TYA
```

Inkrementier-/Dekrementierbefehle

- INC (ADRESSE): Erhöht den Inhalt der angesprochenen Speicherzelle um eins.
- DEC (ADRESSE): Vermindert den Inhalt der angesprochenen Speicherzelle um eins.
- INX: Erhöht den Inhalt des X-Registers um eins.
- DEX: Vermindert den Inhalt des X-Registers um eins.
- INY: Erhöht den Inhalt des Y-Registers um eins.
- DEY: Vermindert den Inhalt des Y-Registers um eins.

Um den Umgang mit dieser Vielzahl neuer - aber sehr ähnlicher - Befehle zu erlernen, sollten Sie mehrere kleine Programme damit erstellen.

2.9 Programmschleifen

Der folgende Abschnitt führt Sie in die Schleifenbildung ein. Ebenso wie in BASIC sind auch in Assembler größere Programme ohne den Einsatz von Schleifen undenkbar.

Assembler unterscheidet sich von BASIC jedoch unter anderem durch die enorme Vielfalt verschiedener Schleifenbefehle, die die unterschiedlichsten Schleifenkonstruktionen ermöglicht.

Im Folgenden wird anhand eines Schleifenbefehls - eines "Branch"-Befehls - die Grundform einer Assembler-Schleife demonstriert.

2.9.1 Schleifen bilden mit BNE

Sie ahnen bestimmt, daß es mit den In- und Dekrementierbefehlen möglich ist, Schleifen zu bilden. In BASIC werden Schleifen ja ebenfalls gebildet, indem eine Zählvariable erhöht oder vermindert wird. Der Schleifenzähler wird in Assembler-Programmen meist nach jedem Durchgang um eins vermindert, wie auch im folgenden BASIC-Programm:

```
100 I=100:REM STARTWERT FUER SCHLEIFENZAEHLER
110 POKE 53280,I:REM SCHLEIFENBEFEHLE
120 I=I-1:REM SCHLEIFENZAEHLER VERMINDERN
130 IF I<>0 THEN GOTO 110:REM BEREITS 100 DURCHGAENGE?
```

Analog diesem BASIC-Programm werden auch in Assembler Schleifen gebildet. Eine Zählvariable erhält einen "Startwert", die Schleifenbefehle werden bearbeitet und die Zählvariable am Schleifenende vermindert.

Diesen Vorgang können wir problemlos mit den kennengelernten Befehlen nachbilden. Als Zählvariable verwenden wir eines der zwei Indexregister X und Y, da der Inhalt dieser Register mit einem kurzen Ein-Byte-Befehl problemlos erhöht oder vermindert werden kann.

```
A C000 A2 64          LDX #$64
A C002 8E 20 D0       STX $D020
A C005 CA             DEX
```

Der Befehl LDX #$64 lädt das X-Register - unseren Schleifenzähler - mit dem Wert 100, analog dem BASIC-Befehl I=100. Der Befehl STX $D020 entspricht dem Befehl POKE 53280,I und überträgt den Inhalt des X-Registers in $D020, jene Speicherzelle, die die Rahmenfarbe bestimmt. DEX vermindert den Inhalt unserer "Zählvariablen" ebenso wie I=I-1 um eins.

Den BASIC-Befehl IF I<>0 THEN GOTO 110 können wir mit dem Erlernten noch nicht umsetzen. Uns fehlt ein Vergleichsbefehl, mit dem wir in Abhängigkeit von einer Bedingung zum Schleifenanfang zurückkehren können.

Der benötigte Befehl lautet BNE (ADRESSE). Es handelt sich um einen sogenannten "Branch-" oder auch "Verzweigungs"-Befehl. BNE entspricht der Kombination der BASIC-Befehle IF und GOTO. Die dem IF folgende Bedingung ist allerdings festgelegt. Verzweigt wird immer dann, wenn die Zählvariable ungleich null ist.

Zur Erläuterung muß ich kurz auf das sogenannte "Status-Register" des Prozessors eingehen. Ich erwähnte bereits, daß uns dieses Statusregister Auskunft über verschiedene "Zustände" des Rechners gibt.

Das Status-Register enthält wie üblich ein Byte, also acht Bits. Jedes dieser Bits besitzt eine spezielle Bedeutung. Uns interessiert momentan ausschließlich Bit 1, das sogenannte "Zero-Flag". Immer dann, wenn ein Befehl, der eine Speicherzelle oder ein Register verändert (INY, DEX, DEC...), das Ergebnis null zur Folge hat, wird dieses Bit gesetzt und erhält den Wert eins.

Beispiel: Wenn das X-Register den Inhalt $01 besitzt, führt ein nachfolgender DEX-Befehl zum Ergebnis null im X-Register. Automatisch wird nun im Status-Register das Zero-Flag gesetzt, das Bit 1 des Status-Registers enthält die Ziffer eins..

Im Gegensatz dazu wird dieses Flag gelöscht (enthält die Ziffer null), wenn die vorhergehende Operation zu irgendeinem Ergebnis ungleich null führte.

Beispiel: Wenn das X-Register den Inhalt eins besitzt, führt ein folgender INX-Befehl zum Ergebnis zwei (X enthält nun den Wert zwei). Da das Ergebnis des DEX-Befehls zwei ist, also ein Wert ungleich null, wird das Zero-Flag gelöscht.

Wichtig für uns ist an diesem Vorgang, daß wir uns nicht im geringsten um dieses Setzen oder Löschen des Zero-Flags kümmern müssen. Diese Aufgabe übernimmt der Rechner für uns.

Der BNE-Befehl überprüft, ob das Zero-Flag momentan gesetzt oder aber gelöscht ist. Wenn es gesetzt ist - also der vorausgegangene DEX-Befehl oder eine beliebige andere Operation zum Ergebnis null führte - erfolgt ein Sprung zur angegebenen Adresse. Die Arbeitsweise dieses Befehls verdeutlicht der volle Befehlsname. BNE bedeutet "branch if not equal zero", "verzweige, wenn nicht gleich null".

Mit dem BNE-Befehl können wir unser Programm vervollständigen.

```
Assembler                              BASIC
-------------------------------------------------------------
A C000 A2 64     LDX #$64              100 I=100
A C002 8E 20 D0  STX $D020             110 POKE 53280,I
A C005 CA        DEX                   120 I=I-1
A C006 D0 FA     BNE $C002             130 IF I<>0 THEN GOTO 20
A C008 00        BRK
```

Dieses Programm entspricht nun exakt dem BASIC-Programm zur Änderung der Rahmenfarbe. Wenn Sie es auf Ihrem C64 starten, werden Sie allerdings von dieser Veränderung sehr wenig mitbekommen. Das obige Programm schreibt die Werte 100,99,..,1,0 einfach zu schnell in die Speicherzelle $D020. Die Schleife läuft für das menschliche Auge mit viel zu hoher Geschwindigkeit ab.

Der Programmablauf:

1. Das X-Register wird durch LDX #$64 mit dem Wert $64 (=dezimal 100) geladen.
2. Dieser Wert wird in die Speicherzelle $D020 kopiert (STX $D020) und dadurch die Rahmenfarbe entsprechend geändert.
3. Der Befehl DEX vermindert den Inhalt des X-Registers um eins, das nun den Wert $63 (=dezimal 99) enthält. Beachten Sie bitte, daß diese Operation ein Ergebnis besitzt, das ungleich null ist, nämlich 99, und das Zero-Flag gelöscht wird.
4. Der Befehl BNE $C002 überprüft anhand des Zero-Flags, ob die letzte Operation das Ergebnis null erbrachte, was an einem gesetzten Zero-Flag erkannt wird. Da dies nicht der Fall war (das Zero-Flag ist gelöscht), wird der Sprung zu Adresse $C002 ausgeführt.

Beachten Sie bitte, daß es für die praktische Benutzung dieses Befehls völlig uninteressant ist, wie das Zero-Flag überprüft wird. Uns interessiert nur, daß das Zero-Flag gesetzt wird, wenn

der Befehl DEX zum Ergebnis null führt, und der folgende BNE-Befehl dieses Ergebnis anhand des Zero-Flags überprüfen kann.

Solange der DEX-Befehl nicht zum Ergebnis null führt, ist die Bedingung "verzweige, wenn nicht gleich null" erfüllt und der nächste Schleifendurchgang beginnt. Die Frage ist nun, wann die Schleife beendet ist.

Bei jedem Schleifendurchgang wird der Inhalt des X-Registers mit dem Befehl DEX um eins vermindert. Nach 99 Durchgängen enthält das X-Register somit den Wert eins. Nun folgt wiederum der DEX-Befehl. Das X-Register wird dekrementiert und erhält den Wert null.

Der DEX-Befehl führte somit zum Ergebnis null und die Bedingung "verzweige, wenn nicht gleich null" ist nicht erfüllt. Diesmal verzweigt der BNE-Befehl nicht zum Befehl an Adresse $C002, zum Schleifenanfang.

Das Programm wird wie in BASIC mit jenem Befehl fortgesetzt, der dem BNE-Befehl folgt (BRK), die Schleife ist nach dem hundersten Durchlauf beendet.

Da Schleifen von ungeheuerer Wichtigkeit bei der Programmierung sind, fasse ich das Gesagte kurz zusammen:

1. Um Schleifen zu bilden, laden wir eines der Indexregister mit einem Startwert, der der gewünschten Anzahl von Schleifendurchgängen entspricht.
2. Wie von BASIC gewohnt, folgen die Befehle, die innerhalb der Schleife bearbeitet werden sollen.
3. Am Schleifenende dekrementieren wir unser "Zählregister" (X- oder Y-Register).
4. Es folgt ein BNE-Befehl mit der Adresse des ersten Schleifenbefehls. Solange der vorhergehende Befehl, die Dekrementierung des Indexregisters, nicht den Wert null ergibt, verzweigt das Programm zur angegebenen Adresse.

2.9.2 Verzögerungsschleifen

Da wir im folgenden sehr häufig mit Schleifen arbeiten werden, ist es angesichts der Geschwindigkeit von Assembler angebracht, für eine gewisse Verzögerung zu sorgen, um den Ablauf der Demoprogramme verfolgen zu können.

Sie wissen bereits von BASIC-Programmen, was unter einer "Verzögerungsschleife" zu verstehen ist: eine Schleife, die keinerlei Funktion außer der Verlangsamung des Programmablaufs besitzt. Eine solche Schleife besteht aus Schleifenanfang und Schleifenende ohne die sonst üblichen Befehle innerhalb der Schleife. In BASIC:

```
100 FOR I=1 TO 1000
110 NEXT I
```

Auch in Assembler sind derartige Verzögerungsschleifen möglich. Die maximale Anzahl an Schleifendurchgängen beträgt jedoch 255, wie im folgenden Programm.

```
A C000 A2 FF          LDX #$FF
A C002 CA             DEX
A C003 DO FD          BNE $C002
```

Assembler ist jedoch derartig schnell, daß selbst 255 Durchgänge für eine Verzögerungsschleife noch viel zu wenig sind. Die vorgestellte Schleife wird in kaum feststellbarer Zeit abgearbeitet.

Um die Verzögerungszeit zu verlängern, wäre nun die Schachtelung zweier Schleifen möglich, die ineinander geschachtelt werden, analog dem folgenden BASIC-Programm.

```
100 FOR I=1 TO 255
110 : FOR J=1 TO 255
120 : NEXT J
130 NEXT I
```

Die innere Schleife wird in diesem Programm 255mal durchlaufen, bevor ein weiterer Durchgang der äußeren Schleife beginnt,

die ebenfalls 255mal durchlaufen wird. Es ergeben sich somit 255*255=65025 Schleifendurchgänge. Das entsprechende Assembler-Programm:

```
Verzögerungsschleifen
      A C000 A2 FF          LDX #$FF
      A C002 A0 FF          LDY #$FF
      A C004 88             DEY
      A C005 D0 FD          BNE $C004
      A C007 CA             DEX
      A C008 D0 F8          BNE $C002
      A C00A 00             BRK
```

Wichtig für das Verständnis dieses Programms ist das Erkennen der Schleifenführung. Die innere Schleife besteht aus den Befehlen:

```
            LDY #$FF
(ADRESSE)   DEY
            BNE $(ADRESSE)
```

Das Y-Register wird mit dem Wert $FF "initialisiert". Der folgende Befehl DEY dekrementiert Y, das anschließend den Wert $FE enthält. Da das Ergebnis von DEY ungleich null ist, wird der bedingte Sprung BNE ausgeführt und der nächste Schleifendurchgang beginnt. Nach 255 Durchgängen führt der DEY-Befehl zum neuen Inhalt null des Y-Registers. Aufgrund dieses Ergebnisses null ist die Bedingung BNE ("verzweige, wenn ungleich null") nicht erfüllt und die innere Schleife wird verlassen.

Der folgende Befehl DEX dekrementiert den Schleifenzähler der äußeren Schleife, das X-Register. Solange dieser DEX-Befehl nicht den Wert null ergibt, wird der bedingte Sprung BNE zum Beginn der äußeren Schleife ausgeführt. Das Y-Register wird erneut mit dem Startwert $FF geladen und die innere Schleife erneut 255mal durchlaufen.

Nach dem 255ten Durchlauf der äußeren Schleife führt der Befehl DEX zum Resultat null als neuem Inhalt des X-Registers. Da der folgende bedingte Sprung somit nicht mehr ausgeführt

wird, wird das Programm mit dem ersten Befehl fortgesetzt, der der Verzögerungsroutine folgt.

Diese geschachtelten Verzögerungsschleifen bewirken eine Verzögerung von knapp einer Sekunde, die für unsere Zwecke völlig ausreicht.

Anhand dieses Programms will ich Ihnen jedoch zum ersten Mal einen Vorgeschmack auf die "Betriebssystem-Routinen" geben. Das Betriebssystem stellt uns eine Unmenge nützlicher Unterprogramme (Routinen) zur Verfügung, unter anderem eine Verzögerungsroutine.

Die Verzögerungszeit dieser Routine beträgt eine Millisekunde. Wenn wir den Aufruf dieser Routine in einer Schleife 255mal vornehmen, beträgt die Gesamtverzögerung somit 255 Millisekunden oder eine viertel Sekunde.

Das Prinzip:

```
          LDX #$FF
(ADRESSE) JSR $(VERZÖGERUNG)
          DEX
          BNE (ADRESSE)
```

Der Aufruf der Verzögerungsroutine erfolgt mit dem Befehl JSR $EEB3, der dem BASIC-Befehl GOSUB entspricht.

Nach jedem Aufruf wird das zu Beginn mit $FF geladene X-Register dekrementiert. Nach dem 255ten Durchlauf führt die Dekrementierung zum Ergebnis null und die Schleife wird verlassen, der bedingte Sprung nicht mehr ausgeführt.

Verzögerung mit Betriebssystem-Routine

```
A C000 A2 FF          LDX #$FF
A C002 29 B3 EE       JSR $EEB3
A C005 CA             DEX
A C006 DO FA          BNE $C002
A C008 00             BRK
```

Beim Einbau dieser Verzögerungsroutine in unsere Programme ist zu beachten, daß durch die Schleife der ursprüngliche Wert des X-Registers natürlich verändert wird.

Wenn die Schleife verlassen wird, besitzt das X-Register den Wert null, unabhängig vom Wert, den es vor dem Eintritt in die Schleife besaß.

Die Verzögerungsroutine wird an der gewünschten Position in ein Programm eingefügt. Soll der ursprüngliche Inhalt des X-Registers auch nach dem Durchlaufen der Schleife erhalten bleiben, verwenden wir am besten das Y-Register als "Zählvariable".

Beim Einbau des Programms muß weiterhin beachtet werden, daß auch der Inhalt des Akkumulators verändert wird, und zwar durch die aufgerufene Betriebssystem-Routine (obwohl dies anhand des Aufrufs nicht unmittelbar zu erkennen ist).

Um die Routine praktisch zu erproben, werden wir sie in das Programm zur Änderung der Bildschirmfarben einbauen.

Da in diesem Programm (siehe oben) der Inhalt des X-Registers durch die Verzögerungsschleife nicht willkürlich verändert werden darf, verwenden wir das Y-Register als Schleifenzähler.

```
Bildschirmfarben verzögert ändern

A C000 A2 64            LDX #$64
A C002 8E 20 D0         STX $D020
A C005 A0 FF            LDY #$FF
A C007 20 B3 EE         JSR $EEB3
A C00A 88               DEY
A C00B D0 FA            BNE $C007
A C00D CA               DEX
A C00E D0 F2            BNE $C002
A C010 00               BRK
```

2.9.3 Effektiver Einsatz von Schleifen

Bei der Erläuterung der indizierten Adressierung versprach ich Ihnen, daß diese zusammen mit Programmschleifen bereits zu recht wirkungsvollen Ergebnissen führt.

Um diese Behauptung zu beweisen, greife ich auf eines unserer Ausgangsbeispiele zurück, das Füllen des Bildschirms mit dem Zeichen A.

Zu Beginn unseres Kurses war dies noch eine recht mühsame Angelegenheit und erforderte eine Unmenge einzelner STA-Befehle.

```
A C000 A9 01          LDA #$01
A C002 8D 00 04       STA $0400
A C005 8D 01 04       STA $0401
A C008 8D 02 04       STA $0402
...
...
```

Ein solches Programm wird bei Ihnen sicher keine Begeisterungsausbrüche auslösen. Eine umständlichere Art und Weise zum Beschreiben des Bildschirms ist kaum denkbar.

Das Problem kann jedoch unter Einsatz der indizierten Adressierung und einer Schleife weit einfacher gelöst werden.

```
Zeichenausgabe mit einer Schleife

A C000 A9 01          LDA #$01
A C002 A2 FF          LDX #$FF
A C004 9D 00 04       STA $0400,X
A C007 CA             DEX
A C008 D0 FA          BNE $C004
A C00A 00             BRK
```

Sie erkennen sicherlich die von $C004-$C008 reichende Schleife. Wie im letzten Kapitel wird die Schleife gebildet, indem das X-

Register mit der gewünschten Anzahl der Schleifendurchgänge geladen und in jedem Schleifendurchgang dekrementiert wird.

Die Schleife wird ausgeführt, solange der DEX-Befehl nicht zum Ergebnis null führt und dadurch das Zero-Flag gesetzt wird. Solange dieser Fall nicht eintritt, verzweigt der BNE-Befehl immer zum Schleifenanfang.

Der Befehl STA $0400,X arbeitet mit indizierter Adressierung. Die Endadresse ergibt sich aus der angegebenen Adresse plus dem aktuellen Inhalt des X-Registers. Beim ersten Schleifendurchgang besitzt X noch den Startwert $FF. Die Adresse ergibt sich somit als $04FF.

In diese aus angegebener Adresse plus X-Register resultierende Adresse wird der Inhalt des Akkumulators geschrieben. Da der Bildschirmspeicher an Adresse $0400 beginnt, wird in die 255te Bildschirmspeicherzelle der Bildschirmcode für A geschrieben.

Beim zweiten Durchgang ergibt sich als resultierende Adresse $04FE, da das X-Register um eins vermindert wurde und nun den Wert $FE besitzt ($0400 + $FE = $04FE).

Das Zeichen A wird somit in die 254te Speicherzelle des Video-RAMs übertragen. Sie erkennen das allgemeine Schema: nach jedem Schleifendurchlauf ist der Wert des X-Registers um eins niedriger als zuvor und daher auch die Endadresse ($0400 + X-Register).

Im letzten Schleifendurchgang enthält das X-Register den Wert eins. In Speicherzelle $0401 wird der Code für A geschrieben. Anschließend wird das X-Register dekrementiert und es folgt wieder der "bedingte Sprungbefehl" BNE, der in diesem Fall nicht ausgeführt wird, da das Ergebnis der letzten Operation (DEX) zu dem Ergebnis null führte.

Das Programm füllt somit der Reihe nach die Speicherzellen $04FF-$0401 mit dem Bildschirmcode des Zeichens A.

Das Programm entspricht bis in Details folgendem BASIC-Programm:

```
100 A=1:REM BILDSCHIRMCODE FUER A
110 X=255:REM ZAEHLER INITIALISIEREN
120 POKE 1024+I,A:REM A IN VIDEO-RAM
130 X=X-1:REM ZAEHLER DEKREMENTIEREN
140 IF X<>0 THEN GOTO 120:REM BEDINGTER SPRUNG
```

Vergleichen Sie Assembler- und BASIC-Programm und probieren Sie beide aus. Der Geschwindigkeitsunterschied ist geradezu unglaublich. Das Assembler-Programm ist übrigens nur elf Byte lang, im Gegensatz zu dem knapp 50 Byte langen BASIC-Programm, das zudem weiteren Speicherplatz für die verwendeten Variablen benötigt.

Da wir inzwischen bereits bei reichlich komplexen Programmen angelangt sind, stelle ich ein weiteres Demoprogramm vor. Wie wäre es denn zur Abwechslung (um die immer wiederkehrende sinnlose Ausgabe von A's zu vermeiden), wenn wir uns den kompletten Zeichensatz unseres Rechners ausgeben lassen?

Diese Ausgabe des kompletten Zeichensatzes ist mit einem winzigen Programm möglich, das nicht umfangreicher als das vorige ist.

Zeichensatz-Ausgabe

```
A C000 A2 FF        LDX #$FF
A C002 8A           TXA
A C003 9D 00 04     STA $0400,X
A C006 CA           DEX
A C007 D0 F9        BNE $C002
A C009 00           BRK
```

Das X-Register wird mit dem Wert $FF initialisiert. Im folgenden Schleifendurchgang wird dieser Wert in den Akkumulator und von dort in die Bildschirmzelle $04FF übertragen. Diese Adresse ergibt sich aus der bei der indizierten Adressierung angegebenen Ausgangsadresse plus dem Inhalt $FF des X-Regi-

sters. Das X-Register wird dekrementiert und - da das Ergebnis ungleich null ist (X=$FE) - der Branch-Befehl BNE ausgeführt.

Im folgenden Durchgang wird der Inhalt des X-Registers ($FE) wieder in den Akkumulator übertragen und in die Speicherzelle $04FE geschrieben.

Der Reihe nach werden somit die Speicherzellen $04FF-$0401 mit den Bildschirmcodes $FF-$01 gefüllt.

Die Schleife läuft wie üblich rückwärts, der Schleifenzähler wird nach jedem Durchgang um eins vermindert.

Das Zeichen mit dem Code $00 wird leider nicht mehr behandelt, da die Schleife verlassen wird, wenn die Dekrementierung des X-Registers zum Ergebnis null führt.

Mit dem bisher Erlernten können wir dieses Problem bereits lösen, indem wir nach dem Verlassen der Schleife das Zeichen mit dem Code $00 "per Hand" in die Speicherzelle $0400 schreiben und Sie den folgenden Befehl hinter dem Branch-Befehl einfügen:

```
...
...
BNE...
STX $0400
BRK
```

Im ersten Moment kommt Ihnen der Befehl STX $0400 wahrscheinlich sonderbar vor. Anbieten würde sich die Befehlsfolge:

```
LDA #$00
STA $0400
```

Bedenken Sie jedoch, daß die Schleife genau dann verlassen wird, wenn das X-Register den Wert null annimmt. In diesem

Moment wird der bedingte Sprung BNE nicht mehr ausgeführt und der erste Befehl nach der Schleife ausgeführt.

Wir wissen daher mit absoluter Sicherheit, daß das X-Register nach Verlassen der Schleife den Wert null enthält. Ein Laden des Akkumulators oder eines Indexregisters mit $00 ist daher überflüssig. Wir können die geschilderte Tatsache ausnutzen und den Inhalt des X-Registers direkt in die Speicherzelle $0400 schreiben.

Bevor ich diesen Schnellkurs der Schleifenprogrammierung abschließe, empfehle ich Ihnen dringend, das Erlernte anhand kleiner Programmieraufgaben zu vertiefen.

Um Ihre Phantasie anzuregen, schließe ich diesen Abschnitt mit ein Programm ab, das einen Ball über den Bildschirm bewegt. Das Programm arbeitet mit den erläuterten Verzögerungsschleifen. Die Geschwindigkeit der Bewegung können Sie beliebig variieren, indem Sie den Startwert des Schleifenzählers entsprechend ändern.

Versuchen Sie den Programmblauf bitte selbst zu durchschauen. Als Hilfestellung befinden sich Kommentare (die Sie bitte nicht abtippen!) hinter den einzelnen Befehlen.

Beachten Sie bitte, daß der Ball an einer bestimmten Bildschirmposition zuerst ausgegeben und nach Ablauf der Verzögerungsschleife wieder gelöscht - mit dem Leerzeichen überschrieben - wird (sonst ist der Bildschirm schnell mit Bällen gefüllt).

Starten Sie das Programm wie üblich mit G C000.

```
Ball bewegen

A C000 A2 FF    LDX #$FF    ;STARTWERT SCHLEIFENZAEHLER
A C002 A9 57    LDA #$57    ;BILDSCHIRMCODE FUER 'BALL'
A C004 9D 00 04 STA $0400,X ;'BALL' AN $0400
                            ;+X AUSGEBEN
A C007 A0 FF    LDY #$FF    ;STARTWERT VERZOEGERUNGSSCHLEIFE
```

```
A C009 20 B3 EE JSR $EEB3   ;AUFRUF VERZOEGERUNGS-
                            ;ROUTINE
A C00C 88       DEY         ;Y-REG.DEKREMENTIEREN
A C00D D0 FA    BNE $C009   ;Y=0? WENN NEIN, SPRUNG ZU $C009
A C00F A9 20    LDA #$20    ;BILDSCHIRMCODE F.LEERZEICHEN
A C011 9D 00 04 STA $0400,X ;BALL LOESCHEN AN $0400+X
A C014 CA       DEX         ;SCHLEIFENZAEHLER DEKREMENTIEREN
A C015 D0 EB    BNE $C002   ;X=0? WENN NEIN,
                            ;SPRUNG ZU $C002
A C017 00       BRK         ;UNTERBRECHUNG => MONITOR
```

2.10 Die wichtigsten Routinen des Betriebssystems

Das Betriebssystem des C64 enthält verschiedene Programme, die uns das "Programmiererleben" erheblich erleichtern. Alle Routinen sind Unterprogramme, vergleichbar mit einem BASIC-Unterprogramm.

BASIC-Unterprogramme werden mit GOSUB aufgerufen und mit RETURN beendet, wonach die Rückkehr zum aufrufenden Programm erfolgt. Assembler-Unterprogramme enden mit dem Befehl RTS ("return from subroutine", "kehre vom Unterprogramm zurück"), der dem BASIC-RETURN entspricht.

Aufgerufen werden diese Routinen mit dem Befehl JSR ("jump to subroutine", "verzweige zu einem Unterprogramm"), dem die Startadresse der Routine folgt.

Bei der Benutzung dieser Routinen ist die sogenannte "Parameterübergabe" wichtig. Den Routinen werden verschiedene Werte übergeben (meist im Akkumulator und den Indexregistern), die die genaue Funktionsweise bestimmen.

In der Praxis ist weiter zu beachten, welche Register und Speicherzellen durch das Unterprogramm verändert werden. Sonst passiert es uns, daß wir in einer Schleife das X-Register als Schleifenzähler verwenden, der Inhalt dieses Registers jedoch durch das in der Schleife aufgerufene Unterprogramm verändert wird.

Im Folgenden stelle ich jene Routinen vor, die am einfachsten zu benutzen sind und dennoch am häufigsten benötigt werden.

Außer der Funktion und der Startadresse wird beschrieben, welche Register die betreffenden Routinen beeinflussen.

Zeichenausgabe mit BSOUT

Mit der Routine BSOUT (Adresse $FFD2) kann ein beliebiges Zeichen auf dem Bildschirm ausgegeben werden. Der ASCII-Code (nicht der Bildschirmcode!) des gewünschten Zeichens wird der Routine im Akkumulator übergeben, bevor der Aufruf mit JSR $FFD2 erfolgt.

Das Zeichen wird an jener Bildschirmposition ausgegeben, an der sich momentan der Cursor befindet und dieser eine Spalte weiterbewegt (BSOUT arbeitet analog dem PRINT-Befehl in BASIC).

Zeichenausgabe mit BSOUT

```
A C000 A9 58          LDA #$58
A C002 20 D2 FF       JSR $FFD2
A C005 00             BRK
```

Nach dem Start wird an der aktuellen Cursorposition das Zeichen X (ASCII-Code $58 = dezimal 88) "geprintet), analog PRINT "X".

Wenn Sie also "G C000" schreiben und RETURN drücken, lautet die Zeile sofort danach "G C000 X". An der Stelle, an der das "X" aufgetaucht ist, war der CURSOR.

Ebenso wie der PRINT-Befehl kann auch BSOUT "Steuerzeichen" verwenden. Mit dem ASCII-Code 147 ($93) können Sie den Bildschirm löschen, ohne jede einzelne Speicherzelle des Video-RAMs mit einem Leerzeichen überschreiben zu müssen.

Ändern Sie bitte den Befehl LDA #$58 in LDA #$93. Auf die gleiche Weise können Sie beliebige weitere Steuerzeichen verwenden.

Befehl: BSOUT ($FFD2)

Funktion: Ausgabe des im Akku übergebenen Zeichens oder Steuercodes
Veränderte Register: keine
Beispiel (Bildschirm löschen und A ausgeben):

```
A C000 A9 93          LDA #$93
A C002 20 D2 FF       JSR $FFD2
A C005 A9 41          LDA #$41
A C007 20 D2 FF       JSR $FFD2
A C00A 00             BRK
```

Tastatur abfragen mit GETIN

Die Routine GETIN ($FFE4) liest ein Zeichen von der Tastatur ein und übergibt den ASCII-Code des Zeichens im Akkumulator. Wurde keine Taste gedrückt, wird der Code $00 übergeben.

GETIN benötigt keinerlei Parameter, beeinflußt aber alle Register!

Mit einer Kombination aus BSOUT und GETIN können wir eine kleine "Textverarbeitung" schreiben. Wir benötigen jedoch zwei bisher noch nicht erläuterte Befehle.

Der BEQ-Befehl

BEQ ist die "Umkehrung" von BNE. Auch dieser Befehl prüft, ob die jeweils letzte Operation zum Ergebnis null führte. Im Gegensatz zu BNE wird in diesem Fall jedoch zur angegeben Adresse verzweigt.

BEQ verzweigt, wenn das Zero-Flag gesetzt ist, also die letzte Operation zum Ergebnis null führte. Ergab sich ein Resultat ungleich null, wird der bedingte Sprung nicht ausgeführt. GETIN und BEQ gestatten somit den Aufbau von Warteschleifen, die erst dann verlassen werden, wenn eine beliebige Taste gedrückt wird.

```
Warteschleife
      A C000 20 E4 FF          JSR $FFE4
      A C003 F0 FB             BEQ $C000
      A C005 00                BRK
```

Der Aufruf von GETIN (JSR $FFE4) liest ein Zeichen von der Tastatur. Wurde keine Taste betätigt, lädt GETIN als letzte Operation den Akkumulator mit dem Wert null. Da die Bedingung "Verzweige, wenn gleich null" (BEQ) erfüllt ist, wird erneut GETIN aufgerufen.

Der JMP-Befehl

Außer den bedingten Sprüngen stellt uns der Rechner einen "unbedingten" Sprung zur Verfügung, der immer ausgeführt wird, analog dem BASIC-Befehl GOTO. Dem JMP-Befehl folgt die Angabe einer Adresse, zu der verzweigt wird.

```
Endlosschleife
      A C000 A9 41             LDA #$41
      A C002 20 D2 FF          JSR $FFD2
      A C005 4C 00 C0          JMP $C000
```

Im Beispiel wird mit BSOUT ein A ausgegeben, anschließend erfolgt ein unbedingter Sprung zum Programmanfang, der immer ausgeführt wird. Es handelt sich somit um eine "Endlosschleife", die niemals beendet wird.

Mit diesen beiden Befehlen können wir unsere "Textverarbeitung" schreiben.

```
Mini-Textverarbeitung
      A C000 20 E4 FF          JSR $FFE4
      A C003 F0 FB             BEQ $C000
      A C005 20 D2 FF          JSR $FFD2
      A C008 4C 00 C0          JMP $C000
```

JSR $FFE4 und BEQ $C000 bilden die erläuterte Eingabeschleife. Wurde eine Taste betätigt, wird das eingegebene Zeichen mit BSOUT auf dem Bildschirm ausgegeben (JSR $FFD2).

Der folgende JMP-Befehl bewirkt die Rückkehr zum Programmanfang. Das Programm läuft daher in einer Endlosschleife, die Zeichen für Zeichen von der Tastatur einliest und ausgibt.

Routine: GETIN ($FFE4)

Funktion: Liest ein Zeichen von der Tastatur und übergibt den ASCII-Code des Zeichens im Akkumulator übergeben ($00 = keine Taste gedrückt)
Veränderte Register: Akkumulator, X, Y
Beispiel (Auf Taste warten und ausgeben):

```
      A C000 20 E4 FF          JSR $FFE4
      A C003 F0 FB             BEQ $C000
      A C005 20 D2 FF          JSR $FFD2
```

Cursor setzen mit PLOT

Wie Sie wissen, ist es beim C64 ohne besonderen Aufwand in BASIC-Programmen nicht möglich, den Cursor auf eine bestimmte Bildschirmposition zu setzen, im Gegensatz zu Assembler-Programmen.

Die PLOT-Routine ($FFF0) setzt den Cursor auf eine beliebige Bildschirmposition. Die gewünschte Spalte (0-39) wird im Y-

Register und die gewünschte Zeile (0-24) im X-Register übergeben. Die PLOT-Routine beeinflußt ebenso wie GETIN den Inhalt aller Register.

Beachten Sie bitte, daß vor dem Aufruf dieser Routine der uns noch unbekannte Befehl CLC in das Programm einzufügen ist.

Routine: PLOT ($FFF0)

Funktion: Setzt den Cursor auf die im Y-Register übergebene Spalte und die im X-Register übergebene Zeile. Voraussetzung: Vor dem Aufruf der Routine muß der Befehl CLC stehen).
Veränderte Register: Akkumulator, X, Y
Beispiel (Zeichen "A" in Spalte 3 von Zeile 5 ausgeben):

```
A C000 A2 03        LDX #$03
A C002 A0 05        LDY #$05
A C004 18           CLC
A C005 20 F0 FF     JSR $FFF0
A C008 A9 41        LDA #$41
A C00A 20 D2 FF     JSR $FFD2
A C00D 00           BRK
```

Demoprogramme

Wir stehen vor dem Ende des ersten Hauptteils. Sie kennen nun die wichtigsten Grundlagen der Assembler-Programmierung und verschiedene Betriebssystem-Routinen, die uns die Programmierung sehr erleichtern.

Zur Übung folgen nun einige Demoprogramme, die den praktischen Umgang mit BSOUT, GETIN, PLOT und den bisher erläuterten Assembler-Befehlen aufzeigen.

Spiegelschrift

Das folgende Programm demonstriert den effektiven Umgang mit der indizierten Adressierung. Der Inhalt der obersten Bild-

schirmzeile wird in der untersten Zeile dann genau wieder in einer Art "Spiegelschrift" ausgegeben.

```
Spiegelschrift: Version 1
      A C000 A2 27          LDX #$27     ;STARTWERT: DEZIMAL 39
      A C002 A0 00          LDY #$00     ;STARTWERT: 0
      A C004 BD 00 04       LDA $0400,X  ;$0400 + X LESEN
      A C007 99 C0 07       STA $07C0,Y  ;UND NACH $07C0
                                         ;+Y SCHREIBEN
      A C00A C8             INY          ;Y INKREMENTIEREN
      A C00B CA             DEX          ;X DEKREMENTIEREN
      A C00C D0 F6          BNE $C004    ;X = 0 ?
                                         ;NEIN => VERZWEIGEN
      A C00E 00             BRK
```

Geben Sie die im Listing abgebildeten Kommentare bitte nicht mit ein! Der Ablauf: Nachdem das X-Register mit dem Startert $27 und das Y-Register mit $00 geladen wurden, beginnt die Programmschleife, in der Zeichen für Zeichen gelesen und in die unterste Zeile geschrieben wird.

Im ersten Durchgang wird der Akkumulator mit dem Inhalt der Speicherzelle $0427 ($0400 + X-Inhalt($27)) geladen. Diese Speicherzelle entspricht der letzten Spalte der obersten Bildschirmzeile.

Das Zeichen wird nun nach $07C0 ($07C0 + Y-Inhalt($00)) geschrieben. $07C0 ist die Adresse der ersten Spalte in der untersten Zeile. Das letzte Zeichen der obersten Zeile kommt somit an den Beginn der untersten Zeile.

Im nächsten Durchgang der Schleife soll das vorletzte Zeichen der ersten Zeile zum zweiten Zeichen der untersten Zeile werden. Entsprechend wird das X-Register dekrementiert und das Y-Register inkrementiert (=> 2.Durchgang: Lesen von $0400 + $26); Schreiben nach $07C0 + $01).

Der Ablauf kann folgendermaßen beschrieben werden: Das Programm "tastet" sich in der obersten Zeile vorwärts und in der untersten rückwärts. Daher werden die Zeichen in - gegenüber

dem Original in der ersten Zeile - in umgekehrter Reihenfolge geschrieben, also in "Spiegelschrift".

Ein "Schönheitsfehler" tritt jedoch auf. Die Schleife überträgt nicht 40, sondern nur 39 Zeichen. Das erste Zeichen der obersten Zeile (an Adresse $0400 = $0400 + $00) wird nicht mehr behandelt, da die Schleife verlassen wird, wenn das X-Register den Inhalt $00 besitzt (Verzweigung mit BNE ("verzweige, wenn ungleich null)).

Die Korrektur ist jedoch sehr einfach. Wir ändern $0400 in $03FF. Beim letzten Schleifendurchgang besitzt X den Inhalt $01 und es wird wie gewünscht auf Adresse $0400 zugegriffen ($0400 = $03FF + $01).

Durch die Korrektur wird jedoch eine weitere Korrektur erforderlich. Beim ersten Durchgang wird nicht mehr auf das letzte Zeichen der ersten Zeile zugegriffen (an Adresse $0427), sondern auf das vorletzte, da $03FF + $27 die Adresse $0426 ergibt. Das Problem wird gelöst, indem das X-Register vor Eintritt in die Schleife statt $27 den geänderten Ausgangswert $28 erhält.

```
Spiegelschrift: Version 2
      A C000 A2 28            LDX #$28     ;STARTWERT: DEZIMAL 40
      A C002 A0 00            LDY #$00     ;STARTWERT: 0
      A C004 BD FF 03         LDA $03FF,X  ;$03FF + X LESEN
      A C007 99 C0 07         STA $07C0,Y  ;UND NACH $07C0
                                           ;+Y SCHREIBEN
      A C00A C8               INY          ;Y INKREMENTIEREN
      A C00B CA               DEX          ;X DEKREMENTIEREN
      A C00C D0 F6            BNE $C004    ;X = 0 ?
                                           ;NEIN => VERZWEIGEN
      A C00E 00               BRK
```

Gehen Sie bei der "Erprobung" des Programms so vor:

1. Löschen Sie den Bildschirm und schreiben Sie einen beliebigen Text in die erste Bildschirmzeile (bleiben Sie dabei im Monitor; Sie müssen nicht mit X nach BASIC zurückkehren).

2. Bewegen Sie den Cursor in die zweite oder dritte Zeile und rufen Sie unsere Routine mit G C000 auf.

Rahmen zeichnen

Das folgende Programm demonstriert die Anwendung der PLOT- und der BSOUT-Routine. Um den Bildschirm wird ein Rahmen gelegt, der in der ersten Zeile (Zeile Nr.0) beginnt und in der elften Zeile (Zeile Nr.10) endet. Der Rahmen endet vor der jeweils letzten Spalte einer Zeile. Der Grund: Sie kennen bestimmt das Phänomen, das Leerzeilen eingefügt werden, wenn man den rechten Bildschirmrand (Spalte 39) beschreibt. Dieses Einfügen wollen wir jedoch vermeiden.

Die Vorgehensweise:

1. In der obersten Bildschirmzeile wird - in einer Schleife - eine Linie gezeichnet, die aus dem Graphikzeichen mit dem ASCII-Code 195 (= $C3) besteht. Vor Beginn der Schleife wird die obere linke Rahmenecke gezeichnet (in Spalte 0), nach der Schleife in Spalte Nummer 38 mit dem entsprechenden Graphikzeichen die obere rechte Ecke gemalt.
2. Die gleiche Linie wird nun in Zeile Nummer 10 gezeichnet, wobei für die untere rechte beziehungsweise linke Ecke entsprechende Graphikzeichen verwendet werden.
3. Ebenfalls in einer Schleife wird die jeweils erste und vorletzte Spalte (vorletzte, um das erwähnte Einfügen von Leerzeilen zu vermeiden) aller zwischen den beiden Linien liegenden Zeilen mit einer senkrechten Linie (ASCII-Code 194 = $C2) beschrieben.

Da dieses Programm doch recht umfangreich ist, wird es "abschnittsweise" erläutert. Am Ende des Kapitels finden Sie das komplette Programmlisting.

```
Rahmen zeichnen

Teil 1: Obere Linie zeichnen
      A C000 A0 00          LDY #$00     ;SPALTE 0
      A C002 A2 00          LDX #$00     ;ZEILE 0
      A C004 18             CLC          ;'PLOT' VORBEREITEN
      A C005 20 F0 FF       JSR $FFF0    ;CURSOR MIT PLOT SETZEN
      A C008 A9 B0          LDA #$B0     ;$B0 = GRAPHIKZ.OBERE
                                         ;LINKE ECKE
      A C00A 20 D2 FF       JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
      A C00D A2 25          LDX #$25     ;X MIT DEZIMAL 37 LADEN
      A C00F A9 C3          LDA #$C3     ;$C3 = GRAPHIKZEICHEN
                                         ;LINIE WAAGR.
      A C011 20 D2 FF       JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
      A C014 CA             DEX          ;SCHLEIFENZÄHLER
                                         ;VERMINDERN
      A C015 D0 FA          BNE $C011    ;LINIEN KOMPLETT
                                         ;AUSGEGEBEN?
      A C017 A9 AE          LDA #$AE     ;JA, DANN GRAPHIKZ.
                                         ;FÜR OBERE
      A C019 20 D2 FF       JSR $FFD2    ;RECHTE ECKE AUSGEBEN
```

Die - nicht einzugebenden - Kommentare zu diesem Teil erklären bereits die wesentlichen Besonderheiten. Zuerst wird der Cursor mit der PLOT-Routine auf die erste Spalte der obersten Zeile gesetzt. Mit BSOUT wird anschließend ein Graphikzeichen ausgegeben, ein "Winkel", der die obere linke Ecke des Rahmens darstellen soll.

Nun folgt eine Schleife, in der 37mal das Graphikzeichen "waagrechte Linie" (ASCII-Code 195 = $C3) ausgegeben wird. Die oberste Zeile ist nun bis auf die vorletzte Spalte komplett behandelt. In dieser wird ebenfalls ein Winkelzeichen ausgegeben, das die obere rechte Ecke des Rahmens darstellt (ASCII-Code 174 = $AE).

```
Teil 2: Untere Linie zeichnen
      A C01C A0 00          LDY #$00     ;SPALTE 0
      A C01E A2 0A          LDX #$0A     ;ZEILE 10
      A C020 18             CLC          ;'PLOT' VORBEREITEN
```

```
A C021 20 F0 FF          JSR $FFF0    ;CURSOR MIT PLOT SETZEN
A C024 A9 AD             LDA #$AD     ;$B0 = GRAPHIKZ.UNTERE
                                      ;LINKE ECKE
A C026 20 D2 FF          JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
A C029 A2 25             LDX #$25     ;X MIT DEZIMAL 37 LADEN
A C02B A9 C3             LDA #$C3     ;$C3 = GRAPHIKZEICHEN
                                      ;LINIE WAAGR.
A C02D 20 D2 FF          JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
A C030 CA                DEX          ;SCHLEIFENZÄHLER
                                      ;VERMINDERN
A C031 D0 FA             BNE $C02D    ;LINIEN KOMPLETT
                                      ;AUSGEGEBEN?
A C033 A9 BD             LDA #$BD     ;JA, DANN GRAPHIKZ.
                                      ;FÜR UNTERE
A C035 20 D2 FF          JSR $FFD2    ;RECHTE ECKE AUSGEBEN
```

Wie unschwer zu erkennen ist, ist Teil 2 - das Zeichnen der unteren Linie des Rahmens in Zeile 10 - mit Teil 1 nahezu identisch. Der einzige Unterschied besteht in der Verwendung anderer Graphikzeichen für die Eckwinkel.

Davon abgesehen ist der Ablauf unverändert: Der Cursor wird auf Spalte 0 von Zeile 10 positioniert, die linke Ecke gezeichnet, anschließend in einer Schleife 37mal eine waagrechte Linie und zuletzt der rechte Eckwinkel ausgegeben.

```
Teil 3 : Senkrechte Linien zeichnen
      A C038 A2 09          LDX #$09     ;SCHLEIFENZÄHLER LADEN
      A C03A A0 00          LDY #$00     ;SPALTE = 0
      A C03C 18             CLC          ;'PLOT' VORBEREITEN
      A C03D 20 F0 FF       JSR $FFF0    ;CURSOR AUF SPALTE 0
                                         ;VON ZEILE X
      A C040 A9 C2          LDA #$C2     ;GRAPHIKZ.SENKRECHTE LINIE
      A C042 20 D2 FF       JSR $FFD2    ;AUSGEBEN
      A C045 A0 26          LDY #$26     ;SPALTE = $26 = DEZIMAL 38
      A C047 18             CLC          ;'PLOT' VORBEREITEN
      A C048 20 F0 FF       JSR $FFF0    ;CURSOR AUF SPALTE 38
                                         ;VON ZEILE X
      A C04B A9 C2          LDA #$C2     ;GRAPHIKZ.SENKRECHTE LINIE
      A C04D 20 D2 FF       JSR $FFD2    ;AUSGEBEN
```

```
A C050 CA          DEX         ;ZEILENZÄHLER VERMINDERN
A C051 D0 E7       BNE $C03A   ;FERTIG? NEIN =>
                               ;VERZWEIGEN
A C053 60          RTS         ;SONST ZURÜCK NACH BASIC
```

Der dritte und letzte Teil unterscheidet sich sehr von den beiden vorangegangen. Innerhalb der Schleife wird der Cursor zweimal gesetzt. Das X-Register dient zugleich als Schleifen- und als Zeilenzähler und erhält zu Beginn den Startwert $09.

Der Cursor wird im ersten Durchgang auf Spalte 0 von Zeile X (also Zeile 9) positioniert und eine senkrechte Linie ausgegeben (ASCII-Code 194 = $C2).

Anschließend wird er auf die Spalte $26 (dezimal 38) der gleichen Zeile gesetzt und dort ebenfalls einen senkrechte Linie ausgegeben.

Zeile 10 enthält nun die senkrechten Begrenzungen des Rahmens, entsprechende Graphikzeichen in der ersten und der vorletzten Spalte. Nun wird die nächsthöhere Zeile behandelt.

Unser Zeilenzähler, das X-Register, wird dekrementiert und ein neuer Schleifendurchgang beginnt, wenn die Bedingung BNE ("verzweige, wenn ungleich null") erfüllt ist, das heißt wenn X nach der Dekrementierung einen beliebigen Wert außer null besitzt.

Auf diese Weise werden alle Zeilen ab Zeile 9 aufwärts behandelt. Wenn der Rahmen komplett ist, wird die Schleife verlassen und ausnahmsweise nicht mit BRK in den Monitor zurückgekehrt.

Diese Routine dürfte in vielen BASIC-Programmen zu gebrauchen sein. Daher endet Sie mit dem Befehl RTS. RTS führt zur Rückkehr in ein BASIC-Programm, wenn dieses die Routine mit SYS 49152 aufrief.

Ein entsprechendes Demoprogramm:

```
100 SYS 49152
110 FOR I=1 TO 1000:NEXT
120 PRINT CHR$(147)
130 SYS 49152
```

Dieses - nicht unbedingt sehr sinnvolle - Demoprogramm ruft die Routine auf, um einen Rahmen zu zeichnen. Nach Ablauf einer Verzögerungsschleife wird der Bildschirm gelöscht und erneut ein Rahmen gezeichnet.

Das komplette Programmlisting:

```
Teil 1: Obere Linie zeichnen
      A C000 A0 00            LDY #$00     ;SPALTE 0
      A C002 A2 00            LDX #$00     ;ZEILE 0
      A C004 18               CLC          ;'PLOT' VORBEREITEN
      A C005 20 F0 FF         JSR $FFF0    ;CURSOR MIT PLOT SETZEN
      A C008 A9 B0            LDA #$B0     ;$B0 = GRAPHIKZ.OBERE
                                           ;LINKE ECKE
      A C00A 20 D2 FF         JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
      A C00D A2 25            LDX #$25     ;X MIT DEZIMAL 37 LADEN
      A C00F A9 C3            LDA #$C3     ;$C3 = GRAPHIKZEICHEN
                                           ;LINIE WAAGR.
      A C011 20 D2 FF         JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
      A C014 CA               DEX          ;SCHLEIFENZÄHLER
                                           ;VERMINDERN
      A C015 D0 FA            BNE $C011    ;LINIEN KOMPLETT
                                           ;AUSGEGEBEN?
      A C017 A9 AE            LDA #$AE     ;JA, DANN GRAPHIKZ.
                                           ;FÜR OBERE
      A C019 20 D2 FF         JSR $FFD2    ;RECHTE ECKE AUSGEBEN
Teil 2: Untere Linie zeichnen
      A C01C A0 00            LDY #$00     ;SPALTE 0
      A C01E A2 0A            LDX #$0A     ;ZEILE 10
      A C020 18               CLC          ;'PLOT' VORBEREITEN
      A C021 20 F0 FF         JSR $FFF0    ;CURSOR MIT PLOT SETZEN
      A C024 A9 AD            LDA #$AD     ;$B0 = GRAPHIKZ.UNTERE
                                           ;LINKE ECKE
      A C026 20 D2 FF         JSR $FFD2    ;MIT 'BSOUT' AUSGEBEN
      A C029 A2 25            LDX #$25     ;X MIT DEZIMAL 37 LADEN
```

```
A C02B A9 C3            LDA #$C3    ;$C3 = GRAPHIKZEICHEN
                                    ;LINIE WAAGR.
A C02D 20 D2 FF         JSR $FFD2   ;MIT 'BSOUT' AUSGEBEN
A C030 CA               DEX         ;SCHLEIFENZÄHLER
                                    ;VERMINDERN
A C031 D0 FA            BNE $C02D   ;LINIEN KOMPLETT
                                    ;AUSGEGEBEN?
A C033 A9 BD            LDA #$BD    ;JA, DANN GRAPHIKZ.
                                    ;FÜR UNTERE
A C035 20 D2 FF         JSR $FFD2   ;RECHTE ECKE AUSGEBEN
Teil 3 : Senkrechte Linien zeichnen
A C038 A2 09            LDX #$09    ;SCHLEIFENZÄHLER LADEN
A C03A A0 00            LDY #$00    ;SPALTE = 0
A C03C 18               CLC         ;'PLOT' VORBEREITEN
A C03D 20 F0 FF         JSR $FFF0   ;CURSOR AUF SPALTE 0
                                    ;VON ZEILE X
A C040 A9 C2            LDA #$C2    ;GRAPHIKZ.SENKRECHTE LINIE
A C042 20 D2 FF         JSR $FFD2   ;AUSGEBEN
A C045 A0 26            LDY #$26    ;SPALTE = $26 = DEZIMAL 38
A C047 18               CLC         ;'PLOT' VORBEREITEN
A C048 20 F0 FF         JSR $FFF0   ;CURSOR AUF SPALTE 38
                                    ;VON ZEILE X
A C04B A9 C2            LDA #$C2    ;GRAPHIKZ.SENKRECHTE LINIE
A C04D 20 D2 FF         JSR $FFD2   ;AUSGEBEN
A C050 CA               DEX         ;ZEILENZÄHLER VERMINDERN
A C051 D0 E7            BNE $C03A   ;FERTIG? NEIN =>
                                    ;VERZWEIGEN
A C053 60               RTS         ;SONST ZURÜCK NACH BASIC
```

Geben Sie die Kommentare bitte nicht ein, sondern wie üblich nur die Assembler-Befehle selbst.

Teil 2: Fortgeschrittene Assembler-Programmierung

Sie kennen nun alle Grundlagen der Assembler-Programmierung und sollten sich erst einmal mit der selbständigen Erstellung weiterer Übungsprogramme beschäftigen. Gerade für die relativen komplexen Themen "Schleifenbildung" und "indirekt-indizierte Adressierung" gilt, daß diese nur durch Übung wirklich zu

verstehen sind. Experimentieren Sie also und lassen Sie sich durch gelegentliche "Abstürze" nicht irritieren. Zum Thema "Abstürze": bei der Assembler-Programmierung ist ein sogenannter "RESET-Schalter" eine absolute Notwendigkeit.

Sie können jederzeit einen RESET-Schalter erwerben, der auf den USER-Port oder einen Floppy-Anschluß aufgesteckt wird.

Wie Sie sicher feststellten, ist es bei der Assembler-Programmierung öfters der Fall, daß sich der Rechner "aufhängt" und nicht zur Weiterarbeit zu bewegen ist.

Vor allem bei der in diesem Aufbaukurs besprochenen "Interrupt-Programmierung" wird dies öfters geschehen.

Mit einem RESET-Schalter ist der Rechner wieder "zum Leben zu erwecken", wobei im Gegensatz zum Ausschalten Assembler-Programme meist unverändert erhalten bleiben.

2.11 Schleifen "unter die Lupe genommen"

Die vorangegangenen Kapitel setzten Sie bereits in die Lage, kleine Assembler-Programme zu schreiben. Sie kennen die Transfer-Befehle, die In-/Dekrementierbefehle, die "Branch"-Befehle BEQ und BNE und, die Befehle JSR und JMP, und die wichtigsten Adressierungsarten.

Da Sie inzwischen bestimmt "auf den Geschmack" gekommen sind und eigene Übungsprogramme schrieben, besitzen Sie inzwischen handfeste Assembler-Grundkenntnisse. Auf den folgenden Seiten werden wir daher mit der Verwirklichung anspruchsvollerer Programmprojekte beginnen, für die wir jedoch weitere Befehle benötigen.

Da diese Projekte höhere Anforderungen an Sie stellen werden, hole ich zuerst einige Versäumnisse nach und nehme Befehle, die bisher nur oberflächlich besprochen wurden, genauer "unter die Lupe". Auf den folgenden Seiten wird vorwiegend die Schleifenbildung behandelt, wobei außer den bereits bekannten

Befehlen BEQ und BNE auch noch weitere nützliche Befehle zur Schleifenbildung eingeführt werden.

Näheres zu den In- und Dekrementierbefehlen

Angenommen, eine Speicherzelle oder ein Register besitzt bereits den Maximalwert $FF und wird nun inkrementiert. Oder aber, umgekehrt, ein Register hat den Inhalt $00 und wird nun dekrementiert.

Wissen Sie, was in einem solchen Fall passiert? Wenn nicht, führen Sie mit mir einige kleine Experimente durch.

```
A C000 A2 FF            LDX #$FF
A C002 E8               INX
A C003 00               BRK
```

Wir laden das X-Register mit $FF und inkrementieren es anschließend. Die Registeranzeige des Monitors sollte anschließend etwa so aussehen:

```
.....AC XR YR .....
.....00.....
```

Das X-Register enthält den Wert $00. Eines der angesprochenen Geheimnisse wäre damit gelöst. Wenn eine Speicherzelle inkrementiert wird, die bereits den höchsten Wert $FF enthält, erhält Sie dadurch den niedrigsten Wert $00.

```
A C000 A2 00            LDX #$00
A C002 CA               DEX
A C003 00               BRK
```

Das zweite Geheimnis deckt dieses Programm auf. Die Registeranzeige:

```
.....AC XR YR .....
.....FF.....
```

Somit steht ebenfalls fest, daß eine Speicherzelle, die den Inhalt $00 besitzt und nun dekrementiert wird, den Wert $FF erhält. Immer dann, wenn ein Ende der Werteskala erreicht ist, der größte oder aber der kleinste mögliche Wert, beginnt die Zählerei wieder beim entgegengesetzten Ende.

Diese Feststellung bezieht sich keineswegs ausschließlich auf das X-Register, sondern auf alle Speicherzellen und Register!

Ein weiteres Mal: BNE und BEQ

Wie die vorhergehenden Kapitel zeigten, setzen uns erst Schleifen in die Lage, sinnvolle Assembler-Programme zu schreiben. Sie kennen nun bereits zwei Schleifenbefehle, die bedingten Sprungbefehle BNE ("branch if not equal zero", "verzweige, wenn ungleich null") und BEQ ("branch if equal zero", "verzweige, wenn gleich null"), wobei sich der Ausdruck "gleich null" beziehungsweise "ungleich null" immer auf das Ergebnis der letzten Operation bezieht, die das Zero-Flag beeinflußte (zum Beispiel Transferbefehle oder In-/Dekrementierbefehle).

Um die Schleifenprogrammierung auf die vielfältigsten Probleme anwenden zu können, benötigen wir weitere Befehle. Der Standardbefehl BNE besitzt einen gravierenden Nachteil. Wenn der Schleifenzähler bis zum Wert null dekrementiert ist, wird die Schleife verlassen. Daher war es uns zuvor nicht möglich, bei der Ausgabe des Zeichensatzes auch das Zeichen mit dem Code $00 - den "Klammeraffen" - auszugeben.

Rufen wir uns kurz in Erinnerung, wie die Befehle BEQ und BNE ablaufen. Die Transfer-Befehle (STA, STX, TAX, TYA...) und die In-/Dekrementierbefehle (INC, DEX...) beeinflussen ein Bit des Status-Registers, das sogenannte "Zero-Flag". Führte einer der genannten Befehle zum Ergebnis null (wurde zum Beispiel der Akku mit dem Wert $00 geladen oder eine Speicherzelle dekrementiert, die zuvor den Inhalt eins besaß), wird automatisch das Zero-Flag gesetzt, ansonsten gelöscht. Die folgende Skizze veranschaulicht den Aufbau des Status-Registers,

wobei uns von den verschiedenen Flags jedoch vorläufig nur das Zero-Flag interessieren soll.

```
Die Bits (=Flags) des Status-Registers

  7      6          5        4      3         2        1     0
Negativ Überlauf (unbenutzt) Break Dezimal Interrupt Zero Carry
```

BEQ und BNE prüfen den Zustand dieses Flags (gesetzt oder gelöscht), um festzustellen, ob die letzte Operation, die das Flag beeinflußte, zum Ergebnis null führte. Wenn ja, verzweigt BEQ zur angegebenen Adresse, ansonsten wird der folgende Programmbefehl bearbeitet.

Wie wir wissen, arbeitet BNE genau umgekehrt. Solange die letzte Operation nicht den Wert null ergab, wird zur angegebenen Adresse verzweigt, ansonsten wird das Programm mit dem nächsten Befehl fortgesetzt.

Diese Erkenntnisse sollen anhand mehrerer Demoprogramme vertieft werden.

```
A C000 A9 00          LDA #$00
A C002 00             BRK
```

Nach dem Aufruf mit G C000 betrachten Sie bitte die Registeranzeige des Monitors, genauer: den Inhalt des Status-Registers (SR). Wie üblich wird eine zweistellige Hexadezimalzahl ausgegeben.

Wenn Sie sich die Mühe machen und die ausgegebene Hexadezimalzahl in eine Dualzal umzuwandeln, werden Sie feststellen, daß das Bit 1 (Stellenwert von Bit 1: $02) dieser Zahl gesetzt ist, das Zero-Flag.

Die einfachste Methode zur Umwandlung einer Hexadezimalzahl in eine Dualzahl verläuft nach folgendem Schema:

1. Gegeben sei eine zweistellige Hexadezimalzahl, zum Beispiel $7F.

2. Wandeln Sie die rechte Ziffer in eine vierstellige Dualzahl um. $F wird zu %1111.
3. Wandeln Sie die linke Stelle entsprechend um. $7 wird zu %0111. Schreiben Sie diese Dualzahl links neben die zuerst ermittelte Dualzahl.
4. Als Ergebnis erhalten Sie im Beispiel $7F = %01111111.

Ein weiteres Beispiel: $3A = $3 und $A = %0011 und %1010 = %00111010. Um zu erkennen, ob das Zero-Flag des Status-Registers - also Bit 1 - gesetzt oder gelöscht ist, reicht es sogar aus, die rechte Ziffer der Hexadezimalzahl - die den Bits 0-3 entspricht - in eine vierstellige Dualzahl umzuwandeln.

In dieser rechten Ziffer ist nach Ablauf des Programms das Bit 1 - das Zero-Flag - tatsächlich gesetzt, da die "Operation" LDA #$00 eine null als Ergebnis hat. Die Gegenprobe:

```
A C000 A9 01          LDA #$01
A C000 00             BRK
```

Die dem BRK-Befehl folgende Registeranzeige gibt das Status-Register wiederum als zweistellige "Hex-Zahl" aus. Das Bit 1 dieser Zahl ist diesmal jedoch gelöscht, wie erwartet, da der Akkumulator mit einem Wert ungleich null geladen wurde.

Schleifen mit BPL und BMI

Bevor ich weitere Schleifenbefehle einführe, zeige ich Ihnen ein kleines Phänomen. Geben Sie bitte ein:

```
A C000 A2 7A          LDX #$7A
A C002 E8             INX
A C003 00             BRK
```

Rufen Sie das Programm wieder mit G C000 auf. Schauen Sie sich nun die Registeranzeige des Monitors an:

```
.....SR AC XR.....
.....3?....7B.....
```

Uns interessieren im Moment nur die Anzeige des X- und des Status-Registers. Das X-Register enthält natürlich den Wert $7B, da es mit dem Wert $7A geladen und anschließend durch INX um eins erhöht wurde.

Vom Status-Register interessiert uns nur Bit Nummer 7. Bit 7 ist das Bit ganz links in einer achtstelligen Dualzahl und somit in der linken Ziffer der Hex-Zahl enthalten. Für diese Ziffer ergibt sich nach dem gezeigten Schema %0011 (wenn $30 ausgegeben wurde).

Das Bit ganz links (Bit 7) ist somit gelöscht. Dieses Bit nennt man "Negativ-Flag" oder N-Flag. Das N-Flag besitzt ebenso wie das Zero-Flag eine spezielle Bedeutung, die wir nun erforschen wollen.

Versuchen Sie nun folgendes: starten Sie das Programm viermal nacheinander mit G C002, daß heißt ab dem Befehl INX, und betrachten Sie sich jeweils die Registeranzeige (X-Register und Statusregister).

Der Inhalt des X-Registers wird sich nach jedem Aufruf um eins erhöhen ($7C, $7D, $7E, $7F), während sich das Status-Register nicht verändert.

Nun passen Sie gut auf. Rufen Sie das Programm ein weiteres Mal mit G C002 auf. Das X-Register enthält nun den Wert $80. Soweit ist alles in Ordnung. Unerklärlich ist jedoch, warum sich auf einmal das Status-Register verändert, um genau zu sein, nur die linke Ziffer ändert sich.

Anstatt $30 wird auf einmal $B0 ausgegeben. Wenn wir diese Zahl "bitweise" untersuchen, stellen wir fest, daß Bit 7, das N-Flag, auf einmal gesetzt ist, während es zuvor die ganze Zeit gelöscht war.

Je nach verwendetem Monitor kann die Anzeige des Status-Registers variieren. Das Negativ-Flag (Bit 7) befindet jedoch auf alle Fälle im beschriebenen "Zustand".

Wenn Sie wollen, können Sie das Programm solange aufrufen, bis das X-Register den Wert $FF enthält und beim nächsten Aufruf der "Überschlag" nach $00 erfolgt. Sie werden feststellen, daß in diesem Moment das N-Flag wieder gelöscht wird.

Sie können natürlich auch LDX #$FF eingeben und das Programm dann nur einmal starten.

Fassen wir unsere "Untersuchungsergebnisse" zusammen: immer dann, wenn der Befehl INX zu einem Resultat führt, das kleiner ist als $80 (=dezimal 128), wird das N-Flag gelöscht.

Führt INX zu einem Ergebnis von 128 oder größer, wird das Negativ-Flag gesetzt.

Ohne weitere Untersuchung will ich Ihnen verraten, daß nicht nur der INX-Befehl, sondern auch DEX, INY, DEY, INC, DEC das N-Flag auf die gleiche Weise beeinflussen. Außer diesen In-/Dekrementier-Befehlen wird das N-Flag unter anderem auch von allen besprochenen "Ladebefehlen" beeinflußt, also von LDA, LDX und LDY.

Diesen Effekt können wir für den "Schleifenbau" ausnutzen. Es existieren zwei relative Sprungbefehle, die den Zustand des N-Flags "testen" und je nach Ergebnis zur angegebenen Adresse verzweigen oder nicht verzweigen.

BPL ("branch if plus", "verzweige, wenn positiv") verzweigt zur angegebenen Adresse, wenn das Negativ-Flag nicht gesetzt ist. BMI ("branch if minus", "verzweige, wenn negativ") verzweigt im entgegengesetzten Fall, wenn das N-Flag gesetzt ist.

```
A C000 A2 10          LDX #$10
A C002 CA             DEX
```

```
A C003 10 FD          BPL $C002
A C005 00             BRK
```

Wenn Sie dieses Programm aufrufen, werden Sie anschließend feststellen, daß das X-Register den Wert $FF enthält. Diese Schleife wurde somit auch dann noch einmal durchlaufen, als das X-Register bereits den Wert $00 enthielt. Erst bei der folgenden Dekrementierung, die den Wert $FF ergab, wurde die Schleife verlassen.

Warum? Nun, solange X von $10 bis $00 heruntergezählt wurde, war das N-Flag gelöscht und die Bedingung "verzweige, wenn positiv" erfüllt, der bedingte Sprung wurde ausgeführt.

In dem Moment, in dem das X-Register von $00 auf $FF dekrementiert wurde, der DEX-Befehl somit zu einem Ergebnis größer als $79 führte, wurde das Negativ-Flag gesetzt und die Bedingung "verzweige, wenn positiv" war nicht länger erfüllt. Die folgende Tabelle gibt diese Zusammenhänge wieder:

Verzweigung	letzte Operation	Zero-Flag	Negativ-Flag
BEQ	gleich null	1	
BNE	ungleich null	0	
BMI	größer $79		1
BPL	kleiner $80		0

Ein Beispiel zum Lesen der Tabelle: BEQ verzweigt, wenn das Ergebnis der letzten Operation den Wert null ergab (Zero-Flag gesetzt).

Die Unterschiede zwischen BNE und BPL bei der Schleifenbildung demonstriert folgendes Programm:

```
Vergleich: BNE und BPL
      A C000 A2 80          LDX #$20
      A C002 8A             TXA
      A C003 9D 00 04       STA $0400,X
      A C006 CA             DEX
```

```
A C007 10 F9        BPL $C002
A C009 00           BRK
```

Das Programm ist altbekannt. Es gibt den Zeichensatz Ihres Rechners auf dem Bildschirm aus. Diesmal wird jedoch der Branch-Befehl BNE durch BPL ersetzt. Den Unterschied sehen Sie nach dem Aufruf mit G C000. Im Gegensatz zur alten Version wird auch das Zeichen mit dem Code $00 ausgegeben, der Klammeraffe.

Mit BPL gebildete Schleifen werden wie erläutert auch dann noch einmal durchlaufen, wenn das Zählregister den Wert $00 besitzt. Der Nachteil des Programms: da das X-Register nicht mit einem Wert geladen werden darf, der größer als $80 ist, wird nur die erste Hälfte des Zeichensatzes ausgegeben, die Zeichen mit den Codes $81-$FF fehlen.

Zusammenfassung

1. Zur Schleifenbildung werden vorwiegend die "Branch"-Befehle BEQ ("verzweige, wenn gleich null") und BNE ("verzweige, wenn ungleich null") beziehungsweise BPL ("verzweige, wenn positiv") und BMI ("verzweige, wenn negativ") eingesetzt. Ist die betreffende Bedingung erfüllt, wird zur angegebenen Adresse verzweigt, ansonsten das Programm mit dem folgenden Befehl fortgesetzt (Vergleiche: IF ... THEN GOTO ...).
2. BEQ und BNE testen das Zero-Flag und stellen am Zustand dieses Flags (gesetzt/gelöscht) fest, ob die letzte Operation zum Ergebnis null führte oder nicht (Ergebnis null: Zero-Flag gesetzt; Ergebnis ungleich null: Zero-Flag gelöscht).
3. BPL und BMI testen das Negativ-Flag und stellen am Flag-Zustand fest, ob die letzte Operation zu einem "positiven" oder aber zu einem "negativen" Ergebnis führte. "Positiv" und "Negativ" kennzeichnen hierbei die Größe des Ergebnisses. Bleibt das Ergebnis unter $80, so wird es als "positiv" (Negativ-Flag gelöscht), darüber aber als "negativ"

(Negativ-Flag gesetzt) bezeichnet. Ein Ergebnis wird daher als negativ bezeichnet, wenn Bit 7 (Wert $80) gesetzt ist.

4. Zum Einsatz von BNE und BPL in Schleifen: mit BNE wird eine Schleife verlassen, wenn der Schleifenzähler dekrementiert wird und anschließend den Wert $00 enthält (das Zero-Flag gesetzt wird). Mit BPL wird eine Schleife erst dann verlassen, wenn die Dekrementierung zum Überschlag von $00 zu $FF führt, das Ergebnis also "negativ" wird. Die BPL-Schleife besitzt daher genau einen Durchgang mehr als die BNE-Schleife.

5. Aus 4. geht hervor, daß das Zählregister bei der BNE-Schleife im letzten Durchgang den Wert $01 besitzt, bei der BPL-Schleife dagegen den Wert $00.

6. Mit BPL gebildete Schleifen können maximal 129mal durchlaufen werden, nämlich dann, wenn das Zählregister mit $80 "initialisiert" wird. Wird es mit einem größeren Wert geladen, passiert folgendes: das X-Register enthält zum Beispiel den Wert $90. Der erste DEX-Befehl führt nun zum Ergebnis $8F. Da dieses Ergebnis größer ist als 127, wird das Negativ-Flag gesetzt, um ein negatives Ergebnis anzuzeigen, und die Bedingung BPL ("verzweige, wenn positiv") ist nicht erfüllt!

7. BNE, BEQ, BPL und BMI können natürlich nur dann zum Testen und eventuellen Verzweigen eingesetzt werden, wenn die interessierende Operation (in Schleifen meist DEX, DEY, INX und INY) auch tatsächlich den Flag-Zustand je nach Ergebnis verändert. Folgende Befehlsklassen, die wir kennen, beeinflussen Zero- und Negativ-Flag: alle In-/Dekrementier-Befehle (DEC, INX...), alle "Lade-Befehle" (LDA, LDY...), alle "Register-Transfer-Befehle" (TAX, TYA...). Nach jedem dieser Befehle können die erläuterten Branch-Befehle für bedingte Sprünge eingesetzt werden.

2.12 "Sprungweite" der Branch-Befehle

Vielleicht ist Ihnen bereits aufgefallen, daß die in den Branch-Befehlen eingegeben Adressen in sehr merkwürdiger Art und

Weise codiert werden. Der im letzten Beispiel verwendete Branch-Befehl wurde in folgender Form umgesetzt:

```
A 03FE 10 F9              BPL $C002
```

Die Zahlen "10" und "F9" werden beim hinten abgedruckten Monitor nicht angezeigt. Benutzen Sie gegebenenfalls den Befehl

```
M 03FE
```

Anstelle der angegebenen Zwei-Byte-Adresse folgt dem Befehlscode für BPL ($10) das Byte $F9. Branch-Befehle sind sogenannte "relative Sprungbefehle". Anstelle der angegebenen Adresse legt der Monitor die Entfernung zu dieser Adresse im Speicher ab.

Nun beträgt die Entfernung im obigen Beispiel sicher weniger als $F9 (=dezimal 249) Byte. Geben Sie bitte zum Vergleich das folgende Programm ein.

```
A C000 A2 FF              LDX #$FF
A C002 D0 03              BNE $C007
A C004 8E 00 04           STX $0400
A C007 00                 BRK
```

In diesem (völlig sinnlosen) Programm erfolgt ein "Vorwärtssprung" zum Befehl BRK an Adresse $C007. Die Entfernung zu diesem Befehl wird mit dem Byte $03 angegeben.

Um diese Angabe zu verstehen, müssen Sie sich die Befehlsausführung in Erinnerung rufen. Der Befehlscode und die Adresse werden gelesen, wobei der Programmzähler jeweils inkrementiert wird.

Nach dem Einlesen des Befehls BNE $C007 weist der Programmzähler somit auf den Befehlscode des folgenden Befehls (STX $C000) und enthält daher die Adresse $C004. Die Angabe der "Sprungweite" bezieht sich auf diese Adresse, von der aus die Entfernung bis zur gewünschten Adresse $C007 genau drei Byte beträgt. Merken Sie sich daher bitte, daß die Angabe der

Sprungweite immer von der Adresse des dem Branch-Befehl folgenden Befehls ausgeht. Nun zu der seltsam großen Sprungweite $F9 aus dem vorhergehenden Beispiel. In der Angabe der Sprungweite muß eine Information darüber enthalten sein, ob es sich um einen "Vorwärts-" oder um einen "Rückwärtssprung" handelt.

Diese Information befindet sich im siebten Bit, das bekanntlich den Wert $80 (=dezimal 128) besitzt. "Sprungweitenbytes", die größer sind als $80, kennzeichnen somit einen Rückwärtssprung.

Diese "relative Adressierung" ist zwar platzsparend, da zur Angabe der Zieladresse ein Byte genügt, besitzt jedoch einen Nachteil. Zur Kennzeichnung eines relativen Sprungs stehen "pro Richtung" nur 128 verschiedene Zahlen zur Verfügung ($00-$7F für Vorwärtssprünge; $80-$FF für Rückwärtssprünge).

Relative Sprünge sind daher auf einen "Umkreis" von 128 Byte beschränkt. In einem großen Programm ist es leider nicht möglich, mit einem relativen Sprung von einem Ende des Programms zum anderen zu gelangen, im Gegensatz zum JMP-Befehl, mit dem zu beliebigen Adressen verzweigt werden kann, unabhängig von deren Entfernung.

Da Assembler-Programme jedoch sehr "kompakt" und kurz sind, reicht diese maximale Sprungweite in der Praxis für alle anfallenden Probleme völlig aus.

Zusammenfassung

1. Alle Branch-Befehle arbeiten mit der "relativen Adressierung", bei der anstelle der absoluten Adresse des Sprungziels die Entfernung zu der betreffenden Adresse im Speicher abgelegt wird.
2. Relative Sprünge sind nach "vorn" (in Richtung höherer Adressen) und nach "hinten" (in Richtung niedrigerer Adressen) möglich. Rückwärtssprünge werden durch ein gesetztes siebtes Bit gekennzeichnet.

3. Ein Vorteil der relativen Adressierung besteht in der Kürze der Adreßangabe. Die Entfernung wird im Gegensatz zur absoluten Adressierung (zwei Byte) mit nur einem Byte angegeben.
4. Der Nachteil der relativen Adressierung liegt in der beschränkten Sprungweite. Mit einem Branch-Befehl kann nur zu einer maximal 128 Byte entfernten Adresse verzweigt werden.

Übungsprogramme zur Schleifenbildung

Die letzten Kapitel boten einiges über Branch-Befehle und Schleifenbildung. Um es Ihnen zu ermöglichen, das angesammelte Wissen auch tatsächlich zu "verdauen", werden wir in diesem Abschnitt einige Programme verwirklichen, die alle mit Schleifen arbeiten.

Ball bewegen

Dieses erste Programmprojekt ähnelt einem früher vorgestellten Programm, das einen Ball über eine Bildschirmzeile bewegte.

Diesmal soll der Ball jedoch quer von rechts unten nach links oben (Spalte 23, Zeile 23, dann Spalte 22 und Zeile 22, zum Schluß Spalte 1, Zeile 1) über den gesamten Bildschirm bewegt werden. Außerdem soll die Bewegung erst fortgesetzt werden, wenn Sie eine beliebige Taste betätigen.

Um den Ball an der jeweils gewünschten Position auszugeben, setzen wir zuerst den Cursor mit der PLOT-Routine des Betriebssystems. Die Ausgabe des ASCII-Codes 113 ($71) malt das für den Ball verwendete Zeichen an der aktuellen Cursorposition.

Bevor der Ball an dieser Position durch Ausgabe eines Leerzeichens (ASCII-Code 32 = $20) wieder gelöscht wird, warten wir mit der GETIN-Routine auf eine Tastenbetätigung.

Lassen Sie sich von der Länge des folgenden Programm-Listings nicht abschrecken. Das Programm ist zwar recht lang, aber dennoch ohne größere Probleme zu verstehen.

Bei genauem Studium des Listings lernen Sie einiges über den effektiven Umgang mit den Registern.

```
Ball bewegen
      A C000 A9 93          LDA #$93     ;ASCII-CODE 147 AUSGEBEN
      A C002 20 D2 FF       JSR $FFD2    ;=> BILDSCHIRM LOESCHEN
      A C005 A2 17          LDX #$17     ;ZEILENZAEHLER INITIAL.

      A C007 8A             TXA          ;ZEILE NACH AKKU
      A C008 A8             TAY          ;AKKU NACH Y-REGISTER
      A C009 18             CLC
      A C00A 20 F0 FF       JSR $FFF0    ;CURSOR SETZEN

      A C00D A9 71          LDA #$71     ;ASCII-CODE FUER BALL
      A C00F 20 D2 FF       JSR $FFD2    ;AUSGEBEN
      A C012 86 FB          STX $FB      ;ZEILE (X) 'RETTEN'

      A C014 20 E4 FF       JSR $FFE4    ;TASTATUR ABFRAGEN
      A C017 F0 FB          BEQ $C014    ;TASTE GEDRUECKT?

      A C019 A6 FB          LDX $FB      ;ZEILE WIEDERHOLEN NACH X
      A C01B 8A             TXA          ;ZEILE NACH AKKU
      A C01C A8             TAY          ;AKKU NACH SPALTE
      A C01D 18             CLC
      A C01E 20 F0 FF       JSR $FFF0    ;CURSOR SETZEN

      A C021 A9 20          LDA #$20     ;ASCII-CODE F.LEERZEICHEN
      A C023 20 D2 FF       JSR $FFD2    ;AUSGEBEN

      A C026 CA             DEX          ;ZEILENZAEHLER DEKREMENT.
      A C027 D0 DE          BNE $C007    ;FERTIG?
      A C029 00             BRK          ;PROGRAMMENDE
```

Der Übersicht wegen sind im Listing Leerzeilen zwischen den verschiedenen Programmteilen eingefügt. Geben Sie diese Leerzeilen bitte nicht mit ein!

Zum Programmablauf: der erste - vorbereitende - Programmteil löscht den Bildschirm, indem mit BSOUT der ASCII-Code 147 ($93) ausgegeben wird. Dieser Code entspricht laut ASCII-Tabelle der Taste CLR, die bekanntlich ebenfalls den Bildschirm löscht. Der erste Programmteil lädt anschließend das X-Register mit dem Wert $17 (=dezimal 23). Das X-Register gibt im weiteren Programmverlauf die Zeile an, in der der Ball auszugeben ist.

Wie erläutert soll die Ballbewegung ab Position 23/23 (Spalte/Zeile) beginnen. Spalte und Zeile sind somit identisch. Daher wird der Inhalt des "Zeilenzählers" X in das Y-Register übertragen, bevor mit der PLOT-Routine der Cursor auf die angegebene Position gesetzt wird (X-Register=Zeile; Y-Register=Spalte).

Leider gibt es keinen Befehl, mit dem der Inhalt des X-Registers direkt in das Y-Registers kopiert werden kann. Daher nehmen wir einen "Umweg" über den Akkumulator. Mit TXA wird der Inhalt in den Akkumulator kopiert, und anschließend mit TAY von dort aus ins Y-Register "weiterbefördert".

Sowohl X- als auch Y-Register enthalten nun den Wert $17 und der Cursor kann auf diese Position gesetzt werden (CLC und JSR $FFF0).

Nun wird der ASCII-Code des "Balls" ausgegeben ($71). Der Ball erscheint in Spalte 23 von Zeile 23. Bevor mit der GETIN-Routine auf eine Taste gewartet wird, kopiert der Befehl STX $FB den Inhalt des X-Registers in die Zeropage-Speicherzelle $FB.

Der Grund: wie erläutert (siehe Betriebssystem-Routinen) verändert die GETIN-Routine die Inhalte der Register. Der aktuelle Wert unseres Zeilenzählers muß jedoch unbedingt erhalten bleiben! Dieser Wert wird daher in die Speicherzelle $FB "gerettet".

Die bereits bekannte Warteschleife ruft immer wieder GETIN auf, solange keine Taste betätigt wurde. Wie erläutert ist die Bedingung "verzweige, wenn gleich null" solange erfüllt, bis eine Taste betätigt wird (siehe GETIN-Routine).

Erfolgte eine beliebige Tastenbetätigung, wird der alte Inhalt des X-Registers zurückgeholt. Das X-Register wird mit dem zuvor nach $FB kopierten Wert geladen. Beachten Sie bitte, daß sowohl STX $FB als auch LDX $FB die erläuterte bytesparende Zeropage-Adressierung verwenden.

Ebenso wie bei der Ausgabe des Balls wird der Inhalt des X-Registers über den "Akkumulator-Umweg" ins Y-Register kopiert und anschließend der Cursor gesetzt, bevor ein Leerzeichen ausgegeben und der Ball damit wieder gelöscht wird.

Das erneute Setzen des Cursors ist nötig, da sich die Cursorposition nach der Ausgabe des Balls veränderte. BSOUT setzt den Cursor nach jedem ausgegebenen Zeichen um eine Spalte nach rechts.

Der erste Schleifendurchgang ist somit beendet. Der Schleifenzähler X wird dekrementiert. Solange Zeile null noch nicht erreicht ist, erfolgt ein weiterer Durchgang (solange DEX nicht zum Ergebnis null führt, ist das Zero-Flag gelöscht und damit die Bedingung "verzweige, wenn ungleich null" erfüllt).

Im zweiten Durchgang besitzt der Schleifenzähler einen um eins verminderten Wert. Der gesamte Vorgang wiederholt sich somit in Spalte 22 von Zeile 22 und so weiter.

Der prinzipielle Ablauf zusammengefaßt:

1. Das Ball-Zeichen wird in der durch den aktuellen X-Wert bestimmten Zeile (und identischer Spalte) ausgegeben.
2. Das Programm wartet auf die Betätigung einer beliebigen Taste.
3. Der zuvor ausgegebene Ball wird durch Überschreiben mit einem Leerzeichen wieder gelöscht, nachdem der Cursor zuvor wieder auf die exakte Ballposition gesetzt wurde.

4. Der Zeilenzähler wird dekrementiert und ein neuer Schleifendurchgang beginnt, außer wenn soeben Zeile 1 "bearbeitet" wurde.

String-Ausgabe

Dieses Programmprojekt ist sehr anspruchsvoll, liefert Ihnen aber dafür zum Schluß ein allgemein verwendbares Unterprogramm, das Sie in den verschiedensten Programmen benutzen können. Es demonstriert zugleich den effektiven Einsatz von BSOUT und der "Unterprogrammtechnik".

Unter einem String verstehen wir eine beliebige Zeichenkette. Bisher war es nur möglich, immer die gleichen Zeichen (A,B,...) auf dem Bildschirm auszugeben.

Bestimmt wünschen Sie sich eine Routine, die wie der PRINT-Befehl beliebige Zeichenketten ausgibt. Wir wissen bereits, wie mit Hilfe der BSOUT-Routine ein Zeichen ausgegeben wird. Der Akkumulator wird mit dem ASCII-Code des Zeichens geladen und BSOUT aufgerufen. Das Zeichen wird an der aktuellen Cursorposition ausgegeben und dieser um eine Stelle weiterbewegt.

Vor der eigentlichen Zeichenausgabe sollte jedoch der Cursor auf die gewünschte Position gesetzt werden. Geben Sie den folgenden Programmteil bitte noch nicht ein. Auf den folgenden Seiten wird das Listing komplett abgebildet.

```
A C000 A2 05        LDX #$05     ;ZEILE 5
A C002 A0 00        LDY #$00     ;SPALTE 0
A C004 18           CLC
A C005 20 F0 FF     JSR $FFF0    ;CURSOR SETZEN
```

Stören Sie sich nicht an dem noch unbekannten Befehl CLC, den die Betriebssystem-Routine zum Setzen des Cursors benötigt. Geben Sie das Programm (ohne Kommentare!) ein, rufen Sie es jedoch noch nicht auf.

Nun folgt der Hauptteil des Programms, die Stringausgabe. Der String muß (siehe BSOUT) in ASCII-Format im Speicher vorliegen. Wir benötigen einen unbenutzten Speicherbereich zur Ablage der einzelnen Bytes. Wir werden den Bereich $033C-$03F6 verwenden, da dieser Bereich beim C64 nur für Cassettenoperationen (LOAD, SAVE...) verwendet wird.

Die meisten unter Ihnen besitzen wahrscheinlich eine Floppy. Der Bereich $033C-$03F6 ist in diesem Fall immer unbenutzt. Ein Trost für Datasetten-Besitzer: auch Sie können diesen Bereich verwenden. Erst wenn Sie ein Programm laden/speichern, wird die Zeichenkette überschrieben werden.

Das Hauptprogramm soll Zeichen für Zeichen aus diesem Bereich lesen und mit BSOUT ausgeben, wozu wir die indizierte Adressierung verwenden.

```
A C008 A2 00          LDX #$00      ;X INITIALISIEREN
A C00A BD 03 3C       LDA $033C,X   ;ZEICHEN AN ADRESSE
                                    ;$033C + X
A C00D F0 07          BEQ $C016     ;ASCII-CODE = 0 ?
A C00F 20 D2 FF       JSR $FFD2     ;ZEICHEN AUSGEBEN
A C012 E8             INX           ;ZAEHLER INKREMENTIEREN
A C013 4C 0A C0       JMP $C00A     ;NAECHSTES ZEICHEN
A C016 00             BRK           ;ALLES AUSGEGEBEN
```

Der Ablauf dieses doch recht komplexen Programms:

1. Vor Beginn der Schleife ($C00A-$C013) wird das X-Register mit $00 geladen.
2. Der erste Schleifenbefehl (LDA $033C,X) lädt den Akkumulator mit dem Inhalt der Speicherzelle $033C plus X. Das X-Register besitzt beim ersten Durchgang den Inhalt $00, die endgültige Adresse ist daher $033C + $00 = $033C, also der Beginn unseres Strings.
3. Der Akkumulator enthält nun das erste Stringzeichen. BEQ $C016 prüft, ob das Ergebnis der letzten Operation (LDA) gleich null war. Wenn ja, wird zu Adresse $C016 verzweigt.

Der Sinn dieser Überprüfung: unsere Routine muß in der Lage sein, das Ende eines Strings zu erkennen. Wir markieren dieses Ende, indem dem letzten Zeichen das Byte $00 folgt. Wenn dieses Byte mit LDA $033C,X gelesen wird, verzweigt BEQ zum Programmende.

4. Das im ersten Durchgang gelesene Zeichen wird mit BSOUT ausgegeben (JSR BSOUT).
5. Der Zähler X, der momentan den Inhalt $00 besitzt, wird erhöht (INX) und zeigt damit im nächsten Durchgang auf das zweite Zeichen des Strings ($033C + $01 = $033D).
6. Der Befehl JMP $C00A verzweigt immer zum Schleifenanfang (unbedingter Sprungbefehl; nicht von einer Bedingung abhängig).
7. Im zweiten Durchgang wird der Akkumulator mit dem Inhalt von Adresse $033C + X, also $033D (X=$01) geladen, dem zweiten Zeichen des Strings. Wenn auch dieses Byte ungleich null ist, wird der Branch-Befehl BEQ wiederum nicht ausgeführt und das Zeichen ebenfalls mit BSOUT ausgegeben.
8. Das Programm "hangelt" sich auf diese Weise durch den gesamten String. X wird nach jedem Durchgang jeweils um eins erhöht und im folgenden Durchgang somit auf die nächste Speicherzelle zugegriffen (siehe "indizierte Adressierung").
9. Das Programm ist beendet, wenn mit dem LDA-Befehl das Byte $00 geladen wird, unsere "Endemarke", die wir an das Stringende anfügen werden. Da in diesem Fall die letzte Operation (LDA) das Ergebnis null ergab, ist die Bedingung BEQ (ADRESSE) "verzweige, wenn gleich null" erfüllt und der Branch-Befehl verzweigt zum Programmende.

Das Gesamtprogramm

```
A C000 A2 05        LDX #$05     ;ZEILE 5
A C002 A0 00        LDY #$00     ;SPALTE 0
A C004 18           CLC
A C005 20 F0 FF     JSR $FFF0    ;CURSOR SETZEN
A C008 A2 00        LDX #$00     ;X INITIALISIEREN
```

```
A C00A BD 3C 03          LDA $033C,X  ;ZEICHEN AN ADRESSE
                                      ;$033C + X
A C00D F0 07             BEQ $C016    ;ASCII-CODE = 0 ?
A C00F 20 D2 FF          JSR $FFD2    ;ZEICHEN AUSGEBEN
A C012 E8                INX          ;ZAEHLER INKREMENTIEREN
A C013 4C 0A C0          JMP $C00A    ;NAECHSTES ZEICHEN
A C016 00                BRK          ;ALLES AUSGEGEBEN
```

Nach der kompletten Eingabe dieses Programms müssen wir noch eine Zeichenkette ab $033C (=dezimal 828) ablegen. Am einfachsten ist eine BASIC-Schleife, die die ASCII-Codes eines Strings zeichenweise "pokt". Verlassen Sie den Monitor und geben Sie ein:

```
100 A$="DIES IST EIN TEST"
110 FOR I=1 TO LEN(A$):POKE 827+I,ASC(MID$(A$,I,1)):NEXT
120 POKE 827+I,0:REM ENDEMARKE $00
```

Starten Sie das BASIC-Programm und rufen Sie anschließend den Monitor auf. Das Assembler-Programm rufen Sie wie immer mit G C000 auf.

Wenn Sie das Programm korrekt eingaben, wird auf dem Bildschirm ab Spalte null in Zeile fünf die Zeichenkette "DIES IST EIN TEST" ausgegeben.

Das Programm können Sie beliebig variieren. Duch Änderung der X- und Y-Werte am Programmanfang bestimmen Sie die Ausgabeposition. Durch Änderung des Strings <A$> im BASIC-Programms kann die auszugebende Zeichenkette geändert werden.

Die Routine ist so allgemein, daß beliebige Zeichenketten ausgegeben werden, wenn folgende Bedingungen beachtet werden:

1. Die Zeichenkette muß ab $033C im Speicher abgelegt werden und darf nicht länger als 187 Zeichen sein ($033C-$03F6).

2. Dem letzten Zeichen muß unbedingt das Byte $00 folgen, da die Routine sonst das Stringende nicht erkennt.

Sollte sie die umständliche Speicherung der Zeichenkette frustrieren, für die wir ein eigenes BASIC-Programm benötigen: in einem späteren Kapitel werden Programmierhilfsmittel erläutert, die eine komfortablere Eingabe von Assembler-Programmen gestatten als ein Monitor (dennoch ist auch mit diesen Hilfsmitteln ein Monitor-Programm unverzichtbar!).

Bei Verwendung eines solchen Hilfsprogramms werden Texte unmittelbar eingegeben. Das Programm erledigt für Sie die Aufgabe, die einzelnen Textzeichen in ASCII-Form abzulegen.

Ein Tip: mit dem Hauptteil des vorgestellten Programms (ab LDX #$00) verfügen Sie über eine allgemeine Ausgabe-Routine, die Sie in beliebige Programme einbauen können. Geschickterweise bauen Sie die Routine als Unterprogramm ein, das mit RTS ("return from subroutine", etwa "verlasse das Unterprogramm") anstatt mit BRK beenden.

Das Unterprogramm kann von beliebigen Programmteilen aus mit JSR (ADRESSE) aufgerufen werden, wie im folgenden Beispiel:

```
Allgemeine String-Ausgaberoutine

A C000 A2 00          LDX #$00      ;X INITIALISIEREN
A C002 BD 3C 03       LDA $033C,X   ;ZEICHEN AN ADRESSE
                                    ;$033C + X
A C005 F0 07          BEQ $C00E     ;ASCII-CODE = 0 ?
A C007 20 D2 FF       JSR $FFD2     ;ZEICHEN AUSGEBEN
A C00A E8             INX           ;ZAEHLER INKREMENTIEREN
A C00B 4C 02 C0       JMP $C002     ;NAECHSTES ZEICHEN
A C00E 60             RTS           ;RETURN FROM UNTERPROG.

A C00F A9 93          LDA #$93      ;ASCII-CODE $93 (147)
A C011 20 D2 FF       JSR $FFD2     ;AUSGEBEN => CLEAR SCREEN
A C014 A0 0F          LDY #$0F      ;ZAEHLER MIT $0F LADEN
```

```
A C016 20 00 C0          JSR $C000     ;AUSGABE-UNTERPROG
                                       ;AUFRUFEN
A C019 A9 0D             LDA #$0D      ;ASCII-CODE $0D (13)
A C01B 20 D2 FF          JSR $FFD2     ;AUSGEBEN =>
                                       ;ZEILENVORSCHUB
A C01E 88                DEY           ;ZAEHLER DEKREMENTIEREN
A C01F D0 F5             BNE $C016     ;SCHLEIFE BEENDET?
A C021 00                BRK           ;STRING 16MAL AUSGEGEBEN
```

Zur besseren optischen Unterscheidung befindet sich eine (nicht einzugebende!) Leerzeile zwischen dem Unter- und dem Hauptprogramm.

Das Hauptprogramm erfordert eine eigene Erläuterung. Ziel des Hauptprogramms ist es, zuerst den Bildschirm zu löschen und anschließend mit Hilfe der Ausgaberoutine den vom BASIC-Programm angelegten String <A$> 15mal auszugeben.

Zum Löschen des Bildschirms wird das Steuerzeichen der Taste CLR verwendet (ASCII-Code 147, als Hex-Zahl $93). Wie beschrieben können mit BSOUT beliebige ASCII-Codes ausgegeben werden.

Der erzielte Effekt ist identisch mit dem BASIC-Befehl PRINT CHR$(147). Da die Ausgaberoutine in einer Schleife 16mal aufgerufen wird, lädt LDY #$0F das als Schleifenzähler verwendete Y-Register mit diesem Wert.

Das Unterprogramm befindet sich vor dem Hauptprogramm und wird mit JSR $C000 aufgerufen, der Inhalt von <A$> einmal ausgegeben.

Nun wird mit BSOUT der ASCII-Code $0D (dezimal 13) ausgegeben. Wenn Sie in der ASCII-Tabelle nachschauen, stellen Sie fest, daß der Code 13 der RETURN-Taste zugeordnet ist. Die Ausgabe dieses Codes bewirkt somit einen "Zeilenvorschub", der Cursor wird auf die erste Spalte der nächsten Zeile gesetzt.

Das Zählregister Y wird mit DEY vermindert und die Schleife erneut durchlaufen, wenn die Dekrementierung nicht zum Er-

gebnis null führte. Wenn doch, wird der Branch-Befehl BNE $C016 nicht ausgeführt und das Programm mit BRK beendet.

Geben Sie dieses Programm bitte ein und starten Sie es nicht mit G C000, sondern mit G C00F. Bedenken Sie, das Hauptprogramm beginnt erst hinter dem Ende des Unterprogramms!

Das Demoprogramm zeigt nicht nur den effektiven Einsatz von BSOUT, sondern auch die Gliederung von Haupt- und Unterprogrammen.

Allgemeine Unterprogramme sollten Sie bei Eingabe mit dem Monitor immer vor dem Hauptprogramm eingeben. Unabhängig vom folgenden Hauptprogramm befinden sich die Unterprogramme dann immer an der gleichen Stelle im Speicher und können immer gleich aufgerufen werden (in diesem Fall mit JSR $C000).

Befindet sich das Hauptprogramm vor dem Unterprogramm, ändert sich die Startadresse des Unterprogramms, je nach Länge des davor liegenden Hauptprogramms. Jedesmal, wenn Sie die Routine in einem Programm verwenden wollen, müssen Sie sich anschauen, ab welcher Adresse das Unterpropramm nun diesmal beginnt.

Bei dem gezeigten Unterprogramm müssen Sie sogar die Sprungadresse des JMP-Befehls, den das Unterprogramm enthält, jedesmal ändern, ein völlig unnötiger Arbeitsaufwand, den Sie sich mit der beschriebenen Methode ersparen.

2.13 Rechnen in Maschinensprache

Das Rechnen in Maschinensprache ist nicht ganz so einfach wie in BASIC, da unser Prozessor nur über Additions- und Subtraktionsbefehle verfügt. Das Programm läßt sich jedoch lösen, indem Multiplikationen und Divisionen auf die grundlegenden Operationen "Addition/Subtraktion" zurückgeführt werden.

Lassen Sie sich durch diese Einschränkung nicht entmutigen. In den wenigsten Fällen wird in einem Assembler-Programm eine Multiplikation oder Subtraktion benötigt. Wir programmieren ja nicht in Assembler, um eine Zinsberechnung durchzuführen oder ein Buchhaltungsprogramm zu schreiben. Für derartige Probleme ist eine höhere Programmiersprache wie BASIC weitaus besser geeignet.

Mit Assembler wollen wir blitzartig Daten im Speicher hin- und herschaufeln, oder vielleicht Sortier- und Eingaberoutinen für BASIC-Programme schreiben (der BASIC-Befehl INPUT ist eine Zumutung). In allen diesen Fällen werden wir die Rechenfähigkeiten von BASIC niemals vermissen.

Doch nun zu den Additions- und Subtraktionsbefehlen, die wir in vielen Programmen benötigen. Arithmetische Operationen finden immer im Akkumulator statt.

Das heißt, wenn wir zwei Zahlen addieren wollen, laden wir die erste Zahl in den Akkumulator und geben anschließend an, welche Zahl zum Akkumulatorinhalt zu addieren ist. Das Ergebnis einer solchen arithmetischen Operation befindet sich anschließend ebenfalls immer im Akkumulator!

ADC ("subtract with carry", "addiere mit Carry-Bit") ist der Additions-Befehl unseres Rechners, SBC ("subtract with carry", "subtrahiere mit Carry-Bit") der Subtraktionsbefehl.

Das "Carry-Bit" oder "Carry-Flag" ist Bit Nummer 0 des Status-Registers (ganz rechts in der Dualzahl %00000000). Wozu benötigen wir dieses Flag bei arithmetischen Operationen?

Bedenken Sie folgendes: der Akkumulator kann als 8-Bit-Register wie jede andere Speicherzelle auch nur Werte zwischen null und 255 annehmen. Angenommen, wir addieren die Zahlen 255 ($FF oder dual %11111111) und eins ($01 oder dual %00000001). Das Ergebnis 256 ist größer als 255 und muß daher zwangsläufig falsch dargestellt werden.

Um auszuprobieren, was sich in diesem Fall ereignet, geben Sie bitte ein:

```
A C000 A9 FF        LDA #$FF     ;AKKU MIT $FF LADEN
A C002 18           CLC          ;WIRD NOCH ERLÄUTERT
A C003 69 01        ADC #$01     ;$01 ZUM AKKU ADDIEREN
A C005 00           BRK
```

Nach dem Aufruf dieses Programms stellen Sie anhand der Registeranzeige fest, daß der Akkumulator den Wert null ($00) enthält!

Analog den Inkrementierbefehlen wird somit nach dem maximalen Wert $FF wieder am "entgegengesetzten" Ende begonnen, mit dem Wert $00. Wenn Sie den Befehl ADC #$01 durch ADC #$02 ersetzen, wird daher das Ergebnis (im Akkumulator) $02 lauten.

Interessant ist nun, daß im Status-Register das Bit 0 (das Carry-Flag) gesetzt wurde. Die rechte Ziffer des Status-Registers lautet $7, also %0111. Ändern Sie zum Vergleich den Befehl LDA #$FF in LDA #$F0 und rufen Sie das Programm erneut auf.

Die ausgegebene rechte Ziffer des Status-Registers ist nun - nach der Addition von $F0 und $01 (=$F1) - eine 4, also dual %0100, das Carry-Flag ist gelöscht. Am Zustand des Carry-Flags können wir somit erkennen, ob ein "Übertrag" stattfand und das Ergebnis größer ist als $FF.

Das Carry-Bit wird nicht automatisch gelöscht, wenn das Ergebnis einer Addition $FF nicht überschreitet. Wir müssen daher dafür sorgen, daß es vor der Addition unbedingt gelöscht ist, wenn es uns nicht "in die Irre führen" soll.

Mit dem Befehl CLC ("clear carry", "lösche das Carry-Bit") können wir dieses Bit "per Hand" löschen, um auszuschalten, daß es bereits vor der Addition zufällig (durch irgendeine vorhergehende Operation) noch gesetzt ist. Wenn wir auf diese Weise

vorgehen, zeigt uns das Carry-Bit zuverlässig an, ob die Addition zu einem Übertrag führte.

Das Carry-Flag besitzt außer dieser Übertragsanzeige eine weitere Funktion. Es stellt eine Art "Verlängerung" des Akkumulators dar, gewissermaßen dessen "neuntes Bit" (Bits 0-8). Das folgende Schema zeigt den Aufbau beider 8-Bit-Register mit dem Carry-Bit als Bit 0 des Status-Registers.

```
Status-Register      Akkumulator
N V - B D I Z C <--  0 0 0 0 0 0 0 0
```

Der Sinn des Carry-Bits wird klar, wenn wir eine Addition vornehmen, bei der das Ergebnis größer ist als $FF.

```
255    11111111
+ 1   +00000001
-------------
256   100000000
```

Wie Sie sehen, sind alle Bits einer achtstelligen Dualzahl gelöscht. Zur Darstellung dieser Zahl werden neun Bits benötigt. Als dieses neunte Bit fungiert das Carry-Flag (=Carry-Bit), das bei einem solchen Übertrag gesetzt wird. Der Befehl ADC ("addiere mit Carry") addiert nicht nur zwei Zahlen, sondern zählt immer das Carry-Flag hinzu.

Wichtig ist diese Arbeitsweise bei der Addition von Zahlen, die größer sind als $FF, also mit zwei Byte dargestellt werden.

Solche Zwei-Byte-Zahlen (oder gleichbedeutend "16-Bit-Zahlen") kennen Sie bereits, nämlich absolut angegebene Adressen ($0400, $C000 etc.). Wir wir wissen, werden diese Adressen in zwei Speicherzellen im Format Low-Byte/High-Byte gespeichert.

Die Zahl $0400 wird zum Beispiel so gespeichert: $00 $04, das heißt in einer Speicherzelle befindet sich das Low-Byte und in der folgenden das High-Byte der Zahl.

Angenommen, in den Speicherzellen $033C-$033F befinden sich zwei solcher 16-Bit-Zahlen, die wir addieren und in den Speicherzellen $0340-$0341 ablegen wollen.

```
Speicherzellen  Inhalt(Low-Byte/High-Byte)
 $033C/$033D          $01/$10 (=$1001)
 $033E/$033F          $FF/$10 (=$10FF)
```

Prinzipiell werden 16-Bit-Werte addiert, indem zuerst die Low-Bytes und anschließend die High-Bytes addiert werden. Das Schema:

```
1.    LADE AKKU MIT DEM INHALT VON $033C ; LOW-BYTE VON ZAHL 1 ($01)
2.    ADDIERE DEN INHALT VON $033E       ; LOW-BYTE VON ZAHL 2 ($FF)
3.    ERGEBNIS IN $0340 ABLEGEN          ; LOW-ERGEBNIS ($00)
4.    LADE AKKU MIT DEM INHALT VON $033D ; HIGH-BYTE VON ZAHL 1 ($10)
5.    ADDIERE DEN INHALT VON $033F       ; HIGH-BYTE VON ZAHL 2 ($10)
6.    ERGEBNIS IN $0341 ABLEGEN          ; HIGH-ERGEBNIS ($20)
```

Nach Ausführung dieser Operationen enthält die Speicherzelle $0340 das "niederwertige" Ergebnis $00, die Speicherzelle $0341 das "höherwertige" Ergebnis $20. Als Gesamtergebnis ergibt sich $2000, das in $0340/$0341 im Low-/High-Format ($00/$20) gespeichert ist.

Das Ergebnis ist offensichtlich falsch ($1001 + $10FF = $2100), da der Übertrag nicht berücksichtigt wurde, der sich bei der Addition der Low-Bytes ergab.

Der Befehl ADC berücksichtigt diesen Übertrag netterweise für uns. Die Addition der Low-Bytes führt zu einem Übertrag und das Carry-Bit wird gesetzt. Bei der folgenden Addition $10 + $10 = $20 wird das Carry-Bit (gesetzt: 1; gelöscht: 0) zum Ergebnis addiert und es ergibt sich korrekterweise der Wert $21.

```
Addition von $1001 und $10FF
      A C000 A9 01          LDA #$01     ;LOW-BYTE VON $1001 IN
      A C002 8D 3C 03       STA $033C    ;$033C ABLEGEN
      A C005 A9 10          LDA #$10     ;HIGH-BYTE VON $1001 IN
      A C007 8D 3D 03       STA $033D    ;$033D ABLEGEN
```

```
A C00A A9 FF        LDA #$FF     ;LOW-BYTE VON $10FF IN
A C00C 8D 3E 03     STA $033E    ;$033E ABLEGEN
A C00F A9 10        LDA #$10     ;HIGH-BYTE VON $10FF IN
A C011 8D 3F 03     STA $033F    ;$033F ABLEGEN

A C014 18           CLC          ;CARRY-FLAG LOESCHEN !!!
A C015 AD 3C 03     LDA $033C    ;LOW-BYTE VON ZAHL 1
A C018 6D 3E 03     ADC $033E    ;UND ZAHL 2 ADDIEREN
A C01B 8D 40 03     STA $0340    ;ERGEBNIS NACH $0340
A C01E AD 3D 03     LDA $033D    ;HIGH-BYTE VON ZAHL 1
A C021 6D 3F 03     ADC $033F    ;UND ZAHL 2 ADDIEREN
A C024 8D 41 03     STA $0341    ;ERGEBNIS NACH $0341
A C027 00           BRK
```

Das Programm besteht aus zwei Teilen. Im ersten Teil werden die Zahlen $1001 und $10FF im Format Low-Byte/High-Byte in $033C-$033F abgelegt.

Diese 16-Bit-Zahlen werden im zweiten Teile addiert. Unumgänglich ist zuvor der Befehl CLC ("clear carry", also "lösche das Carry-Bit"), um dafür zu sorgen, daß das Carry-Bit nicht noch als Ergebnis irgendeiner früheren Addition gesetzt ist und das Ergebnis verfälscht.

Zuerst werden die Low-Bytes addiert und das Ergebnis in $0340 abgelegt. Nun werden die High-Bytes addiert und zum Ergebnis der Wert des Carry-Bits (0 oder 1) dazugezählt. Im Beispiel wurde das Carry-Bit durch den Übertrag bei der Addition der Low-Bytes gesetzt, zur folgenden "höherwertigen" Addition zählt der Prozessor daher noch eine eins dazu.

Der Übertrag ist damit korrigiert und das Ergebnis, das in $0341 abgelegt wird, stimmt. Sie können sich davon überzeugen, indem Sie eingeben:

```
M 0340
```

Der Monitor gibt nach diesem Befehl die Werte der Speicherzellen $0340-$0347 aus. In den ersten beiden Speicherzellen ($0340

und $0341) befindet sich unser Ergebnis $00 und $21, also die Zahl $2100 im Format Low-Byte/High-Byte.

Die Addition von zwei 16-Bit-Zahlen erfolgt somit automatisch völlig korrekt, wenn wir vor den Additionen das Carry-Flag mit dem CLC-Befehl löschen.

Merken Sie sich bitte, daß der Befehl ADC zum Akkumulator-Inhalt einen Wert (oder den Inhalt einer Speicherzelle) plus dem Carry-Bit (0 oder 1) addiert.

Ähnlich einfach können zwei 16-Bit-Zahlen subtrahiert werden. Zuerst werden die Low-, dann die High-Bytes mit dem Befehl SBC ("subtract with carry") subtrahiert. Einen eventuellen "Untertrag" bei der Subtraktion der Low-Bytes (Beispiel: $01-$02) berücksichtigt der SBC-Befehl automatisch. Diesmal muß jedoch vor Beginn der Subtraktion das Carry-Flag nicht gelöscht, sondern mit dem Befehl SEC ("set carry-flag", "setze das Carry-Flag") gesetzt werden.

Merken Sie sich als allgemeine Regel: Vor Additionen wird das Carry-Flag mit dem Befehl CLC gelöscht, vor Subtraktionen mit dem entgegengesetzt wirkendem Befehl SEC gesetzt!

```
Subtraktion von $2001 und $0002
      A C000 A9 01            LDA #$01     ;LOW-BYTE VON $2001 IN
      A C002 8D 3C 03         STA $033C    ;$033C ABLEGEN
      A C005 A9 20            LDA #$20     ;HIGH-BYTE VON $2001 IN
      A C007 8D 3D 03         STA $033D    ;$033D ABLEGEN
      A C00A A9 02            LDA #$02     ;LOW-BYTE VON $0002 IN
      A C00C 8D 3E 03         STA $033E    ;$033E ABLEGEN
      A C00F A9 00            LDA #$00     ;HIGH-BYTE VON $0002 IN
      A C011 8D 3F 03         STA $033F    ;$033F ABLEGEN

      A C014 18               SEC          ;CARRY-FLAG SETZEN !!!
      A C015 AD 3C 03         LDA $033C    ;LOW-BYTE VON ZAHL 1
      A C018 6D 3E 03         SBC $033E    ;UND ZAHL 2 SUBTRAHIEREN
      A C01B 8D 40 03         STA $0340    ;ERGEBNIS NACH $0340
      A C01E AD 3D 03         LDA $033D    ;HIGH-BYTE VON ZAHL 1
```

```
A C021 6D 3F 03     SBC $033F    ;UND ZAHL 2 SUBTRAHIEREN
A C024 8D 41 03     STA $0341    ;ERGEBNIS NACH $0341
A C027 00           BRK
```

Dieses Programm subtrahiert von $2001 die Zahl $0002. Bei der Subtraktion der Low-Bytes ($01-$02) entsteht ein "Untertrag", der automatisch berücksichtigt wird. Mit M 0340 wird Ihnen das korrekte Ergebnis $1FFF angezeigt ($2001 - $0002 = $1FFF).

Eine weitere Anwendung der Rechenoperationen demonstriert das folgende Programm. Es ermittelt die Länge eines BASIC-Programms (in Byte) und gibt sie auf dem Bildschirm aus.

Um dieses Programm zu verstehen, müssen Sie jedoch wissen, daß in der Zeropage die Adressen enthalten sind, an denen ein BASIC-Programm beginnt beziehungsweise endet. Die "BASIC-Adressen" sind beim C64 in den Speicherzellen $2B/$2C (BASIC-Anfang) beziehungsweise in $2D/$2E (BASIC-Ende) untergebracht (im Adreßformat, also zuerst Low- und dann High-Byte).

Programmlänge ausgeben

```
A C000 38           SEC          ;SUBTRAKTION VORBEREITEN
A C001 A5 2D        LDA $2D      ;SUBTRAHIEREN:
                                 ;LOW-BYTE ENDE
A C003 E5 2B        SBC $2B      ;MINUS LOW-BYTE ANFANG
A C005 AA           TAX          ;ERGEBNIS NACH X BRINGEN
A C006 A5 2E        LDA $2E      ;SUBTRAHIEREN:
                                 ;HIGH-BYTE ENDE
A C008 E5 2C        SBC $2C      ;MINUS HIGH-BYTE ANFANG
A C00A 20 CD BD     JSR $BDCD    ;2-BYTE-ZAHL AUSGEBEN
A C00D 00           BRK
```

Wie bei Subtraktionen erforderlich, wird das Carry-Flag gesetzt. Im Anschluß daran wird die Endadresse des BASIC-Programms (in $2D/2E) von der Startadresse (in $2B/$2C) subtrahiert.

Zuerst werden die Low-Bytes subtrahiert. Das Ergebnis wird vom Akkumulator in das X-Register übertragen. Anschließend

werden die High-Bytes subtrahiert. Danach befindet sich das Low-Byte des Ergebnisses im X-Register und das High-Byte im Akkumulator.

Genau diese Parameter (X-Register: Low-Byte; Akkumulator: High-Byte) verlangt die Routine INTOUT ($BDCD), eine Routine des BASIC-Interpreters, mit der beliebige Integerwerte auf dem Bildschirm an der aktuellen Cursorposition ausgegeben werden.

Das Ergebnis dieses kleinen Programms ist nicht völlig korrekt. Aus Gründen, deren Erläuterung an dieser Stelle zu weit führt, wird nicht die echte Länge des BASIC-Programms ausgegeben, sondern die Länge plus zwei. Im Normalfall stört diese winzige Differenz jedoch kaum.

Wenn Sie wollen, können Sie Ihr frisch erworbenes Wissen anwenden und vom ermittelten Ergebnis die Zahl zwei subtrahieren, um ein völlig korrektes Ergebnis zu erhalten.

2.14 Die Vergleichsbefehle

Bei der Schleifenbildung lernten wir die Branch-Befehle BNE, BEQ, BPL und BMI kennen. Diese Befehle verwendeten wir als Kombination aus einer Bedingung (IF..THEN) und einem Sprungbefehl (GOTO).

Der Haken an dem Einsatz dieser Befehle war jedoch, daß nur festgelegte Bedingungen zulässig war. BEQ und BNE erkennen, ob der Wert $00 geladen (LDA, LDX, LDY) oder durch In-/Dekrementierung (DEX, DEY, INX, INY) erreicht wurde.

BPL und BMI stellten fest, ob ein Lade- oder In-/Dekrementierbefehl zu einem Ergebnis führt, das größer ist als $80.

Was tun wir jedoch, wenn wir eine Schleife mit dem Startwert eins und dem Endwert zehn benötigen? Wir vergleichen unseren Zähler mit dem gewünschten Endwert.

Die "Compare"- oder "Vergleichs"-Befehle CMP, CPX und CPY vergleichen den Inhalt einer angegebenen Speicherzelle (oder eines Registers) mit einem beliebigen Wert. Der Vergleichswert kann unmittelbar (CMP #$0A, CPX #$0A, CPY #$0A) oder absolut (als Inhalt einer Speicherzelle: CMP $0400, CPX $0400, CPY $0400) angegeben werden.

Selbstverständlich ist auch die kürzere (und schnellere) Zeropage-Adressierung möglich (CMP $04). Voraussetzung ist natürlich, daß sich der Vergleichswert in einer Speicherzelle der Zeropage befindet.

Damit sind alle Adressierungsarten der Befehle CPX und CPY genannt. Der Befehl CMP (der dem Akkumulatorinhalt mit einem Wert vergleicht) gestattet zusätzlich die indizierte Adressierung (CMP $0400,X = "vergleiche den Akku mit dem Inhalt von Speicherzelle $0400 + X"). Nähere Angaben zu den möglichen Adressierungsarten der Compare-Befehle finden Sie im Anhang.

Die Vergleichsbefehle werden erst in diesem Kapitel eingeführt, weil das vorige Kapitel die Grundlage zum Verständnis bildet. Bei einem Vergleichsbefehl wird der Vergleichswert automatisch vom Inhalt des Akkumulators (bei CMP) oder einem der Indexregister (bei CPX und CPY) abgezogen.

Der Registerinhalt bleibt jedoch unverändert erhalten! Dieses merkwürdige Verhalten können Sie sich so vorstellen: der Vergleichswert wird abgezogen, anschließend wird der ursprüngliche Registerinhalt jedoch wiederhergestellt.

Die Frage ist, welchen Sinn diese Subtraktion ohne Resultat besitzt. Ein Vergleichsbefehl setzt oder löscht - je nach Ergebnis der Subtraktion - die drei kennengelernten Flags (Zero-Flag, Negativ-Flag, Carry-Flag).

Folgende Ergebnisse werden durch den Zustand der Flags angezeigt:

1. Der Inhalt des Registers ist größer als der Vergleichswert: Carry-Flag gesetzt (1), Negativ- und Zero-Flag gelöscht (0).
2. Der Inhalt des Registers ist genauso groß wie der Vergleichswert: Carry-Flag und Zero-Flag gesetzt (1), Negativ-Flag gelöscht (0).
3. Der Inhalt des Registers ist kleiner als der Vergleichswert: Negativ-Flag gesetzt (1), Carry-Flag und Zero-Flag gelöscht (0).

Ein Beispiel: Der Befehl CMP #$0A vergleicht den Inhalt des Akkumulators mit dem unmittelbar angegebenen Wert $0A. Angenommen, der Akkumulator enthält ebenfalls den Wert $0A. Bei der folgenden Subtraktion wird $0A (der Vergleichswert) von $0A (dem Akkumulator-Inhalt) subtrahiert.

Das Ergebnis ist null und sowohl das Carry- als auch das Zero-Flag ist gesetzt. Mit BEQ ("verzweige, wenn gleich null") können wir in diesem Fall verzweigen.

Beachten Sie bitte, was das Negativ-Flag (und somit die Befehle BPL und BMI) unter "größer" beziehungsweise "kleiner" verstehen. Wenn zum Beispiel ein Register den Wert $98 enthält und mit $05 verglichen wird, wird das Negativ-Flag nicht gelöscht, sondern gesetzt!

Warum? Weil $98-$05 (die "automatische Subtraktion" bei Vergleichsbefehlen) zum Ergebnis $94 führt, einem "negativen" Ergebnis (positiv=$00-$79, "negativ"=$80-$FF).

Merken Sie sich daher bitte: BPL verzweigt nicht immer, wenn der Registerinhalt größer ist als der Vergleichswert, sondern nur dann, wenn die Subtraktion REGISTER - VERGLEICHSWERT zu einem "positiven" Ergebnis führt (Ergebnis kleiner als $80). Gleiches gilt umgekehrt für BMI.

Diese Besonderheit sollte bei der im Folgenden erläuterten Schleifenbildung unbedingt beachtet werden.

2.15 Schleifen und Vergleichsbefehle

Vergleichsbefehle können hervorragend zur Schleifenbildung eingesetzt werden. Wir sahen, daß uns die Flags eindeutige Auskünfte darüber liefern, ob ein Registerinhalt größer, gleich oder kleiner als ein Vergleichswert ist.

Um nun abhängig vom Ergebnis bestimmte Operationen auszuführen, benutzen wir die Branch-Befehle, die bekanntlich die Flags testen und je nach Flag-Zustand verzweigen oder nicht.

Im Normalfall wollen wir unterscheiden, ob der Schleifenzähler größer, gleich oder kleiner als der Vergleichswert ist, analog den folgenden BASIC-Schleifen.

Die Funktion der Branch-Befehle BEQ, BNE, BPL und BMI könnte (ohne näher auf die Flags einzugehen) so ausgedrückt werden:

- BEQ: Verzweige, wenn Register gleich Vergleichswert.
- BNE: Verzweige, wenn Register ungleich Vergleichswert.
- BPL: Verzweige, wenn Register größer oder gleich Vergleichswert und die Differenz nicht größer ist als $79.
- BMI: Verzweige, wenn Register kleiner als Vergleichswert und die Differenz nicht größer ist als $79.

Was der Zusatz "und die Differenz nicht größer ist als $79" bei BPL und BMI zu bedeuten hat, wurde im vorigen Abschnitt erläutert.

Mit dieser Kurzbeschreibung können wir problemlos die folgenden BASIC-Schleifen in Assembler schreiben.

```
Verzweige, solange Schleifenzähler ungleich Vergleichswert (BASIC)
      100 X=0
      110 PRINT "A";
      120 X=X+1
      130 IF X<>10 THEN GOTO 110
```

```
Verzweige, solange Schleifenzähler ungleich Vergleichswert (Assembler)
      A C000 A2 00            LDX #$00     ;ZAEHLER MIT $00 LADEN
      A C002 A9 41            LDA #$41     ;ASCII-CODE FUER 'A'
      A C004 20 D2 FF         JSR $FFD2    ;'A' AUSGEBEN
      A C007 E8               INX          ;ZAEHLER ERHOEHEN
      A C008 E0 0A            CPX #$0A     ;ZAEHLER MIT $0A
                                           ;VERGLEICHEN
      A C00A D0 F8            BNE $C004    ;WENN UNGLEICH:
                                           ;WEITERMACHEN
      A C00C 00               BRK
```

Dieses Programm verwirklicht zum ersten Mal eine Schleife, die "aufwärts" zählt. Endlich, werden wohl viele sagen. Der Ablauf entspricht der BASIC-Variante 1, die üblicherweise mit einer FOR..NEXT-Schleife verwirklicht wird.

```
100 FOR I=0 TO 10
110 : PRINT"A";
120 NEXT
```

Der Zähler - das X-Register, zu Beginn mit dem Inhalt $00 "initialisiert" - wird nach jedem Schleifendurchgang inkrementiert und anschließend mit $0A verglichen (CPX #$0A). Das Zero-Flag wird nur dann gesetzt, wenn beide Werte gleich groß sind (und die Subtraktion daher den Wert null ergibt), bei Ungleichheit wird es gelöscht.

Der Befehl BNE testet dieses Flag und kann daher verwendet werden, um zu prüfen, ob das X-Register bereits den Wert $0A enthält. Solange X nicht den Vergleichswert $0A enthält, verzweigt der BNE-Befehl wieder zum Schleifenanfang.

Das nächste A wird ausgegeben und X wieder inkrementiert. Der Vorgang wiederholt sich zehnmal (X= 0,1..,8,9). Beim zehnten Durchgang besitzt X den Wert $0A, BNE stellt Gleichheit zwischen Register und Vergleichswert fest und die Schleife wird verlassen.

```
Verzweige, solange Schleifenzähler größer/gleich Vergleichswert (BASIC)
      100 X=100
      110 PRINT "A";
      120 X=X-1
      130 IF X>=10 THEN GOTO 110

Verzweige, solange Schleifenzähler größer/gleich Vergleichswert
(Assembler)
      A C000 A2 64            LDX #$64      ;ZAEHLER MIT $64 LADEN
      A C002 A9 41            LDA #$41      ;ASCII-CODE FUER 'A'
      A C004 20 D2 FF         JSR $FFD2     ;'A' AUSGEBEN
      A C007 CA               DEX           ;ZAEHLER VERMINDERN
      A C008 E0 0A            CPX #$0A      ;ZAEHLER MIT $0A
                                            ;VERGLEICHEN
      A C00A 10 F8            BPL $C004     ;WENN GRÖSSER/GLEICH:
                                            ;WEITERMACHEN
      A C00C 00               BRK
```

In diesem Programm wird der Zähler mit $64 (=dezimal 100) geladen und nach jedem Durchgang dekrementiert. CPX #$0A vergleicht das X-Register mit dem Wert $0A, indem vom Registerinhalt $0A subtrahiert werden.

Die Flags werden je nach Ergebnis der Subtraktion gesetzt oder gelöscht und anschließend mit BPL das Negativ-Flag getestet. BPL verzweigt, wenn das Negativ-Flag gelöscht ist. Dies ist der Fall, wenn der Registerinhalt größer oder gleich dem Vergleichswert ist. Erst wenn diese Bedingung nicht mehr erfüllt ist (X kleiner ist als der Vergleichswert $0A), verzweigt BPL nicht mehr.

Bei der Verwendung von BPL und BMI ist große Vorsicht geboten (denken Sie daran, was im letzten Kapitel zu den Begriffen "größer"/"kleiner" und deren spezieller Bedeutung für diese Befehle gesagt wurde).

Wenn Sie im vorliegenden Programm den ersten Befehl (LDX #$64) durch LDX #$FF ersetzten und das Programm erneut aufrufen, werden Sie verblüfft feststellen, daß ein einziges A ausgegeben wird.

Nach dem ersten Durchgang wird der Inhalt von X ($FE) mit $0A verglichen. Die Subtraktion ergibt $F4, ein "negatives" Ergebnis. Das Negativ-Flag wird gesetzt und BPL verzweigt nicht. Größer bedeutet für BPL, daß das Ergebnis der Subtraktion beim Vergleichsbefehl "positiv" ist, also kleiner als $80.

Entsprechendes gilt umgekehrt für BMI (das Ergebnis wird als größer gewertet, wenn das Register um mehr als $79 kleiner ist als der Vergleichswert).

BPL und BMI sollten daher nur für "kleine" Schleifen mit maximal $79 (127) Durchgängen eingesetzt werden, um unangenehme Überraschungen zu vermeiden.

BCS und BCC

Die Vergleichsbefehle setzen oder Löschen außer dem Zero- und dem Negativ-Flag auch das Carry-Flag. Es bietet sich daher an, auch dieses Flag zur Schleifenbildung auszunutzen. Uns fehlen jedoch noch die Befehle, die dieses Flag testen und entsprechend verzweigen oder nicht verzweigen.

BCS ("branch on carry set", "verzweige bei gesetztem Carry-Bit") verzweigt, wenn das Carry-Flag gesetzt ist. Ein Vergleichsbefehl setzt das Carry-Flag immer dann, wenn der Registerinhalt größer als gleich dem Vergleichswert ist. In diesem Fall verzweigt BCS.

BCS wird daher gern als Ersatz für BPL verwendet, um Probleme mit "negativen" Vergleichsergebnissen zu vermeiden (wenn X minus Vergleichswert größer ist als $79).

BCC ("branch on carry clear", "verzweige bei gelöschtem Carry-Bit"), verzweigt im umgekehrten Fall, wenn das Carry-Flag gelöscht ist, das heißt wenn beim Vergleich ("automatische Subtraktion") ein "Unterlauf" stattfand. Ein solcher "Unterlauf" findet statt, wenn das Register (sein Inhalt) kleiner ist als der Ver-

gleichswert. BCC besitzt nicht die Problematik von BMI (siehe oben) und wird daher eingesetzt, wenn die Differenz zwischen Register und Vergleichswert größer als $79 ist.

Das letzte Beispiel mit BCS anstatt BPL:

```
A C000 A2 64        LDX #$64     ;ZAEHLER MIT $64 LADEN
A C002 A9 41        LDA #$41     ;ASCII-CODE FUER 'A'
A C004 20 D2 FF     JSR $FFD2    ;'A' AUSGEBEN
A C007 CA           DEX          ;ZAEHLER VERMINDERN
A C008 E0 0A        CPX #$0A     ;ZAEHLER MIT $0A
                                 ;VERGLEICHEN
A C00A B0 F8        BCS $C004    ;WENN GRÖSSER/GLEICH:
                                 ;WEITERMACHEN
A C00C 00           BRK
```

Noch einmal, da dies oft falsch gemacht wird: im Gegensatz zu BPL und BMI ist für BCS und BCC uninteressant, ob ein Vergleichsergebnis "positiv" oder "negativ" ist.

Beide Befehle können immer problemlos eingesetzt werden, auch dann, wenn zu Beginn der Schleife die Differenz zwischen Zähler und Vergleichswert größer ist als $79.

Denken Sie jedoch bitte daran, daß die in Schleifen häufig verwendeten Befehle INX, DEX, INY und DEY das Carry-Flag nicht beeinflussen!

Eine Schleife wie

```
          LDX #$0A
(ADRESSE) DEX
          BCS (ADRESSE)
```

ist daher eine Endlosschleife. Wenn wie in diesem Beispiel BCS das Carry-Flag testen soll, muß zuvor ein Vergleich des X-Registers mit $00 stattfinden.

```
          LDX #$0A
(ADRESSE) DEX
                          CMP #$00
          BCS (ADRESSE)
```

Von den bisher behandelten Befehlen beeinflussen das Carry-Flag nur ADC, SBC, CLC, SEC und die Vergleichsbefehle CMP, CPX, CPY.

Noch ein wichtiger Hinweis: Angenommen, ADRESSE - also das Sprungziel - befindet sich nicht vor dem Befehl DEX, sondern vor LDX #$0A. Auch die letzte Version bildet in diesem Fall eine Endlosschleife, da das X-Register nicht wirklich heruntergerzählt, sondern nach jeder Verzweigung wieder von Neuem mit dem Ausgangswert $0A geladen wird.

Achten Sie bitte darauf, daß das Ziel einer Verzweigung niemals die "Initialisierung", das Laden eines Registers mit einem Startwert, sondern immer der folgende Befehl ist, also der erste Befehl innerhalb der Schleife.

Schleifenbildung: Zusammenfassung

Wie die vorausgegangenen Kapitel zeigten, ist die Schleifenbildung in Assembler sehr komplex. Sie werden viel Übung benötigen, bis Sie für jeden Fall die optimalen Schleife bilden können.

In diesem Abschnitt will ich weniger über Flags sprechen, sondern Ihnen "Faustregeln" für die Anwendung der Vergleichs- und Sprungbefehle geben.

Um Ihnen den Umgang mit den Sprungbefehlen zu erleichtern, beschreibe ich deren Funktion noch einmal im wenigen Worten. Beachten Sie bitte unbedingt die Besonderheiten bei BPL und BMI.

- BEQ: Verzweige, wenn Register gleich Vergleichswert.
- BNE: Verzweige, wenn Register ungleich Vergleichswert.

- BCS: Verzweige, wenn Register größer oder gleich dem Vergleichswert.
- BCC: Verzweige, wenn Register kleiner als Vergleichswert.
- BPL: Verzweige, wenn Register größer oder gleich Vergleichswert und die Differenz nicht größer ist als $79.
- BMI: Verzweige, wenn Register kleiner als Vergleichswert und die Differenz nicht größer als $79.

Merken Sie sich bitte vor allem, daß bei Schleifen unbedingt zu berücksichtigen ist, ob die letzte Operation vor einem Branch-Befehl auch tatsächlich das vom betreffenden Befehl getestete Flag beeinflußt.

Folgende bisher besprochene Befehle beeinflussen die uns bekannten Flags (Zero-Flag, Negativ-Flag, Carry-Flag):

Befehl	Beeinflußte Flags
LDA	Negativ, Zero
LDX	Negativ, Zero
LDY	Negativ, Zero
TAX	Negativ, Zero
TAY	Negativ, Zero
TXA	Negativ, Zero
TYA	Negativ, Zero
INX	Negativ, Zero
INY	Negativ, Zero
INC	Negativ, Zero
DEX	Negativ, Zero
DEY	Negativ, Zero
DEC	Negativ, Zero
ADC	Negativ, Zero, Carry
SBC	Negativ, Zero, Carry
CMP	Negativ, Zero, Carry

Denken Sie bitte daran: alle Branch-Befehle testen den aktuellen Zustand eines Flags und prüfen somit das Ergebnis der letzten

Operation, die das betreffende Flag beeinflußte. Was passiert, wenn diese Tatsache nicht berücksichtigt wird, zeigt das folgende Beispiel:

```
          LDX #$0A
(ADRESSE) ...
          ...
          DEX
          LDA #$FF
          BNE $(ADRESSE)
```

In diesem Beispiel wird nach der Dekrementierung des Zählers der Akkumulator mit $FF geladen. BNE testet das Zero-Flag und verzweigt, wenn es gelöscht ist. Die Schleife soll beendet werden, wenn das X-Register den Wert null annimmt und daher das Zero-Flag gesetzt wird.

Der folgende LDA-Befehl beeinflußt das Zero-Flag jedoch ebenfalls. Da $FF ungleich null ist, wird das Zero-Flag unmittelbar vor dem Test mit BNE gelöscht und BNE verzweigt immer (Endlosschleife)!

Abhilfe schafft ein zusätzlicher Vergleichsbefehl, der das X-Register mit $00 vergleicht und nun als letzter Befehl vor BNE das Zero-Flag beeinflußt.

```
          LDX #$0A
(ADRESSE) ...
          ...
          DEX
          LDA #$FF
          CPX #$00
          BNE $(ADRESSE)
```

Zugegeben, dieses Beispiel ist nicht allzu sinnvoll. Im Normalfall wird man den "automatischen" Vergleich mit $00 vorziehen, der bei DEX erfolgt und eine andere Reihenfolge der Befehle wählen (LDA #$FF : DEX : BNE$(ADRESSE)), durch die sich der Vergleichsbefehl erübrigt.

In der Praxis können jedoch sehr wohl Fälle auftreten, in denen tatsächlich ein expliziter Vergleich des Zählregisters mit $00 angebracht ist, weil nach der Dekrementierung des Zählregisters Befehle folgen, die das getestete Flag beeinflussen und daher das Ergebnis der Dekrementierung verfälschen.

Denken Sie bitte daran, daß bei der Schleifenbildung mit BCC und BCS ein vorangehender Vergleichsbefehle notwendig ist, da die In-/Dekrementierbefehle das von BCC und BCS getestete Carry-Flag nicht beeinflussen.

Übungsprogramme zur Schleifenbildung

Sie kennen nun fast alle Branch-Befehle und können theoretisch bereits Schleifen beliebiger Art bilden. Der richtige Umgang mit den Branch-Befehlen setzt jedoch viel Übung voraus. Daher wird sich dieses Kapitel nicht mit weiteren Befehlen, sondern ausschließlich mit Übungsprogrammen beschäftigen.

Steuerung per Tastatur

Mit den Vergleichs- und den Branch-Befehlen sind wir in der Lage, ein Programm zu schreiben, das das Grundelement aller Videospiele demonstriert, die Steuerung eines Objektes durch den Benutzer.

Als Objekt verwenden wir einen Ball, der mit den Cursortasten in beliebige Richtungen bewegt wird. Das Prinzip kann jedoch ebensogut auf Sprites oder Shapes übertragen werden.

Der prinzipielle Ablauf:

1. Ein Zeilenzähler (X-Register) und ein Spaltenzähler (Y-Register) wird ständig die aktuelle Ballposition festhalten.
2. Ein vorbereitender Programmteil löscht den Bildschirm und setzt das X-Register auf den Wert zwölf (Zeile zwölf) und das Y-Register auf den Wert 20 (Spalte 20). Die da-

durch festgelegte Position soll in diesem Fall immer die Ausgangsposition des Balls sein.

3. In der nun beginnenden Schleife wird der Ball an der durch X und Y festgelegten Position gezeichnet.
4. Nach der Ausgabe des Balls wird die Tastatur abgefragt. Wurde eine Taste betätigt, wird der Ball an der aktuellen Position gelöscht.
5. Je nach betätigter Cursortaste wird der Zeilenzähler (X-Register) oder der Spaltenzähler (Y-Register) entsprechend verändert.
6. Mit einem absoluten Sprungbefehl (JMP) wird immer zum Schleifenbeginn zurückgekehrt und der Ball an der neuen Position ausgegeben (siehe 3.).

Vorbereitung

```
A C000 A9 93            LDA #$93     ;ASCII-CODE FÜR CLEAR
A C002 20 D2 FF         JSR $FFD2    ;SCREEN AUSGEBEN
A C005 A2 0C            LDX #$0C     ;ZEILENZÄHLER: $0C (=12)
A C007 A0 14            LDY #$14     ;SPALTENZÄHLER: $14 (=20)
Ball an aktueller Position ausgeben
A C009 18               CLC          ;VORBEREITUNG F.PLOT-
                                     ;ROUTINE
A C00A 20 F0 FF         JSR $FFF0    ;PLOT => CURSOR SETZEN
A C00D A9 71            LDA #$71     ;ASCII-CODE FÜR
                                     ;BALLZEICHEN
A C00F 20 D2 FF         JSR $FFD2    ;BSOUT => BALL AUSGEBEN
Tastatur abfragen
A C012 8E 3C 03         STX $033C    ;X-WERT 'RETTEN'
A C015 8C 3D 03         STY $033D    ;Y-WERT 'RETTEN'
A C018 20 E4 FF         JSR $FFE4    ;GETIN => TASTATUR
                                     ;ABFRAGEN
A C01B F0 FB            BEQ $C018    ;TASTE GEDRÜCKT?
A C01D 8D 3E 03         STA $033E    ;TASTE: ZEICHENCODE
                                     ;'RETTEN'
Ball an aktueller Position löschen
A C020 AE 3C 03         LDX $033C    ;X-WERT ZURÜCKHOLEN
A C023 AC 3D 03         LDY $033D    ;Y-WERT ZURÜCKHOLEN
A C026 18               CLC          ;VORBEREITUNG F.PLOT-
                                     ;ROUTINE
```

```
A C027 20 F0 FF          JSR $FFF0    ;PLOT => CURSOR SETZEN
A C02A A9 20             LDA #$20     ;ASCII-CODE FÜR
                                      ;LEERZEICHEN
A C02C 20 D2 FF          JSR $FFD2    ;BSOUT => BALL LÖSCHEN
Spalten- und Zeilenzähler korrigieren
A C02F AD 3E 03          LDA $033E    ;TASTE: ZEICHENCODE HOLEN
A C032 C9 11             CMP #$11     ;CODE FÜR 'CURSOR DOWN'?
A C034 D0 04             BNE $C03A    ;SPRUNG, WENN NEIN
A C036 E8                INX          ;SONST ZEILE ERHÖHEN
A C037 4C 09 C0          JMP $C009    ;UND ZUM SCHLEIFENANFANG
A C03A C9 1D             CMP #$1D     ;CODE FÜR 'CURSOR RIGHT'?
A C03C D0 04             BNE $C042    ;SPRUNG, WENN NEIN
A C03E C8                INY          ;SONST SPALTE ERHÖHEN
A C03F 4C 09 C0          JMP $C009    ;UND ZUM SCHLEIFENANFANG
A C042 C9 91             CMP #$91     ;CODE FÜR 'CURSOR UP'?
A C044 D0 04             BNE $C04A    ;SPRUNG, WENN NEIN
A C046 CA                DEX          ;SONST ZEILE VERMINDERN
A C047 4C 09 C0          JMP $C009    ;UND ZUM SCHLEIFENANFANG
A C04A C9 9D             CMP #$9D     ;CODE FÜR 'CURSOR LEFT'?
A C04C D0 BB             BNE $C009    ;SPRUNG, WENN NEIN
A C04E 88                DEY          ;SONST SPALTE VERMINDERN
A C04F 4C 09 C0          JMP $C009    ;UND ZUM SCHLEIFENANFANG
```

Um Ihnen den Überblick zu erleichtern, wurde das Listing entsprechend dem prinzipiellen Ablauf unterteilt und mit (nicht abzutippenden) Kommentaren versehen.

Der vorbereitende Teil löscht den Bildschirm, indem der ASCII-Code der Taste CLR (147 = $93) ausgegeben wird. Anschließend werden die Register mit jenen Werten "initialisiert", die der Bildschirmmitte entsprechen. Denken Sie beim folgenden Ablauf immer daran, daß das X-Register die Zeile und das Y-Register die Spalte bestimmen soll.

Teil 2 bildet den Schleifenanfang. Der Cursor wird mit der PLOT-Routine des Betriebssystems auf die durch X und Y festgelegte Bildschirmposition gesetzt. Anschließend wird an der jeweiligen Position mit BSOUT das Ballzeichen (ASCII-Code $71 = 113) ausgegeben.

Der folgende Teil enthält eine Warteschleife, die mit der Routine GETIN auf die Betätigung einer Taste wartet. Da GETIN alle drei Register verändert, müssen zuvor die aktuellen X- und Y-Werte "gerettet" werden. Die Inhalte dieser Register werden in die Speicherzellen $033C und $033D kopiert. Nachdem die Warteschleife verlassen wird, enthält der Akkumulator den ASCII-Code der gedrückten Taste. Da der Akkumulatorinhalt vom folgenden Programmteilteil verändert wird, muß dieser ASCII-Code ebenfalls "gerettet" werden (in $033E).

Der Cursor wird nun wieder mit PLOT auf die aktuelle Ballposition gesetzt (nachdem zuvor die ursprünglichen Werte der Indexregister aus den Speicherzellen zurückgeholt wurden) und der Ball durch Überschreiben mit einem Leerzeichen (ASCII-Code $20 = 32) gelöscht.

Den letzten Teil des Programms bildet die Veränderung der Indexregister je nach betätigter Cursortaste. Dieser Programmteil zeigt, wie mit Hilfe der Vergleichsbefehle ein Register mit beliebigen Werten verglichen wird.

Der ASCII-Code der betätigten Taste wurde in $033E zwischengespeichert (STA $033E). Dieser Code wird nun zurückgeholt (LDA $033E) und mit den ASCII-Codes der vier Cursortasten (DOWN, RIGHT, UP, LEFT) verglichen.

Der Befehl CMP #$11 vergleicht den Inhalt des Akkumulators mit dem Wert $11, dem ASCII-Code der Taste CURSOR DOWN. Bei Ungleichheit verzweigt BNE nach $C03A, ansonsten wird der folgende Befehl INX ausgeführt und der Zeilenzähler erhöht. Nach dieser Korrektur der Ballzeile erfolgt ein Sprung zum Schleifenanfang (JMP $C009), wo der Ball an der neu festgelegten Position ausgegeben wird.

Die folgenden Abschnitte dieses Programmteils verlaufen analog. Der Akkumulator wird mit dem ASCII-Code einer Cursortaste verglichen. Bei Ungleichheit wird zum nächsten Abschnitt verzweigt. Findet keine Verzweigung statt, wurde die mit CMP überprüfte Taste betätigt und die durch X und Y festgelegte Ballposition entsprechend korrigiert.

Nach dem Aufruf dieses Programms können Sie den "Ball" mit den vier Cursortasten in beliebige Richtungen steuern.

Achten Sie bitte darauf, den "Ball" nicht "aus dem Bildschirm heraus" zu bewegen, da sonst "unkontrolliert" Speicherzellen außerhalb des Bildschirmspeichers verändert werden (und dies kann zu merkwürdigen Resultaten führen).

Ich stelle Ihnen nun ein BASIC-Programm vor, dessen Ablauf exakt dem Assembler-Programm entspricht.

Dessen Ablauf kann durch Vergleich mit den entsprechenden Abschnitten des BASIC-Programms sehr gut nachvollzogen werden.

```
100 PRINT CHR$(147):REM BILDSCHIRM LOESCHEN
110 X=12:Y=20:REM POSITION INITIALISIEREN
120 :
130 POKE 214,X:POKE 211,Y:SYS 58732:REM CURSOR SETZEN
140 PRINT CHR$(113);:REM BALL AUSGEBEN
150 :
160 GET A$:IF A$="" THEN 160
170 A=ASC(A$):REM ASCII-CODE DER TASTE
180 :
190 POKE 214,X:POKE 211,Y:SYS 58732:REM CURSOR SETZEN
200 PRINT CHR$(32);:REM BALL LOESCHEN
210 :
220 IF A<>17 THEN 250:REM UNGLEICH CURSOR DOWN?
230 X=X+1:REM ZEILE ERHOEHEN
240 GOTO 130:REM ZUM SCHLEIFENANFANG
250 IF A<>29 THEN 280:REM UNGLEICH CURSOR RIGHT?
260 Y=Y+1:REM SPALTE ERHOEHEN
270 GOTO 130:REM ZUM SCHLEIFENANFANG
280 IF A<>145 THEN 310:REM UNGLEICH CURSOR UP?
290 X=X-1:REM ZEILE VERMINDERN
300 GOTO 130:REM ZUM SCHLEIFENANFANG
310 IF A<>157 THEN 130:REM UNGLEICH CURSOR LEFT?
320 Y=Y-1:REM SPALTE VERMINDERN
330 GOTO 130:REM ZUM SCHLEIFENANFANG
```

Eine Erläuterung ist zum Verständnis des BASIC-Programms notwendig: die Befehle POKE 214,X:POKE 211,Y:SYS 58732 sind "maschinennah"!

Beim C64 kann der Cursor direkt auf eine bestimmte Bildschirmposition gesetzt werden, wenn in Speicherzelle 214 die gewünschte Zeile und in 211 die Spalte "gepokt", und anschließend eine Betriebssystem-Routine an Adresse 58732 aufgerufen wird.

Sollten Sie sich wundern, daß zum Setzen des Cursors mit der PLOT-Routine das Carry-Flag mit CLC gelöscht werden muß, beim Setzen von BASIC aus jedoch die Übergabe von Spalte und Zeile völlig ausreicht: PLOT besitzt eigentlich zwei Funktionen, das Setzen des Cursors und das "Lesen" der aktuellen Cursorposition. Welche Funktion Sie wählen, bestimmen Sie mit Hilfe des Carry-Flags.

Wenn dieses Flag vor dem Aufruf mit JSR $FFF0 nicht gelöscht, sondern mit SEC gesetzt wird, wird die momentane Cursorposition von PLOT ermittelt und in den Registern übergeben. Nach dem Aufruf enthält das Y-Register die Spalte und das X-Register die Zeile.

Verbesserung des Steuerungsprogramms

Das vorgestellte Programm eignet sich hervorragend, um Ihnen einige Feinheiten der Assembler-Programmierung zu zeigen.

Ein echter Schwachpunkt des Programms ist die fehlende "Bereichsüberprüfung". Ein Beispiel: der Ball befindet sich in der obersten Zeile und der Benutzer betätigt CURSOR UP, um ihn eine Zeile noch oben zu bewegen. Da dieser Spezialfall nicht abgefragt wird, wird der Zeilenzähler X dekrementiert und enthält anschließend den Wert $FF ($00 minus $00 => $FF), eine "ein wenig" hohe Zeilennummer.

Das Problem wird am einfachsten gelöst, indem überprüft wird, ob durch die "Neupositionierung" die Bildschirmgrenzen über-

schritten würden. Wenn ja, wird ohne Positionsänderung sofort zum Schleifenanfang zurückgekehrt.

Der Programmteil zur Korrektur der Ballposition wird wie folgt erweitert:

Spalten- und Zeilenzähler korrigieren

```
A C02F AD 3E 03          LDA $033E     ;TASTE: ZEICHENCODE HOLEN
A C032 C9 11             CMP #$11      ;CODE FÜR 'CURSOR DOWN'?
A C034 D0 08             BNE $C03E     ;SPRUNG, WENN NEIN
A C036 E0 17             CPX #$17      ;X MIT $17(=23)
                                       ;VERGLEICHEN
A C038 B0 CF             BCS $C009     ;SPRUNG, WENN
                                       ;GRÖSSER/GLEICH
A C03A E8                INX           ;SONST ZEILE ERHÖHEN
A C03B 4C 09 C0          JMP $C009     ;UND ZUM SCHLEIFENANFANG
A C03E C9 1D             CMP #$1D      ;CODE FÜR 'CURSOR RIGHT'?
A C040 D0 08             BNE $C04A     ;SPRUNG, WENN NEIN
A C042 C0 26             CPY #$26      ;Y MIT $26(=38)
                                       ;VERGLEICHEN
A C044 B0 C3             BCS $C009     ;SPRUNG, WENN
                                       ;GRÖSSER/GLEICH
A C046 C8                INY           ;SONST SPALTE ERHÖHEN
A C047 4C 09 C0          JMP $C009     ;UND ZUM SCHLEIFENANFANG
A C04A C9 91             CMP #$91      ;CODE FÜR 'CURSOR UP'?
A C04C D0 08             BNE $C056     ;SPRUNG, WENN NEIN
A C04E E0 00             CPX #$00      ;X MIT $00 VERGLEICHEN
A C050 F0 B7             BEQ $C009     ;SPRUNG, WENN GLEICH
A C052 CA                DEX           ;SONST ZEILE VERMINDERN
A C053 4C 09 C0          JMP $C009     ;UND ZUM SCHLEIFENANFANG
A C056 C9 9D             CMP #$9D      ;CODE FÜR 'CURSOR LEFT'?
A C058 D0 AF             BNE $C009     ;SPRUNG, WENN NEIN
A C05A C0 00             CPY #$00      ;Y MIT $00 VERGLEICHEN
A C05C F0 AB             BEQ $C009     ;SPRUNG, WENN GLEICH
A C05E 88                DEY           ;SONST SPALTE VERMINDERN
A C05F 4C 09 C0          JMP $C009     ;UND ZUM SCHLEIFENANFANG
```

Bis $C034 bleibt das Programm unverändert. Zur Eingabe des korrigierten Teils geben Sie daher dem Assemble-Befehl als Startadresse $C034 an (A C034 BNE $C03E).

Am Beispiel des Programmteils $C032-$C03B (Funktion BALL NACH UNTEN) können die Änderungen stellvertretend für die folgenden Programmteile erläutert werden.

Bevor der Zeilenzähler X erhöht wird, erfolgt eine Überprüfung, ob der untere Bildschirmrand bereits erreicht wurde. Der Befehl CPX #$17 vergleicht den Inhalt des X-Registers mit dem Wert $17 (=dezimal 23) und setzt die Flags entsprechend.

BCS testet das Carry-Flag und verzweigt, wenn der Vergleich ergab, daß der X-Wert größer oder gleich $17 ist, das heißt wenn der untere Rand bereits erreicht ist. Ist X kleiner als $17, verzweigt BCS nicht und der Zeilenzähler wird wie in der letzten Programmversion inkrementiert.

Entsprechendes gilt für die folgenden Teile. Der Spaltenzähler Y wird bei CURSOR RIGHT nur dann inkrementiert, wenn der Vergleich CPY #$26 (=dezimal 38) ergab, daß Y kleiner ist als $26. Wenn Y dagegen gleich oder größer als $26 ist, verzweigt BCS zum Programmanfang.

Da es nicht möglich ist, daß der Zeilenzähler X den Wert $17 überschreitet beziehungsweise Y größer wird als $26, kann im vorliegenden Programm BCS durch BEQ ersetzt werden.

Ein Beispiel hierzu: angenommen, der Benutzer gibt "Dauerfeuer" mit CURSOR RIGHT. Das Y-Register wird solange inkrementiert, bis der Wert $26 erreicht ist. Besitzt Y diesen Wert, ergibt der Befehl CMP #$26 Gleichheit und BEQ verzweigt ebenso wie zuvor BCS (größer oder gleich) zum Schleifenanfang.

Die Funktionen CURSOR UP und CURSOR LEFT vergleichen das X- beziehungsweise Y-Register mit $00. Ist X gleich $00 (oberste Bildschirzeile), wird verzweigt (BEQ verzweigt bei Gleichheit) und der Zeilenzähler nicht dekrementiert. Ebenso wird verzweigt, wenn der Spaltenzähler Y gleich $00 ist.

Blinkende Bildschirmzeile

Das folgende Demoprogramm dient zur Auffrischung Ihres Wissens über die indirekte Adressierung. Wir kennen alle Befehle und Adressierungsarten, die wir benötigen, um eine Bildschirmzeile blinken zu lassen.

Blinken bedeutet: die betreffende Zeile muß abwechselnd "invertiert" und wieder "normalisiert" werden. Da wir im folgenden Programm nicht mit BSOUT arbeiten werden, sondern mit LDA und STA direkt auf den Bildschirmspeicher zugreifen, schauen Sie sich bitte die Bildschirmcode-Tabelle in Ihrem Handbuch an.

Am Ende der Tabelle befindet sich eine kleine Notiz: "Die Codes 128 bis 255 bilden die reversen Zeichen zu den Codes 0 bis 127". Das heißt, indem wir zu einem bestimmten Bildschirmcode 128 ($80) addieren, erhalten wir die inverse Darstellung des betreffenden Zeichens.

```
Bildschirmcode   Zeichen
      1            A
      2            B
...
...
    129            A invers
    130            B invers
...
...
```

Mit der indizierten Adressierung können wir sehr elegant der Reihe nach auf die 40 Zeichen einer Zeile zugreifen, diese lesen, $80 addieren und wieder (invers) zurückschreiben. Das Programm wird in mehreren Schritten entwickelt.

```
Version 1
      A C000 A2 00          LDX #$00     ;ZÄHLER INITIALISIEREN
      A C002 BD 00 04       LDA $0400,X  ;$0400 + X NACH AKKU
      A C005 18             CLC          ;ADDITION VORBEREITEN
      A C006 69 80          ADC #$80     ;$80 ADDIEREN
      A C008 9D 00 04       STA $0400,X  ;AKKU NACH $0400 + X
      A C00B E8             INX          ;ZÄHLER ERHÖHEN
```

```
A C00C E0 28        CPX #$28      ;X MIT $28 (40)
                                  ;VERGLEICHEN
A C00E D0 F2        BNE $C002     ;SPRUNG, WENN UNGLEICH
A C010 00           BRK           ;PROGRAMMENDE
```

In diesem Programm wird eine aufwärts zählende Schleife benutzt. Im ersten Durchgang besitzt X den Inhalt $00. In den Akku wird daher der Bildschirmcode an Adresse $0400 + X = $0400 geladen.

Mit CLC wird die Addition vorbereitet, $80 (=dezimal 128) addiert und der "invertierte Bildschirmcode" nach $0400 zurückgeschrieben.

Der Zähler wird inkrementiert und mit $28 (=dezimal 40) verglichen. Wurde die Zeile noch nicht komplett bearbeitet, ist X kleiner also $28 und BNE verzweigt, da Ungleichheit festgestellt wird (statt BNE könnte selbstverständlich ebenso BCC ("verzweige, wenn kleiner") verwendet werden.

Im nächsten Durchgang wiederholt sich der Vorgang. Da X jedoch nun den Wert $01 besitzt, wird auf $0401 ($0400 + X) zugegriffen.

Bereits früher erwähnte ich, daß in Assembler-Programmen meist abwärts zählende Schleifen verwendet werden. Warum dies so ist, zeigt das vorliegende Programm. Ob die Bildschirmzeile von rechts nach links (aufwärts zählend) oder aber von links nach rechts (abwärts zählend) invertiert wird, ergibt für die praktische Anwendung sicher keinen Unterschied. Mit einer abwärts zählenden Schleife ist das Programm jedoch kürzer (und eleganter).

Version 2

```
A C000 A2 27        LDX #$27      ;ZÄHLER INITIALISIEREN
A C002 BD 00 04     LDA $0400,X   ;AKKU MIT $0400 + X LADEN
A C005 18           CLC           ;ADDITION VORBEREITEN
A C006 69 80        ADC #$80      ;$80 ADDIEREN
A C008 9D 00 04     STA $0400,X   ;AKKU NACH $0400 + X
A C00B CA           DEX           ;ZÄHLER VERMINDERN
```

```
A C00C 10 F4          BPL $C002     ;SPRUNG, WENN POSITIV
A C00E 00             BRK           ;PROGRAMMENDE
```

In dieser abwärts zählenden Schleife wird X mit $27 initialisiert und daher zuerst auf das letzte Zeichen der Zeile zugegriffen ($0400 + $27 = $0427). Bei jedem Durchgang wird der Zähler dekrementiert, das Programm bearbeitet die Zeile beginnend mit Adresse $0427 und arbeitet sich bis zu Adresse $0400 "zurück".

Zur Überprüfung, ob die letzte Speicherzelle $0400 bereits behandelt wurde, muß BPL eingesetzt werden. BPL verzweigt, wenn das Ergebnis der letzten Operation positiv war (zwischen $00 und $79). Die letzte Operation besteht aus der Dekrementierung des Zählregisters, das vom Startwert $27 heruntergezählt wird.

Auch beim Wert $00 soll noch einmal verzweigt werden, um im folgenden Durchgang die Speicherzelle $0400 zu "behandeln". Zu diesem Zweck ist BPL hervorragend geeignet, da auch das Ergebnis $00 noch positiv ist und erst im folgenden Durchgang ein negatives Ergebnis ($FF) erzielt wird.

Wenn Sie BPL durch BNE ersetzen, werden Sie feststellen, daß die Speicherzelle $0400 nicht mehr invertiert wird. Wenn das X-Register den Wert $01 enthält und nun zu $00 dekrementiert wird, verzweigt BNE ("verzweige, wenn nicht gleich null") nicht mehr.

Sie sehen, für Schleifen, die auch beim Zählerwert null ein letztes Mal durchlaufen werden sollen, ist BPL optimal geeignet. Voraussetzung: die Schleife soll nicht mehr als 127mal durchlaufen werden.

Ist diese Voraussetzung nicht erfüllt, wird ein zusätzlicher Vergleichsbefehl benötigt und BNE oder BCS eingesetzt.

```
        LDX #$(STARTWERT)          LDX #$(STARTWERT)
(ADRESSE) ...              (ADRESSE) ...
        ...                        ...
        DEX                        DEX
```

```
CMP #$00                    CMP #$FF
BCS $(ADRESSE)              BCS (ADRESSE)
```

Sehen Sie sich diese verschiedenen Versionen bitte genau an. Bei der Programmierung in Assembler ist es außerordentlich wichtig, den Unterschied zwischen den verschiedenen Branch-Befehle zu kennen, um diese optimal einsetzen zu können.

Kommen wir zum zweiten Programmteil, der "Normalisierung" der Zeile, die analog verläuft, abgesehen davon, daß nicht $80 addiert, sondern subtrahiert wird.

```
A C000 A2 27          LDX #$27      ;ZÄHLER INITIALISIEREN
A C002 BD 00 04       LDA $0400,X   ;AKKU MIT $0400 + X LADEN
A C005 38             SEC           ;SUBTRAKTION VORBEREITEN
A C006 E9 80          SBC #$80      ;$80 SUBTRAHIEREN
A C008 9D 00 04       STA $0400,X   ;AKKU NACH $0400 + X
A C00B CA             DEX           ;ZÄHLER VERMINDERN
A C00C 10 F4          BPL $C002     ;SPRUNG, WENN POSITIV
A C00E 00             BRK           ;PROGRAMMENDE
```

Beachten Sie bitte, daß das Carry-Bit gesetzt werden muß, um die Subtraktion vorzubereiten.

Es bietet sich nun an, beide Programmteile nicht einfach "hintereinanderzuhängen", sondern ein wenig eleganter miteinander zu verbinden. Eine Möglichkeit besteht darin, nur eine Schleife zu verwenden und zu prüfen, ob eine Addition oder aber eine Subtraktion vorzunehmen ist.

```
Version 3
A C000 A2 27          LDX #$27      ;X INITIALISIEREN
A C002 BD 00 04       LDA $0400,X   ;$0400 + X IN AKKU
A C005 30 06          BMI $C00D     ;SPRUNG, WENN NEGATIV
A C007 18             CLC           ;SONST ADD.VORBEREITEN
A C008 69 80          ADC #$80      ;UND $80 ADDIEREN
A C00A 4C 10 C0       JMP $C010     ;SUBTRAKTION ÜBERSPRINGEN
A C00D 38             SEC           ;SUBTRAKTION VORBEREITEN
A C00E E9 80          SBC #$80      ;$80 SUBTRAHIEREN
A C010 9D 00 04       STA $0400,X   ;AKKU NACH $0400 + X
```

```
A C013 CA              DEX          ;ZÄHLER VERMINDERN
A C014 10 EC           BPL $C002    ;SPRUNG, WENN POSITIV
A C016 4C 00 C0        JMP $C000    ;NEUSTART DES PROGRAMMS
```

Erstaunlicherweise ist dieses Programm nur unwesentlich länger als einer der beiden Teile "Zeile invertieren" und "Zeile normalisieren".

Der Grund liegt im äußerst effektiven Einsatz von BPL. Nach dem Laden des Inhaltes der Speicherzelle $0400,X prüft BPL, ob der geladene Wert positiv oder negativ ist.

Ist er negativ (größer als $79), verzweigt BMI ("verzweige, wenn negativ") zum Teil "$80 subtrahieren" und das offensichtlich bereits invertierte Zeichen wird normalisiert.

Ist der geladene Wert dagegen positiv, also kleiner oder gleich $79, ist das Zeichen normal dargestellt und muß invertiert werden. Entsprechend wird $80 addiert und der folgende Programmteil "$80 subtrahieren" einfach übersprungen (JMP $C010).

Das Programm prüft somit selbstständig, ob die Zeichen dieser Zeile normalisiert oder bereits invertiert sind und kehrt den jeweiligen Zustand um. Wurde die Zeile komplett behandelt, erfolgt mit JMP $C000 ein Neustart. Die Zeile wird abwechselnd invertiert und wieder normalisiert.

Diese dritte Version zeigt, wie schnell Assembler wirklich ist. Die "Blinkfrequenz" ist so hoch, daß wir nur ein Flackern wahrnehmen. Die Konsequenz: wir benötigen eine Verzögerungsschleife.

```
Version 4
Bildschirmzeile invertieren/normalisieren
      A C000 A2 27           LDX #$27     ;X INITIALISIEREN
      A C002 BD 00 04        LDA $0400,X  ;$0400 + X IN AKKU
      A C005 30 06           BMI $C00D    ;SPRUNG, WENN NEGATIV
      A C007 18              CLC          ;SONST ADD.VORBEREITEN
      A C008 69 80           ADC #$80     ;UND $80 ADDIEREN
```

```
A C00A 4C 10 C0          JMP $C010    ;SUBTRAKTION ÜBERSPRINGEN
A C00D 38                SEC          ;SUBTRAKTION VORBEREITEN
A C00E E9 80             SBC #$80     ;$80 SUBTRAHIEREN
A C010 9D 00 04          STA $0400,X  ;AKKU NACH $0400 + X
A C013 CA                DEX          ;ZÄHLER VERMINDERN
A C014 10 EC             BPL $C002    ;SPRUNG, WENN POSITIV
Verzögerungsschleife
A C016 A2 FF             LDX #$FF     ;SCHLEIFENZÄHLER INIT.
A C018 20 B3 EE          JSR $EEB3    ;VERZÖGERUNGSROUTINE
A C01B CA                DEX          ;ZÄHLER DEKREMENTIEREN
A C01C D0 FA             BNE $C018    ;SPRUNG, WENN
                                      ;UNGLEICH NULL
A C016 4C 00 C0          JMP $C000    ;NEUSTART DES PROGRAMMS
```

Nachdem eine Zeile komplett invertiert/normalisiert wurde, wird in einer Schleife 255mal die Verzögerungsroutine $EEB3 aufgerufen.

Trotz dieser extrem langen Verzögerung ist das Blinken noch ein wenig "hektisch". Wenn Sie wollen, können Sie den Verzögerungsschleife in eine weitere Schleife mit Y als Zählregister "einpacken".

Ein Standardtrick

Die letzte Version des Programms "blinkende Bildschirmzeile" ist für unsere Begriffe bereits recht annehmbar. Assembler-"Profis" würden jedoch einen bestimmten Programmteil sofort ändern: das "Überspringen" der Subtraktions-Routine.

Erinnern wir uns: wenn die Zeile noch nicht invertiert ist, addiert der Additionsteil $80 und der folgende Subtraktionsteil wird mit einem JMP-Befehl übersprungen.

So seltsam es auf den ersten Blick erscheinen mag: der Drei-Byte-Befehle JMP kann durch einen kürzeren (zwei Bytes) Branch-Befehl ersetzt werden.

Addiert wird $80 zu einem Wert, der sich irgendwo zwischen $00 und $79 befindet (normales Zeichen). Das Ergebnis liegt daher zwischen $80 und $FF. Da wir genau wissen, daß es niemals den Wert $00 annimmt, können wir JMP durch BNE ersetzen und somit ein Byte einsparen. Die Bedingung "verzweige, wenn ungleich null" ist immer erfüllt!

Dieser Einsatz von Verzweigungen ist ein "Standardtrick" bei der Assembler-Programmierung. Ein JMP-Befehl kann immer dann durch einen Branch-Befehl ersetzt werden, wenn wir genau wissen, daß ein bestimmtes Flag immer gesetzt oder gelöscht ist. In diesem Fall ist der Branch-Befehl völlig gleichwertig, aus dem bedingten wird ein unbedingter Sprung.

```
            LDA $($0400)
            CMP #$01
            BNE $(ADRESSE1)
            LDX #$0A
            JMP $(ADRESSE2)
(ADRESSE1)  LDX #$0B
(ADRESSE2)  ...
            ...
```

```
            LDA $($0400)
            CMP #$01
            BNE $(ADRESSE1)
            LDX #$0A
            BNE $(ADRESSE2)
(ADRESSE1)  LDX #$0B
(ADRESSE2)  ...
            ...
```

Die Aufgabe dieses Programms: der Akkumulator wird mit dem Inhalt der Speicherzelle $0400 geladen. Besitzt $0400 den Wert $01, soll das X-Register mit $0A geladen werden, ansonsten mit $0B.

Im ersten Fall folgt dem Befehl LDX #$0A ein unbedingter Sprung mit JMP (linke Version), um das Laden des X-Registers mit $0B zu übergehen. Der JMP-Befehl kann (rechte Version) problemlos durch den kürzeren Befehl BNE ersetzt werden. Wir wissen, daß der LDX-Befehl das von BNE getestete Zero-Flag beeinflußt. Da der Wert $0A geladen wird, ist die Bedingung "verzweige, wenn ungleich null" mit Sicherheit erfüllt. In diesem Fall ist BNE ein ebenso unbedingter Sprung wie JMP.

Dieser Standardtrick wird in praktisch jedem Assembler-Programm eingesetzt und im Folgenden daher auch von mir verwendet. Nach kurzer Zeit erscheint Ihnen diese Methode wahr-

scheinlich nicht mehr als "Trick", sondern als eine für Sie völlig normale und empfehlenswerte Vorgehensweise.

```
(ADRESSE1) ...
           ...
           LDX #$FF
(ADRESSE2) ...
           ...
           DEX
           CPX #$0A
           BNE (ADRESSE2)
           JMP (ADRESSE1)
```

```
(ADRESSE1) ...
           ...
           LDX #$FF
(ADRESSE2) ...
           ...
           DEX
           CPX #$0A
           BNE (ADRESSE2)
           BEQ (ADRESSE1)
```

In diesem Programm soll ADRESSE1 den Programmanfang kennzeichnen und ADRESSE2 den ersten Befehl in einer Schleife mit X als Zählregister. Der Schleifenzähler X wird - beginnend mit dem Wert $FF - abwärts gezählt, dekrementiert.

Der Befehl CPX #$0A vergleicht den Inhalt des X-Registers mit $0A. Wenn X nicht $0A enthält, ergibt die bei Vergleichsbefehlen durchgeführte Subtraktion ein Ergebnis ungleich null (Beispiel: X=$FF => $FF - $0A =$F5) und die Bedingung BNE ("verzweige, wenn ungleich null") ist erfüllt, BNE verzweigt zum Schleifenanfang.

Erst wenn X bis $0A heruntergezählt ist, wird die Schleife verlassen. Der Vergleichsbefehl ergibt $00 als Ergebnis ($0A - $0A = $00) und die Sprungbedingung "verzweige, wenn ungleich null" ist nicht erfüllt. Der folgende JMP-Befehl wird ausgeführt und zum Programmanfang gesprungen.

Die rechte Version zeigt, daß JMP durch BEQ ersetzt werden kann. Wir wissen, daß das Ergebnis des Vergleichs gleich null sein muß, wenn dieser Befehl erreicht wird und daß daher BEQ ("verzweige, wenn gleich null") mit Sicherheit verzweigen wird.

Merken wir uns: ein JMP-Befehl kann immer dann durch einen Branch ersetzt werden, wenn wir mit Sicherheit wissen, daß eine bestimmte Bedingung auf alle Fälle erfüllt ist (Ergebnis gleich null, ungleich null, positiv, negativ...). Wir setzen in einem sol-

chen Fall den Branch-Befehl ein, der genau diese Bedingung testet (BEQ, BNE, BPL, BMI...) und sparen ein Byte gegenüber einem JMP.

2.16 Stapeloperationen

In den folgenden Kapiteln werden wir weitere Befehle und Adressierungsarten kennenlernen, mit denen die Assembler-Programmierung erheblich einfacher und vielseitiger wird.

Wie üblich folgt anschließend ein Kapitel mit Demoprogrammen, die den effektiven Einsatz aller bis dahin behandelten Befehle demonstrieren.

In diesem Abschnitt beschäftigen wir uns mit einem bisher vernachlässigten Speicherbereich unseres Rechners, dem "Stack". Der Stack oder "Stapel" ist ein spezieller Speicherbereich, der bei $0100 beginnt und bis $01FF reicht. Der Stapel umfaßt somit die Page (oder "Seite") 1 des Speichers, jene Seite, die der Zeropage unmittelbar folgt.

Der Stack ist nach dem sogenannten "LIFO"-Prinzip organisiert (LIFO=Last in, first out). Im Buch "Programmierung des 6502" von R.Zaks wird der Stack mit einem der häufig im Auto verwendeten "Münzspeicher" (Zehn-Pfennig-Stücke für Parkuhren) verglichen.

Bei der Datenablage auf dem Stack wird der betreffende Wert von oben auf den Münzspeicher "geschoben", alle darunterliegenden Münzen werden nach unten gedrückt. Diese Datenablage übernimmt der Befehl PHA ("push akku", "schiebe den Akku (auf den Stack)"), der den aktuellen Inhalt des Akkumulators zuoberst auf dem Stack ablegt.

Mit dem Befehl PLA ("pull akku", "ziehe den Akku (vom Stack)") wird die oberste "Münze" vom Stack entnommen und in den Akkumulator gebracht.

Da diese Art der Datenablage vom Prozessor verwaltet wird, ist der Stack hervorragend zur kurzfristigen Speicherung von Daten geeignet. Bisher mußten wir den Inhalt des Akkumulators in einer unbenutzten Speicherzelle zwischenspeichern.

```
LDA $0400 ;INHALT VON $0400 LESEN
STA $033C ;AKKUMULATOR 'RETTEN'
LDA #$FF  ;AKKU MIT $FF LADEN
...
...
LDA $033C ;URSPRÜNGL.AKKUINHALT ZURÜCKHOLEN
```

Mit den Befehlen PHA und PLA bietet sich die Zwischenspeicherung auf dem Stack an. Mit PHA wird der Akkumulatorinhalt als oberstes Element auf dem Stack abgelegt und - wenn der ursprüngliche Wert wieder benötigt wird - mit PLA zurückgeholt.

```
LDA $0400 ;INHALT VON $0400 LESEN
PHA       ;AKKU AUF STACK 'SCHIEBEN'
LDA #$FF  ;AKKU MIT $FF LADEN
...
...
PLA       ;OBERSTEN WERT VOM STACK 'ZIEHEN'
```

Der große Vorteil des Stacks liegt darin, daß wir uns nicht darum kümmern müssen, welche Speicherzellen unbenutzt sind und zur Zwischenspeicherung von Daten verwendet werden können.

Auf dem Stack können wir bis zu 255 Werte zwischenspeichern und später wieder zurückholen.

```
LDA #$01  ;AKKU MIT $01 LADEN
PHA       ;AUF STACK SCHIEBEN
LDA #$02  ;AKKU MIT $02 LADEN
PHA       ;AUF STACK SCHIEBEN
...
...
PLA       ;$02 VOM STACK ZIEHEN
```

```
STA $0400 ;UND NACH $0400
PLA       ;$01 VOM STACK ZIEHEN
STA $0400 ;UND NACH $0401
```

Beachten Sie bitte bei diesem Beispiel, daß der erste PLA-Befehl das zuletzt auf den Stack geschobene Element holt, der zweite Befehl PLA das vorletzte Element (First in, last out) und erinnern Sie sich an die sehr passende Analogie mit dem Münzspeicher.

Auf den Münzspeicher schieben Sie zuerst eine Ein- und dann eine Zwei-Pfennig-Münze. Die Zwei-Pfennig-Münze befindet sich nun über der Ein-Pfennig-Münze und wird beim "Ziehen" einer Münze zuerst geholt.

In der Praxis muß sehr oft nicht der Inhalt des Akkumulators, sondern der Inhalt eines der Indexregister "gerettet" werden. In diesem Fall kopieren wir den Registerinhalt in den Akkumulator und "pushen" diesen mit PHA.

Wenn wir den Registerwert wieder benötigen, "pullen" wir den abgelegten Wert mit PLA und übertragen ihn mit TAX oder TAY wieder in das betreffende Register.

```
LDX $0400 ;X MIT INHALT VON $0400 LADEN
TXA       ;X NACH AKKU KOPIEREN
PHA       ;AKKU ALS OBERSTEN WERT AUF STACK SCHIEBEN
...
...
PLA       ;OBERSTEN WERT VOM STACK ZIEHEN
TAX       ;AKKU NACH X KOPIEREN
```

Mit dieser Methode ist es selbstverständlich jederzeit möglich, auch beide Indexregister vorübergehend auf dem Stack zu speichern und, wenn benötigt, die ursprünglichen Werte zurückzuholen.

```
TXA ;X NACH AKKU
PHA ;AKKU ALS - VORLÄUFIG - OBERSTES ELEMENT AUF STACK
TYA ;Y NACH AKKU
```

```
PHA ;AKKU ALS OBERSTES ELEMENT AUF STACK
...
...
PLA ;OBERSTES STACK-ELEMENT HOLEN
TAY ;UND NACH Y
PLA ;NEUES OBERSTES ELEMENT HOLEN
TAX ;UND NACH X
```

Beachten Sie die Reihenfolge: Der Inhalt des X-Registers wird als oberstes Element auf den Stack gebracht, anschließend jedoch durch den folgenden PHA-Befehl (Y-Registerinhalt) "nach unten gedrückt".

Beim Holen beider Werte wird zuerst der ursprüngliche Inhalt des Y-Registers geholt, die darunterliegenden Werte rücken nach und anschließend wird mit dem zweiten PLA-Befehl der ursprüngliche X-Inhalt geholt, der sich nun an der "Stapelspitze" befindet.

Übrigens: die beiden Befehle TXA und PHA sind nicht länger als der "konventionelle" Befehl STX $033C, sogar kürzer. Beides sind Ein-Byte-Befehle, im Gegensatz zum Drei-Byte-Befehl STX $033C. Gleiches gilt für PLA/TAX im Vergleich zu LDX $033C.

Ab und zu werden wir in den folgenden Kapiteln zwei weitere Befehle benötigen, die mit dem Stack arbeiten, PHP und PLP.

PHP ("push processor") "schiebt" das Status-Register des Prozessors als oberstes Element auf den Stack. PLP ("pull processor") "zieht" das oberste Element vom Stack und überträgt es in das Status-Register.

Mit der Kombination dieser Befehle können wir jederzeit den aktuellen Flagzustand auf den Stack "retten" und bei Gelegenheit zurückholen.

```
          LDX #$FF
(ADRESSE) ...
          ...
```

```
DEX
PHP
LDA #$0A
PLP
BNE (ADRESSE)
```

In diesem - nicht unbedingt sehr sinnvollen - Beispiel wird nach der Dekrementierung des Zählregisters X der aktuelle Flagzustand - das Status-Register - auf den Stack "gerettet", da die Flags durch den Befehl LDA #$0A verändert werden.

Vor dem Testen des Zero-Flags wird der ursprüngliche Inhalt des Status-Registers mit PLP wieder vom Stack geholt, bevor der Test des Zero-Flags mit BNE erfolgt.

Beachten Sie bei Datenspeicherung auf dem Stack immer das LIFO-Prinzip (das zuletzt auf den Stack gebrachte Element wird als erstes geholt)!

```
LDA #$01
PHA       ;$01 'PUSHEN'
LDA #$02
PHA       ;$02 'PUSHEN'
LDA #$03
PHA       ;$03 'PUSHEN'
...
...
PLA       ;$03 'PULLEN'
STA $0400
PLA       ;$02 'PULLEN'
STA $0401
PLA       ;$01 'PULLEN'
STA $0400
```

Wie wir noch sehen werden, ist es zudem außerordentlich wichtig, daß zu jedem PHA auch ein PLA gehört, der das Element zu einem späteren Zeitpunkt wieder vom Stack holt!

Wenn Sie noch der Methode vorgehen: "Alles, was später eventuell gebraucht werden könnte, rette ich auf den Stack. Viel-

leicht hole ich diese Werte wieder irgendwann vom Stack, vielleicht auch nicht", "hängt" sich Ihr Programm früher oder später auf, das heißt, Sie müssen den Rechner ausschalten oder aber - wenn Sie sich inzwischen einen solchen angeschafft haben - den Reset-Schalter betätigen.

Die indirekte Adressierung

Wir kennen die implizite (TAX, PHA..), die unmittelbare (LDA #$FF..), die absolute (LDX $0400), die Zeropage- (LDA $F0) und die indizierte (LDA $0400,X) Adressierung.

Zwei Adressierungsarten fehlen uns noch, die "indirekte" und die "indirekt indizierte" Adressierung. Die Erläuterung dieser Adressierungsarten wurde bis zu diesem Zeitpunkt zurückgestellt, da beide sehr ungewöhnlich und eher zu verstehen sind, wenn bereits Assembler-Grundkenntnisse (die Sie inzwischen besitzen) vorhanden sind.

"Indirekte" Adressierung bedeutet, daß die zu verwendende Adresse nicht direkt (wie bei LDA $0400), sondern über einen Umweg angegeben wird.

Dem jeweiligen Befehl wird eine Adresse angegeben, die die eigentlich gewünschte Adresse enthält. Eine Analogie: wir benötigen eine Telefonnummer und rufen die Telefonauskunft an. Im Telefonbuch finden wir die Nummer der Auskunft. Die Nummer der Auskunft ist ein "Umweg", über den wir letzendlich die gewünschte Nummer erfahren.

Entsprechend verläuft die indirekte Adressierung. Dem jeweiligen Befehl geben wir die Adresse einer Speicherzelle an, die die eigentlich gewünschte Adresse enthält.

```
Zieladresse in $033C/$033D ablegen
      A C000 A9 0A            LDA #$0A      ;LOW-BYTE VON $C00A
      A C002 8D 3C 03         STA $033C     ;NACH $033C SCHREIBEN
      A C005 A9 C0            LDA #$C0      ;HIGH-BYTE VON $C00A
      A C007 8D 3D 03         STA $033D     ;NACH $033D SCHREIBEN
```

```
Hauptprogramm
      A C00A EE 20 D0          INC $D020     ;$D020 INKREMENTIEREN
      A C00D 6C 3C 03          JMP ($033C)   ;JMP INDIREKT !!!
```

Dieses Programm inkrementiert in einer Endlosschleife die Farbe des Bildschirmrahmens ($D020 = Rahmenfarbe beim C64).

Die ersten vier Befehle legen die Adresse $C00A in Form von Low-Byte/High-Byte in den Speicherzellen $033C (Inhalt $0A) und $033D (Inhalt $C0) ab. Anschließend wird die Rahmenfarbe inkrementiert.

Der folgende JMP-Befehl ist ein "indirekter Sprung". In Klammern wird dann die Adresse angegeben, an der die eigentliche Sprungadresse zu finden ist!

Das heißt: dieser JMP-Befehl springt nicht zur Adresse $033C, sondern zu jener Adresse, die in $033C und $033D in der Form Low-Byte/High-Byte abgelegt ist.

Mit dem vorbereitenden Teil wurde die Adresse $C00A abgelegt, die Adresse des Befehls INC $D020. Der JMP-Befehl findet ab der Adresse $033C daher die eigentliche Sprungadresse $C00A vor, zu der gesprungen wird.

In diesem Beispiel ist JMP($033C) somit von der Funktion her identisch mit dem direkten Sprungbefehl JMP $C00A.

Merken wir uns: dem indirekten Sprung JMP (...) wird ein sogenannter "Vektor" oder "Zeiger" angegeben, der die eigentliche Zieladresse enthält. Ein Vektor besteht aus zwei aufeinanderfolgenden Speicherzellen, die eine 16-Bit-Adresse - die eigentliche Sprungadresse - in der Form Low-Byte/High-Byte enthalten, dem "Adreßformat".

Die indirekte Adressierung ist nur mit dem JMP-Befehl möglich und wird uns nicht weiter interessieren. Diese Adressierungsart wurde erläutert, da sie die Grundlage für die eminent wichtige "indirekt-indizierte" Adressierung bildet, die im folgenden Abschnitt beschrieben wird.

Die indirekt-indizierte Adressierung

Die "indirekt-indizierte" Adressierung ist eine Kombination aus der soeben beschriebenen indirekten und der bereits bekannten indizierten Adressierung.

Sie existiert in zwei Varianten, der "Vorindizierung" und der "Nachindizierung". Wir werden uns fast ausschließlich mit der Nachindizierung beschäftigen, die wesentlich wichtiger ist als die Vorindizierung. Ein Beispiel:

```
LDA ($FB),Y
```

Die Klammer um die Zeropage-Adresse $FB zeigt an, daß es sich um einen Zeiger (Vektor) handelt, daß $FB und die folgende Speicherzelle $FC eine Adresse enthalten. Nehmen wir an, in $FB befindet sich der Wert $00 und in $FC der Wert $C0.

Speicherzellen	Inhalt
$FB	$00
$FC	$C0

$FB/$FC enthalten somit die 16-Bit-Adresse $C000 (Low-Byte $00/High-Byte $C0). Ähnlich wie bei der indizierten Adressierung wird zu dieser Adresse der Inhalt des Y-Registers addiert. Angenommen, Y enthält den Inhalt $04. Dann ergibt sich als endgültige Adresse $C004 ($C000 + $04 = $C004).

Der Befehl LDA ($FB),Y lädt den Inhalt von Speicherzelle $C004 in den Akkumulator, wenn $FB/$FC die Adresse $C000 und das Y-Register den Wert $04 enthalten.

Diese Variante der indirekt-indizierten Adressierung heißt nachindiziert, weil zuerst die indirekt angegebene Zieladresse gelesen und danach der Inhalt des Y-Registers addiert wird.

Wichtig: die in Klammern angegebene Adresse - der Zeiger - muß sich in der Zeropage befinden! Außerdem kann im Gegensatz zur indizierten Adressierung nur das Y-Register zur Indizierung eingesetzt werden, LDA ($FB),X ist nicht zulässig!

Mit dieser Adressierungsform können wir höchst elegant auf beliebige Speicherbereiche zugreifen, auf "Blöcke". Da sich der Zeiger jedoch in der Zeropage befinden muß, sind wir bei dieser Adressierungsart auf wenige Speicherzellen beschränkt.

Folgende Zeropageadressen sind unbenutzt und eignen sich zur Ablage von Zeigern:

```
$FB-$FE (4 Byte)
```

In der Praxis benötigt man niemals mehr als zwei Zeiger gleichzeitig. Einigen wir uns auf folgende Konvention: für einen Zeiger verwenden wir $FB/$FC, benötigen wir einen zweiten Zeiger, benutzen wir zusätzlich $FD/$FE.

Nun jedoch zum praktischen Einsatz dieser Adressierungsart. Erinnern Sie sich: mit der indizierten Adressierung konnten wir problemlos auf bis zu 256 aufeinanderfolgende Speicherzellen zugreifen, indem wir den Wert des Indexregisters mit einer Schleife veränderten.

```
          LDA #$01
          LDX #$00
(ADRESSE) STA $0400,X
          DEX
          BNE (ADRESSE)
```

Dieses Programm schreibt in die ersten 256 (Schleife inclusive null) Speicherzellen des Video-RAMs den Bildschirmcode des Zeichens A. Um eine Schleife 256mal (inclusive der null) zu durchlaufen, benötigten wir bislang den Befehl BCS und einen zusätzlichen Vergleichsbefehl.

```
          LDA #$01
          LDX #$FF
(ADRESSE) STA $0400,X
          DEX
          CPX #$FF
          BCS (ADRESSE)
```

Mit einem kleinen "Trick", der in der ersten Version des Programms angewendet wird, verringert sich dieser Aufwand. Das Zählregister X wird mit $00 initialisiert, nicht wie üblich mit $FF. Zuerst wird daher auf $0400 zugegriffen. Anschließend wird X dekrementiert und es erfolgt der bei Dekrementierungen bekannte "Unterlauf" von $00 zu $FF.

Im zweiten Durchlauf wird daher auf das "entgegengesetzte" Ende des Blocks zugegriffen, auf $04FF und in den folgenden Durchgängen wie üblich nacheinander auf $04FE, $04FD, $04FC,...,$0401.

Zuletzt wird $0401 behandelt, da die folgende Dekrementierung zum Ergebnis $00 führt ($01 - $00 = $00) und die Bedingung BNE ("verzweige, wenn ungleich null") nicht erfüllt ist.

Daß $0400 nicht mehr behandelt wird, ist jedoch nicht tragisch. Auf diese Speicherzelle wurde bereits beim ersten Durchlauf zugegriffen.

Mit diesem kleinen Trick können wir problemlos einen kompletten Block von 256 Zeichen behandeln, inclusive des problematischen "nullten" Bytes.

Um alle vier "Bildschirmseiten" mit dem Zeichen A zu füllen, benötigten wir bisher ein Programm wie das folgende.

```
          LDA #$01
          LDX #$00
(ADRESSE) STA $0400,X
          STA $0500,X
          STA $0600,X
          STA $0700,X
          DEX
          BNE (ADRESSE)
```

Diese Schleife behandelt zuerst die Speicherzellen $0400, $0500, $0600 und $0700, also die ersten Speicherzellen jeder Seite. Anschließend wird das X-Register dekrementiert und erhält den Wert $FF ($00 - $01 = $FF).

Im folgenden Durchgang werden daher $04FF, $05FF, $06FF und $07FF behandelt, die jeweils letzten Speicherzellen der vier Bereiche.

Nach erfolgter Dekrementierung des X-Registers enthält es den Wert $FE und die jeweils vorletzten Speicherzellen werden mit dem Code für A gefüllt ($04FE, $05FE, $06FE, $07FE).

Im letzten Durchgang hat X den Wert $01 und die Speicherzellen $0401, $0501, $0601 und $0701 werden behandelt.

Alle vier Seiten des Bildschirmspeichers ($0400-$07FF) sind mit dem Bildschirmcode des Zeichens A gefüllt worden.

Mit der indirekt-indizierten Adressierung wird das Programm nicht unbedingt kürzer, jedoch weitaus "eleganter".

```
Bildschirm mit A's füllen

Zeiger auf Anfang des Bildschirmspeichers nach $FB/$FC
      A C000 A0 00              LDA #$00      ;LOW-BYTE VON $0400
      A C002 85 FB              STA $FB       ;NACH $FB
      A C004 A0 04              LDA #$04      ;HIGH-BYTE VON $0400
      A C006 85 FC              STA $FC       ;NACH $FC
Schleifenzähler und Akkumulator initialisieren
      A C008 A9 01              LDA #$01      ;$01 = CODE FÜR 'A'
      A C00A A2 04              LDX #$04      ;X=ZÄHLER ÄUSSERE SCHLEIFE
      A C00C A0 00              LDY #$00      ;Y=ZÄHLER INNERE SCHLEIFE
Hauptprogramm
      A C00E 91 FB              STA ($FB),Y   ;'A' NACH ($FB),Y
      A C010 88                 DEY           ;ZÄHLER DEKREMENTIEREN
      A C011 D0 FB              BNE $C00E     ;SPRUNG, WENN
                                              ;UNGLEICH NULL
      A C013 E6 FC              INC $FC       ;SONST ZEIGER AUF
                                              ;NÄCHSTE PAGE
      A C015 CA                 DEX           ;ZÄHLER DEKREMENTIEREN
      A C016 D0 F4              BNE $C00C     ;SPRUNG, WENN
                                              ;UNGLEICH NULL
      A C018 00                 BRK           ;PROGRAMMENDE
```

Im vorbereitenden Teil wird die Startadresse des Bildschirms ($0400) in $FB/$FC in der Form Low-Byte/High-Byte abgelegt, der "Zeiger" auf den Bildschirm erzeugt. Zur Vorbereitung gehört ebenfalls das Laden des Akkumulators mit $01, dem Bildschirmcode des Zeichens A, und die Initialisierung der Schleifenzähler X und Y mit $04 beziehungsweise $00.

Der Hauptteil besteht aus zwei ineinandergeschachtelten Schleifen. Die innere Schleife mit Y als Zählregister ist für die Behandlung eines Blocks mit 256 Zeichen zuständig.

Betrachten wir den ersten Durchgang: Y enthält den Wert $00. Der Inhalt des Akkumulators ($01=A) wird in die Speicherzelle $0400 geschrieben (der Zeiger $FB/$FC weist auf $0400; zu $0400 wird Y=$00 addiert).

Y wird dekrementiert (DEY) und hat nun den Wert $FF, das heißt BNE verzweigt zu Adresse $C00E. Der Befehl STA ($FB),Y bezieht sich nun auf die Speicherzelle $04FF (Zeiger in $FB/$FC weist auf $0400; Y gleich $FF => $04FF), die letzte Speicherzelle der ersten Bildschirmseite.

Y wird erneut dekrementiert (=> Y=$FE) und zum Beginn der inneren Schleife verzweigt. Der Bildschirmcode des Zeichens A wird in die Speicherzelle $04FE geschrieben und so weiter.

Im letzten Durchgang der inneren Schleife besitzt Y den Wert $01, der Befehl STA ($FB),Y bezieht sich daher auf $0401, die vorletzte Zelle von Bildschirmseite 1 ($0400 wurde bereits - im ersten Durchgang - behandelt).

DEY führt nun zum Ergebnis $00 und BNE verzweigt nicht mehr, die innere Schleife wird verlassen. Entscheidend an diesem Programm ist der Befehl INC $FC. $FC ist das High-Byte des Zeigers, der momentan auf die Adresse $0400 weist. Wird das High-Byte dieses Zeigers inkrementiert, weist er auf den Beginn der zweiten Bildschirmseite, also auf den nächsten zu behandelnden Block. Aus dem Zeiger $00/$04 (=$0400 in der Form Low/High) in den Speicherzellen $FB/$FC wird durch die

Inkrementierung von $FC der neue Zeiger $00/$05 (=$0500 in der Form Low/High).

Das X-Register wird dekrementiert und - da X ungleich null - mit BNE zu $C00C verzweigt. Nachdem das Y-Register wieder mit $00 intialisiert wurde, wird die innere Schleife erneut 256mal durchlaufen. Diesmal wird jedoch statt auf den Bereich $0400-$04FF auf $0500-$05FF zugegriffen.

Die äußere Schleife wird insgesamt viermal durchlaufen (LDX #$04), um alle vier Seiten des Bildschirms zu behandeln.

Sie sehen, mit der indirekt-indizierten Adressierung kann sehr elegant auf beliebig große Blöcke zugegriffen werden. Nachdem ein Block mit 256 Byte behandelt wurde, wird das High-Byte des Zeigers inkrementiert und in der inneren Schleife der folgende 256-Byte-Block bearbeitet.

Noch eine Bemerkung: am Ende der äußeren Schleife wird zu $C00C verzweigt, um durch erneute Initialisierung des Zählregisters Y mit dem Startwert $00 die nächsten 256 Durchgänge der inneren Schleife vorzubereiten.

Wenn Sie sich das Programm näher anschauen, stellen Sie jedoch fest, daß Y nach dem Verlassen der inneren Schleife sowieso den Wert $00 besitzt und eine Initialisierung mit diesem Wert daher überflüssig ist. Anstatt zu $C00C kann daher gleich zu $C00E (dem eigentlichen Beginn der inneren Schleife) verzweigt werden.

Ich muß zugeben, daß das Demoprogramm zur Behandlung des Bildschirms mit der indizierter Adressierung kürzer ist, da das Einrichten des Zeigers in $FB/$FC entfällt.

Bedenken Sie jedoch, wie groß der Aufwand mit indizierter Adressierung wird, wenn umfangreiche Speicherbereiche bearbeitet werden, zum Beispiel der Bereich $8000-$8FFF (vier Kilobyte).

```
Indizierte Adressierung
          LDX #$00
(ADRESSE) STA $8000,X
          STA $8100,X
          STA $8200,X
          STA $8300,X
          STA $8400,X
          STA $8500,X
          STA $8600,X
          STA $8700,X
          STA $8800,X
          STA $8900,X
          STA $8A00,X
          STA $8B00,X
          STA $8C00,X
          STA $8D00,X
          STA $8E00,X
          STA $8F00,X
          DEX
          BNE (ADRESSE)
```

```
Indirekt-indizierte Adressierung
          LDA #$00
          STA $FB
          LDA #$80
          STA $FC
          LDX #$10
          LDY #$00
(ADRESSE) STA ($FB),Y
          DEY
          BNE (ADRESSE)
          INC $FC
          DEX
          BNE (ADRESSE)
```

Bei der indirekt-indizierten Adressierung ist es gleichgültig, ob nur ein Block mit 256 Bytes bearbeitet wird, oder aber beliebig viele Blöcke. Das X-Reister wird mit der jeweiligen Blockanzahl geladen, der Rest des Programms bleibt unverändert. Der Umgang mit der indirekt-indizierten Adressierung:

1. In der Zeropage wird ein Zeiger auf den Beginn des Speicherbereichs in zwei aufeinanderfolgenden Speicherzellen abgelegt (Low-Byte, dann High-Byte).
2. Der Zähler der äußeren Schleife wird mit der Anzahl der zu bearbeitenden 256-Byte-Blöcke geladen.
3. Der Zähler der inneren Schleife wird initialisiert.
4. Die innere Schleife wird durchlaufen und greift mit indirekt-indizierter Adressierung auf die erste Seite des betreffenden Speicherbereichs zu.
5. Wenn die innere Schleife verlassen wird (ein Block ist komplett bearbeitet), wird das High-Byte des Zeigers inkrementiert. Der Zeiger weist nun genau 256 Byte weiter, also auf den Beginn des nächsten Blocks.

6. Der "Blockzähler" X wird dekrementiert und - wenn ungleich null - zum Beginn der inneren Schleife verzweigt. Die nächsten 256 Byte werden bearbeitet.

Übungsprogramme zur indirekt-indizierten Adressierung

Sie kennen nun die flexibelste Form der Adressierung, die unser Rechner besitzt. Diese Adressierungsart ist dermaßen vielseitig verwendbar, daß wohl kaum ein größeres Assembler-Programm ohne die indirekt-indizierte Adressierung auskommt.

Außer LDA können folgende Befehle mit der beschriebenen Adressierung verwendet werden: ADC, CMP, SBC und STA.

Da diese Adressierungsart so eminent wichtig ist, stelle ich Ihnen in diesem Abschnitt einige Übungsprogramme vor, die außer STA auch LDA und CMP mit indirekt-indizierter Adressierung verwenden.

Selektierte Zeichen invertieren

Angenommen, Sie wollen ein bestimmtes Zeichen überall, wo es auf dem Bildschirm vorhanden ist, invertieren. Ohne die indirekt-indizierte Adressierung wird Ihnen dieses Problem einige Kopfschmerzen bereiten, mit dieser Adressierungsart ist es dagegen sehr einfach zu lösen.

```
Zeiger einrichten
      A C000 A0 00          LDA #$00     ;LOW-BYTE VON $0400
      A C002 85 FB          STA $FB      ;NACH $FB
      A C004 A0 04          LDA #$04     ;HIGH-BYTE VON $0400
      A C006 85 FC          STA $FC      ;NACH $FC
Zähler initialisieren
      A C008 A2 04          LDX #$04     ;ZÄHLER ÄUSSERE SCHLEIFE
                                         ;INITIALISIEREN
      A C00A A0 00          LDY #$00     ;ZÄHLER INNERE SCHLEIFE
                                         ;INITIALISIEREN
```

```
Schleifenanfang
      A C00C B1 FB          LDA ($FB),Y  ;AKKU MIT ($FB),Y LADEN
      A C00E C9 20          CMP #$20     ;MIT $20(=SPACE)
                                         ;VERGLEICHEN
      A C010 D0 04          BNE $C016    ;SPRUNG, WENN UNGLEICH
      A C012 A9 A0          LDA #$A0     ;AKKU MIT $A0(=SPACE
                                         ;INVERS) LADEN
      A C014 91 FB          STA ($FB),Y  ;UND NACH ($FB),Y
                                         ;SCHREIBEN
      A C016 88             DEY          ;ZÄHLER INNEN
                                         ;DEKREMENTIEREN
      A C017 D0 F3          BNE $C00C    ;SPRUNG, WENN
                                         ;UNGLEICH NULL
      A C019 E6 FC          INC $FC      ;SONST ZEIGER AUF
                                         ;NÄCHSTE SEITE
      A C01B CA             DEX          ;ZÄHLER AUSSEN
                                         ;DEKREMENTIEREN
      A C01C D0 EE          BNE $C00C    ;SPRUNG, WENN
                                         ;UNGLEICH NULL
      A C01E 00             BRK          ;PROGRAMMENDE
```

Der Aufbau unterscheidet sich kaum vom vorhergehenden Programm. Der einzige prinzipielle Unterschied besteht darin, daß der Branch-Befehl am Ende der äußeren Schleife nicht zur - überflüssigen - Neuinitialisierung des Y-Registers, sondern sofort zum Beginn der inneren Schleife führt.

Wirkliche Unterschiede ergeben sich in der inneren Schleife. Diese Schleife besitzt die Aufgabe, jede Speicherzelle des Bildschirms auf den Inhalt $20 (Leerzeichen oder "Space") zu überprüfen.

Mit LDA ($FB),Y wird indirekt-indiziert auf die Speicherzellen zugegriffen. Wenn der Inhalt der untersuchten Zelle gleich $20 ist, wird der folgende Branch-Befehl (BNE $C016) nicht ausgeführt. Der Akkumulator wird mit $A0 geladen ($A0 = $20 + $80 = inverses Leerzeichen) und dieser Bildschirmcode ebenfalls indirekt-indiziert in die betreffende Speicherzelle zurückgeschrieben.

Auf diese Weise werden alle vier Bildschirmseiten behandelt (LDX #$04) und alle Leerzeichen invertiert.

Welche Zeichen invertiert werden, bestimmen Sie. Wollen Sie zum Beispiel alle A's invertieren, ersetzen Sie CMP #$20 durch CMP #$01 und LDA #$80 durch LDA #$81 ($01=Bildschirmcode für inverses A).

Durch Variieren dieser beiden Befehle können Sie ein beliebiges Zeichen auf dem gesamten Bildschirm durch ein ebenfalls beliebiges anderes Zeichen ersetzen (mit CMP #$20 und LDA #$01 werden alle Leerzeichen durch A's ersetzt).

2.17 Windowing

Jedes moderne Programm arbeitet heutzutage mit "Windowing", das heißt mit "Fenstern". Darunter ist zu verstehen, daß in einem beliebigen Bildschirmbereich zum Beispiel ein Hilfstext oder ein Menü "eingeblendet" und nach getaner Arbeit wieder "ausgeblendet" wird. Nach dem "Ausblenden" ist der ursprüngliche Bildschirmzustand unverändert widerhergestellt.

Dieses Windowing kann selbstverständlich auch mit einem BASIC-Programm durchgeführt werden. Über den aktuellen Bildschirminhalt wird mit PRINT-Befehlen ein Hilfstext gelegt.

Wird der Hilfstext nicht mehr benötigt, wird der Bildschirm gelöscht und mit einer weiteren Anzahl von PRINT-Befehlen der urspüngliche Inhalt (Text, Adresse etc.) wieder auf den Bildschirm geschrieben.

Mit blitzschnellem "Ein-" beziehungsweise "Ausblenden" hat diese Vorgehensweise allerdings nichts gemeinsam. Der ursprüngliche Bildschirminhalt wird mit PRINT-Befehlen sehr gemütlich Zeile für Zeile wiederhergestellt.

Mit der indirekt-indizierten Adressierung sind wir in der Lage, echtes Windowing in Assembler zu verwirklichen. Wir "retten" den kompletten Bildschirminhalt (vier Seiten a 256 Zeichen) in

einen freien Speicherbereich und kopieren ihn auf Kommando wieder zurück. Für das folgende Programm benötigen wir zwei Zeiger.

Der erste Zeiger weist auf die Startadresse des Bildschirmspeichers $0400, der zweite Zeiger auf die erste Speicherzelle des freien Bereichs.

Beim C64 verwenden wir den Bereich $C400-$CFFF, der sich hinter dem eingentlichen Programm befindet.

Wir verwenden ein Unterprogramm, das mit RTS endet und von BASIC aus mit dem SYS-Befehl aufgerufen wird. Das Unterprogramm besitzt zwei verschiedene "Einsprungadressen", die über die jeweilige Funktion entscheiden (Bildschirm retten/geretteten Inhalt zurückkopieren).

```
Version 1
Zeiger einrichten (Einsprung 1)
      A C000 A9 00            LDA #$00     ;LOW-BYTE VON $0400
      A C002 85 FB            STA $FB      ;NACH $FB
      A C004 A9 04            LDA #$04     ;HIGH-BYTE VON $0400
      A C006 85 FC            STA $FC      ;NACH $FC
      A C008 A9 00            LDA #$00     ;LOW-BYTE VON $C400
      A C00A 85 FD            STA $FD      ;NACH $FD
      A C00C A9 C4            LDA #$C4     ;HIGH-BYTE VON $C400
      A C00E 85 FE            STA $FE      ;NACH $FE
      A C010 4C 23 C0         JMP $C023    ;FOLGENDEN TEIL
Zeiger einrichten (Einsprung 2)            ;ÜBERSPRINGEN
      A C013 A9 00            LDA #$00     ;LOW-BYTE VON $C400
      A C015 85 FB            STA $FB      ;NACH $FB
      A C017 A9 C4            LDA #$C4     ;HIGH-BYTE VON $C400
      A C019 85 FC            STA $FC      ;NACH $FC
      A C01B A9 00            LDA #$00     ;LOW-BYTE VON $0400
      A C01D 85 FD            STA $FD      ;NACH $FD
      A C01F A9 04            LDA #$04     ;HIGH-BYTE VON $0400
      A C021 85 FE            STA $FE      ;NACH $FE
Bereich kopieren
      A C023 A2 04            LDX #$04     ;BLOCKZÄHLER: 4 BLÖCKE
                                           ;KOPIEREN
```

```
A C025 A0 00        LDY #$00     ;Y INITIALISIEREN
A C027 B1 FB        LDA ($FB),Y  ;ZEICHEN LESEN
A C029 91 FD        STA ($FD),Y  ;ZEICHEN KOPIEREN
A C02B 88           DEY          ;ZÄHLER INNEN
                                 ;DEKREMENTIEREN
A C02C D0 F9        BNE $C027    ;KOMPLETTEN BLOCK KOPIERT?
A C02E E6 FC        INC $FC      ;HIGH-BYTE VON ZEIGER 1
                                 ;INKREMENTIEREN
A C030 E6 FE        INC $FE      ;HIGH-BYTE VON ZEIGER 2
                                 ;INKREMENTIEREN
A C032 CA           DEX          ;BLOCKZÄHLER
                                 ;DEKREMENTIEREN
A C033 D0 F2        BNE $C027    ;ALLE BLÖCKE KOPIERT?
A C035 60           RTS          ;ZURÜCK NACH BASIC!
```

Der umfangreichste Teil dieses Programms besteht aus dem Einrichten der beiden Zeiger. Der Einsprung 1 (SYS 49152) richtet in $FB/$FC einen Zeiger auf die Startadresse des Bildschirmspeichers und in $FD/$FE einen zweiten Zeiger auf den freien Speicherbereich $C400 ein. Wenn die beiden Zeiger eingerichtet sind, wird der nun folgende Programmteil übersprungen (JMP $C023).

An der zweiten Einsprungadresse $C013 (SYS 49171) werden ebenfalls zwei Zeiger eingerichtet, jedoch hier mit vertauschten Adressen ($C400 in $FB/$FC und $0400 in $FD/$FE).

Zu Beginn des Hauptprogramms erfolgt die bereits bekannte Initialisierung der Zählregister. Der "Blockzähler" X wird mit dem Wert $04 geladen, da auf insgesamt vier Blöcke mit je 256 Byte zugegriffen werden soll. Das Y-Register wird mit $00 initialisiert und die innere Schleife beginnt.

Betrachten wir zuerst den Fall, daß der Benutzer das Programm mit SYS 49152 aufrief. Der Einsprung an Adresse $C000 führte zur Einrichtung folgender Zeiger:

```
$FB/$FC = Zeiger auf $0400
$FD/$FE = Zeiger auf $C400
```

Im ersten Durchgang lädt der Befehl LDA ($FB),Y den Akkumulator mit dem Inhalt der Speicherzelle $0400 (Zeigerinhalt + Y-Register => $0400 + $00). Der Inhalt dieser Speicherzelle wird mit STA ($FD),Y nach $C400 kopiert (Zeigerinhalt + Y-Register => $C400 + $00).

Das Y-Register wird dekrementiert (=> Y=$FF) und - da das Ergebnis $FF die Bedingung BNE $C027 erfüllt - die Schleife erneut durchlaufen.

Aufgrund des Y-Wertes $FF wird nun jedoch auf das entgegengesetzte Blockende zugegriffen. Der Akkumulator wird mit dem Inhalt der Speicherzelle $04FF ($0400 + $FF) geladen und nach $C4FF ($C400 + $FF) kopiert.

In den folgenden Durchgängen wird Y dekrementiert und der Reihe nach $04FE nach $C4FE, dann $04FD nach $C4FD und zuletzt $0401 nach $C401 kopiert. Der erste 256-Byte-Block ($0400-$04FF) wurde vollständig nach $C400-$C4FF kopiert.

Die High-Bytes beider Zeiger werden nun inkrementiert und weisen in den folgenden Durchgängen auf die Startadresse des nächsten Blocks ($0500 und $C500). Die nächsten 256 Durchgänge der inneren Schleife kopieren daher den Inhalt des Blocks $0500-$05FF nach $C500-$C5FF.

Der Vorgang wiederholt sich insgesamt viermal. Dann besitzt das X-Register - das nach jeder Blockübertragung dekrementiert wird - den Inhalt $00 und die Bedingung BNE $C027 ist nicht mehr erfüllt.

Der gesamte Bildschirmspeicher wurde kopiert und mit RTS erfolgt die Rückkehr nach BASIC.

Analog verläuft die Wiederherstellung des ursprünglichen Bildschirminhalts. Das Programm wird von BASIC aus mit SYS 49171 (=$C013) aufgerufen und startet vom zweiten Einsprungpunkt aus. Die Zeiger werden nun genau umgekehrt eingerichtet:

```
$FB/$FC = Zeiger auf $C400
$FD/$FE = Zeiger auf $0400
```

Der Kopiervorgang verläuft daher von $C400-$C7FF nach $0400-$07FF. Der zuvor "gerettete" Bildschirminhalt wird in den Bildschirmspeicher zurückkopiert.

Bevor ich Ihnen ein kleines BASIC-Programm vorstelle, das die praktische Anwendung dieses Assembler-Programms demonstriert, zeige ich Ihnen, wie dieses Programm weitaus kürzer und eleganter geschrieben wird.

1. Die Befehlsfolge:

```
LDA #$00
STA $FB
LDA #$04
STA $FC
LDA #$00
STA $FD
...
...
```

   kann erheblich verkürzt werden. Sowohl nach $FB als auch nach $FC soll der Wert $00 geschrieben werden. Es genügt vollkommen, den Akkumulator einmal mit $00 zu laden und dann mit zwei aufeinanderfolgenden Befehlen in beide Speicherzellen zu übertragen. Gleiches gilt für die Zeigereinrichtung am zweiten Einsprungpunkt.
2. Unabhängig vom Einsprungpunkt (1 oder 2) wird auf alle Fälle in $FB und in $FD das Low-Byte $00 übertragen. Es bietet sich daher an, nur die Speicherung der High-Bytes vom Einsprungpunkt abhängig zu machen und erst nach der "Zusammenführung" der Einsprungpunkte die identischen Low-Bytes in $FB und $FD zu speichern.
3. Beim Einsprung an Adresse $C000 wird nach dem Einrichten der Zeiger der folgende Programmteil mit JMP $C023 übersprungen. Der Drei-Byte-Befehl JMP kann durch den Zwei-Byte-Befehl BNE ersetzt werden. Der letzte Befehl, der das Zero-Flag beeinflußte, das BNE te-

stet, war LDA #$90 (Ergebnis ungleich null). Wir wissen daher mit absoluter Sicherheit, daß die Bedingung BNE ("verzweige, wenn ungleich null") erfüllt ist und der folgende Teil mit diesem Branch-Befehl übersprungen wird.

```
Version 2
Zeiger-High-Bytes einrichten (Einsprung 1)
      A C000 A9 04          LDA #$04      ;HIGH-BYTE VON $0400
      A C002 85 FC          STA $FC       ;NACH $FC
      A C004 A9 C4          LDA #$C4      ;HIGH-BYTE VON $C400
      A C006 85 FE          STA $FE       ;NACH $FE
      A C008 D0 08          BNE $C012     ;FOLGENDEN TEIL
                                          ;ÜBERSPRINGEN
Zeiger-High-Bytes einrichten (Einsprung 2)
      A C00A A9 C4          LDA #$C4      ;HIGH-BYTE VON $C400
      A C00C 85 FC          STA $FC       ;NACH $FC
      A C00E A9 04          LDA #$04      ;HIGH-BYTE VON $0400
      A C010 85 FE          STA $FE       ;NACH $FE
Gemeinsame Low-Bytes einrichten
      A C012 A9 00          LDA #$00      ;LOW-BYTE $00
      A C014 85 FB          STA $FB       ;NACH $FB
      A C016 85 FD          STA $FD       ;UND NACH $FD
Bereich kopieren
      A C018 A2 04          LDX #$04      ;4 BLÖCKE KOPIEREN
      A C01A A0 00          LDY #$00      ;ZÄHLER INNEN
                                          ;INITIALISIEREN
      A C01C B1 FB          LDA ($FB),Y   ;ZEICHEN LESEN
      A C01E 91 FD          STA ($FD),Y   ;ZEICHEN KOPIEREN
      A C020 88             DEY           ;ZÄHLER INNEN
                                          ;DEKREMENTIEREN
      A C021 D0 F9          BNE $C01C     ;KOMPLETTEN BLOCK KOPIERT?
      A C023 E6 FC          INC $FC       ;HIGH-BYTE ZEIGER 1
                                          ;INKREMENTIEREN
      A C025 E6 FE          INC $FE       ;HIGH-BYTE ZEIGER 2
                                          ;INKREMENTIEREN
      A C027 CA             DEX           ;BLOCKZÄHLER
                                          ;DEKREMENTIEREN
      A C028 D0 F2          BNE $C01C     ;ALLE BLÖCKE KOPIERT?
      A C02A 60             RTS           ;ZURÜCK NACH BASIC
```

Dieses Programm ist um einiges kürzer als die erste Version. Der Einsprung für die zweite Programmfunktion (Bildschirm wiederherstellen) ändert sich allerdings ($C00A). Dieser Programmteil wird nun mit SYS 49162 von BASIC aus aufgerufen.

Ein Demoprogramm:

```
100 PRINT CHR$(147):REM BILDSCHIRM LOESCHEN
110 FOR I=1 TO 20
120 : PRINT "TEST DER WINDOWING-ROUTINE"
130 NEXT
140 :
150 SYS 49152:REM BILDSCHIRM RETTEN
160 PRINT CHR$(147):PRINT "BITTE TASTE BETAETIGEN"
170 GET A$:IF A$="" THEN 170:REM AUF TASTE WARTEN
180 SYS 49162:REM BILDSCHIRM ZURUECKHOLEN
```

Das Demoprogramm schreibt zwanzigmal den String "TEST DER WINDOWING-ROUTINE" auf den Bildschirm, ruft die Routine zum "Retten" auf, löscht den Bildschirm und wartet auf eine Taste.

Anschließend wird die Routine zum "Zurückholen" des Bildschirminhalts aufgerufen und der alte Zustand wiederhergestellt.

Wie Sie sehen, kann man bereits mit relativ kleinen Assembler-Programmen äußerst nützliche und vielseitig anwendbare Routinen erstellen.

Zugriff auf beliebig große Speicherbereiche

In den vorangegangenen Beispielen wurden jeweils komplette 256-Byte-Blöcke mit der indirekt-indizierten Adressierung behandelt. Es ist jedoch mit ein wenig mehr Aufwand problemlos möglich, auch "krumme" Blocklängen zu bearbeiten (zum Beispiel 258 Byte = ein 256-Byte-Block plus die ersten beiden Byte des folgenden Blocks).

Das nächste Übungsprogramm invertiert die ersten zehn Zeilen des Bildschirms, behandelt also 400 Byte (ein 256-Byte-Block plus 144 (=$90) Byte des folgenden Blocks).

```
Vorbereitung
      A C000 A9 00      LDA #$00      ;LOW-BYTE VON $0400
      A C002 85 FB      STA $FB       ;NACH $FB
      A C004 A9 04      LDA #$04      ;HIGH-BYTE VON $0400
      A C006 85 FC      STA $FC       ;NACH $FC
      A C008 A2 01      LDX #$01      ;1 KOMPLETTER BLOCK
      A C00A A0 FF      LDY #$FF      ;Y INITIALISIEREN
Innere Schleife
      A C00C B1 FB      LDA ($FB),Y   ;AKKU MIT ($FB),Y LADEN
      A C00E 18         CLC           ;ADDITION VORBEREITEN
      A C00F 69 80      ADC #$80      ;$80 ADDIEREN
      A C011 91 FB      STA ($FB),Y   ;UND NACH ($FB),Y ZURÜCK
      A C013 88         DEY           ;ZÄHLER DEKREMENTIEREN
      A C014 C0 FF      CPY #$FF      ;UND MIT $FF VERGLEICHEN
      A C016 D0 F4      BNE $C00C     ;SPRUNG, WENN UNGLEICH
Äußere Schleife
      A C018 E6 FC      INC $FC       ;HIGH-BYTE DES ZEIGERS
                                      INKREMENTIEREN
      A C01A CA         DEX           ;BLOCKZÄHLER VERRINGERN
      A C01B 30 06      BMI $C023     ;SPRUNG, WENN
                                      ;NEGATIV ($FF!)
      A C01D D0 ED      BNE $C00C     ;NOCH EINEN
                                      ;BLOCK BEARBEITEN
      A C01F A0 8F      LDY #$8F      ;Y MIT RESTBYTES
                                      ;INITIALISIEREN
      A C021 D0 E9      BNE $C00C     ;UND ZUR INNEREN SCHLEIFE
      A C023 00         BRK           ;FERTIG
```

Daß die Schleifenführung in diesem Programm recht komplex ist, verdanken wir dem "Blockrest" von 144 Bytes, der nach Bearbeitung eines kompletten Blocks zu behandeln ist.

Zuerst wird wie üblich in $FB/$FC ein Zeiger auf die Startadresse des Bildschirmspeichers ($0400) eingerichtet und das X-Register mit der Anzahl "vollständiger" Blöcke (mit je 256 Byte) geladen.

Das Y-Register wird nicht wie gewohnt mit $00, sondern mit $FF initialisiert. Warum, sehen wir bei der Behandlung des "Restes".

In der inneren Schleife wird nun der Reihe nach auf die Speicherzellen $04FF-$0400 zugegriffen (Y wird dekrementiert). Zu den jeweiligen Bildschirmcodes wird $80 addiert und damit der Code "invertiert", bevor er in die betreffende Speicherzelle zurückgeschrieben wird.

Interessant ist nun das Schleifenende. Y wird mit $FF verglichen. Im ersten Durchgang besitzt Y diesen Wert und $04FF wird behandelt. Vor dem Vergleich wird Y jedoch dekrementiert (=> Y=$FE) und BNE verzweigt zum Schleifenanfang, ebenso wie in den folgenden Durchgängen.

Im letzten Durchgang besitzt Y den Wert $00. Die Speicherzelle $0400 wird "invertiert", Y wird dekrementiert (=> Y=$FF) und ist somit identisch mit dem Vergleichswert $FF. BNE verzweigt nicht mehr, die innere Schleife wird verlassen.

Wie üblich wird nun das High-Byte des Zeigers erhöht, der anschließend auf den folgenden Block weist (auf $0500). Der Blockzähler X wird dekrementiert.

Der Branch-Befehl BMI interessiert uns momentan noch nicht. Da X den Wert $00 besitzt, wird er jedenfalls nicht ausgeführt (X ist - noch - positiv).

BNE überprüft, ob bereits alle - kompletten - Blöcke behandelt wurden. Da nur ein Block zu bearbeiten war (LDX #$01), ist dies der Fall. X besitzt momentan den Wert $00 und BNE verzweigt nicht.

Nun müssen die restlichen 144 (=$90) Byte des folgenden Blocks behandelt werden. Y wird mit $8F geladen und erneut zum Beginn der inneren Schleife gesprungen. Diese wird wiederum durchlaufen, bis Y den Wert $FF annimmt. Behandelt wird somit der restliche Bereich von $058F bis $0500 ($90 Byte).

Wenn die innere Schleife verlassen wird, erfolgt erneut die Inkrementierung des Zeiger-High-Bytes und die Dekrementierung des Blockzählers X. Das X-Register besaß zuvor den Wert $00 und nimmt nun den negativen Wert $FF an.

Die Bedingung BMI ist erfüllt und es wird zum Programmende (BRK) verzweigt.

Ich versprach Ihnen, die Erklärung für den Vergleich CPY #$FF nachzuliefern. Betrachten wird den Fall, daß Y wie üblich mit $00 initialisiert wird und die Verzweigung ohne zusätzlichen Vergleichsbefehl mit BNE erfolgt.

```
(ADRESSE) LDA ($FB),Y
          ...
          ...
          DEY
          BNE (ADRESSE)
          ...
          ...
          LDY #$8F
          BNE (ADRESSE)
```

Mit dieser Schleife werden alle vollständigen Blöcke korrekt behandelt, jedoch nicht der "Rest". Nachdem Y mit $8F geladen und zu (ADRESSE) verzweigt wurde, werden der Reihe nach die Speicherzellen $058F-$0501 behandelt.

Wenn Y den Wert $01 annimmt, bewirkt die folgende Dekrementierung (DEY), daß die Bedingung BNE nicht erfüllt ist, die Schleife wird verlassen.

Das erste Byte des nur teilweise zu behandelnden letzten Blocks - die Speicherzelle $0500 - wird somit ausgelassen.

Mit dem vorliegenden Programm besitzen Sie eine allgemeine Routine zur Behandlung beliebiger Speicherbereiche. Voraussetzung zum Einsatz dieser Routine ist jedoch, daß die betreffenden Blöcke mit einer rückwärts zählenden Schleife behandelt werden können, was im nächsten Programm nicht der Fall ist.

Fehlermeldungen ausgeben

Das folgende Demoprogramm gibt alle Fehlermeldungen des BASIC-Interpreters auf dem Bildschirm aus. Diese Fehlermeldungen befinden sich im ROM (nicht beschreibbarer, nur lesbarer Speicher), und zwar von $A19E-$A327. Beim C64 müssen 393 Bytes bearbeitet werden, also ein kompletter 256-Byte-Block ($A19E-$A29D) und die ersten 137 (=$89) Zeichen des folgenden Blocks ($A29E-$A327).

Die Fehlermeldungen sind auf eine ganz spezielle Art und Weise abgelegt. Um das Ende einer Fehlermeldung zu erkennen, ist beim letzten Zeichen Bit 7 gesetzt. Da der Code dieses letzten Zeichens daher größer als $79 ist, kann das Ende einer Meldung nach dem Lesen der einzelnen Zeichen mit LDA ($FB),Y durch BMI oder BPL erkannt werden.

Die Meldung selbst ist im ASCII-Code gespeichert und wird von uns mit BSOUT Zeichen für Zeichen ausgegeben.

Da dieses Programm um einiges komplexer ist als alle bisherigen Demoprogramme, erläutere ich zuerst den prinzipiellen Ablauf.

1. In $FC/FB wird ein Zeiger auf die Anfangsadresse $A19E eingerichtet. Zur Ausgabe der Fehlermeldungen muß eine aufwärts zählende Schleife verwendet werden, sonst wird "TOO MANY FILES" in der Form "SELIF YNAM OOT" ausgegeben.
   Diese aufwärts zählende Schleife stellt bei der Behandlung des Restes ein Problem dar. Bei welchem Byte die Behandlung eines Blockes endet, wird der inneren Schleife mit der Speicherzelle $033C mitgeteilt, die zu Beginn den Wert $00 erhält (kompletten Block behandeln).
   Den Rest der Vorbereitung bildet wie üblich das Laden des X-Registers mit der Blockanzahl ($01) und die Initialisierung des Y-Registers mit $00.
2. In der inneren Schleife wird Zeichen für Zeichen ab jener Position gelesen, auf die der Zeiger in $FB/$FC weist. Ist der ASCII-Code kleiner als $80, handelt es sich nicht um das letzte Zeichen einer Fehlermeldung. Das Zeichen wird

mit BSOUT ausgegeben und zum Schleifenende verzweigt, um den folgenden Programmteil zu überspringen.

3. Dieser Programmteil führt eine "Sonderbehandlung" durch, wenn das letzte Zeichen einer Fehlermeldung erkannt wurde. $80 wird vom ASCII-Code subtrahiert, um das Zeichen zu "normalisieren" und anschließend mit BSOUT auszugeben. Jede Fehlermeldung soll in einer eigenen Zeile ausgegeben werden. Zum Abschluß einer Meldung wird daher der ASCII-Code $0D (=dezimal 13: Code der RETURN-Taste) ausgegeben und ein "Zeilenvorschub" bewirkt.
4. Bevor die nächste Fehlermeldung ausgegeben wird, wartet das Programm (mit GETIN) auf eine Tastenbetätigung, um ein "Durchscrollen" des Bildschirms zu vermeiden (beim C64 werden 29 Fehlermeldungen ausgegeben, der Bildschirm umfaßt jedoch nur 25 Zeilen).
5. Y wird inkrementiert und mit dem Inhalt von $033C verglichen. Bei Ungleichheit erfolgt ein weiterer Durchgang, das nächste Zeichen wird gelesen.
6. Ebenso wie im vorigen Programm wird nach der kompletten Ausgabe eines Blocks das High-Byte des Zeigers inkrementiert, der Blockzähler X dekrementiert und - wenn ungleich null - eventuell der nächste Block ausgegeben.
7. Nach "vollständiger" Ausgabe aller Blöcke erhält $033C (der Vergleichswert) den Wert $8A, da die innere Schleife erneut gestartet wird, nun jedoch nur $90 Byte des letzten Blocks auszugeben sind ($00-$8A).

```
Vorbereitung
      A C000 A9 9E            LDA #$9E      ;LOW-BYTE VON $A19E
      A C002 85 FB            STA $FB       ;NACH $FB
      A C004 A9 A1            LDA #$A1      ;HIGH-BYTE VON $A19E
      A C006 85 FC            STA $FC       ;NACH $FC
      A C008 A9 00            LDA #$00      ;SCHLEIFENENDWERT $00
      A C00A 8D 3C 03         STA $033C     ;NACH $033C
                                            ;(=>KOMPLETTE BLÖCKE)
      A C00D A2 01            LDX #$01      ;1 KOMPLETTER BLOCK
      A C00F A0 00            LDY #$00      ;Y INITIALISIEREN
      Innere Schleife
      A C011 B1 FB            LDA ($FB),Y   ;AKKU MIT ($FB),Y LADEN
```

```
      A C013 30 06          BMI $C01B    ;SPRUNG, WENN GRÖSSER
                                         ;ALS $79
      A C015 20 D2 FF       JSR $FFD2    ;ZEICHEN AUSGEBEN (BSOUT)
      A C018 4C 33 C0       JMP $C033    ;SONDERBEHANDLUNG
                                         ;ÜBERSPRINGEN!
      A C01B 38             SEC          ;SUBTRAKTION VORBEREITEN
      A C01C E9 80          SBC #$80     ;$80 SUBTRAHIEREN
      A C01E 20 D2 FF       JSR $FFD2    ;NORMALISIERTES ZEICHEN
                                         ;AUSGEBEN
      A C021 A9 0D          LDA #$0D     ;$0D (=CODE FÜR RETURN)
      A C023 20 D2 FF       JSR $FFD2    ;AUSGEBEN (BSOUT)
      A C026 98             TYA          ;Y-WERT AUF
      A C027 48             PHA          ;STACK RETTEN
      A C028 8A             TXA          ;X-WERT AUF
      A C029 48             PHA          ;STACK RETTEN
      A C02A 20 E4 FF       JSR $FFE4    ;MIT GETIN AUF
      A C02D F0 FB          BEQ $C02A    ;TASTE WARTEN
      A C02F 68             PLA          ;X-WERT VOM
      A C030 AA             TAX          ;STACK HOLEN
      A C031 68             PLA          ;Y-WERT VOM
      A C032 A8             TAY          ;STACK HOLEN
      A C033 C8             INY          ;SCHLEIFENZÄHLER
                                         ;DEKREMENTIEREN
      A C034 CC 3C 03       CPY $033C    ;MIT $033C VERGLEICHEN
      A C037 D0 D8          BNE $C011    ;SPRUNG, WENN UNGLEICH
Äußere Schleife
      A C039 E6 FC          INC $FC      ;HIGH-BYTE DES ZEIGERS
                                         ;INKREMENTIEREN
      A C03B CA             DEX          ;BLOCKZÄHLER
                                         ;DEKREMENTIEREN
      A C03C 30 0A          BMI $C048    ;FERTIG (X NEGATIV)?
      A C03E D0 D1          BNE $C011    ;MIT KOMPLETTEN BLOCKS
                                         ;FERTIG?
      A C040 A9 8A          LDA #$8A     ;RESTLICHE BYTEANZAHL
      A C042 8D 3C 03       STA $033C    ;NACH $033C
                                         ;(=VERGLEICHSWERT)
      A C045 4C 11 C0       JMP $C011    ;=> ANFANG INNERE SCHLEIFE
      A C048 00             BRK          ;FERTIG !!!
```

Im Grunde genommen besteht kein allzu großer Unterschied zwischen diesem und dem vorhergehenden Programm. Der Schleifenaufbau ist sehr ähnlich, abgesehen davon, daß diese Schleife vorwärts gezählt wird.

$033C erhält zu Beginn den Wert $00 (siehe Vorbereitung). Die innere Schleife beginnt mit dem Y-Wert $00, der bis $FF aufwärts gezählt wird. Beim folgenden Durchgang entspricht der Inhalt des Y-Registers dem Vergleichswert $00 in der Speicherzelle $033C und ein kompletter Block wurde behandelt.

Das High-Byte des Zeigers ($FB/$FC) wird inkrementiert und weist nun auf den nächsten 256-Byte-Block. Der Blockzähler X wird dekrementiert und - da nur ein Block komplett zu behandeln ist (LDX #$01) - besitzt nun den Wert $00. Die Bedingung BNE $C011 ist nicht erfüllt und die restlichen Bytes sollen ausgegeben werden.

$033C erhält nun den Wert $8A, entsprechend der noch ausstehenden Byteanzahl. Mit JMP $C011 wird die innere Schleife erneut gestartet und von Y=$00 bis Y=$8A aufwärts gezählt.

Folgende Besonderheiten sollten Sie beachten:

- Wenn die innere Schleife nach der Behandlung eines kompletten Blocks verlassen wird, besitzt das Y-Register immer den Inhalt $00, da die Behandlung eines vollständigen Blocks bedeutet, daß Y bis zum Vergleichswert $00 (in $033C) inkrementiert wird. Der folgende Sprung zum Schleifenanfang kann daher zu LDA ($FB),Y erfolgen, das Laden des Y-Registers mit $00 (LDY #$00) ist überflüssig.
- Da GETIN ($FFE4) - die Betriebssystem-Routine zur Tastaturabfrage - die Registerinhalte verändert, müssen diese zuvor auf den Stack "gerettet" und nach Verlassen der Warteschleife wieder vom Stack geholt werden.
- Wie im letzten Programm verzweigt BMI zum Programmende, wenn nach dem letzten vollständigen Block die restlichen Bytes übertragen und X erneut dekrementiert wurde. X besitzt dann den Wert $FF und die Bedingung BMI ("verzweige, wenn negativ") ist erfüllt.

- Gleiches gilt für die Überprüfung des letzten Zeichens einer Fehlermeldung (Bit 7 gesetzt). Auch hier verzweigt BMI, wenn der ASCII-Code des Zeichens größer ist als $79 (negativ = Werte zwischen $80 und $FF).
- Wenn Sie wollen, können Sie den Befehl JMP $C011 durch BNE $C011 ersetzen. Der letzte Befehl, durch den das Zero-Flag beeinflußt wurde, ist LDA #$8A, der zu einem Ergebnis ungleich null führt. Der bedingte Sprung mit BNE wird in diesem speziellen Fall immer ausgeführt.

Logische Operationen

In den vorangegangenen Kapiteln beschäftigten wir uns ausschließlich mit Bytes, die wir lasen, speicherten, addierten oder inkrementierten.

Nun bewegen wir uns ein Stockwerk tiefer und steigen auf die "Bit-Ebene" herab. Mit den Befehlen AND, ORA und EOR können wir gezielt einzelne Bits beeinflussen.

Das Prinzip ist immer gleich: der Akkumulatorinhalt wird mit einem beliebigen Wert nach den Regeln einer der genannten logischen Operationen "verknüpft". Das Ergebnis befindet sich ebenso wie bei arithmetischen Operationen anschließend im Akkumulator. Bei jeder logischen Operation werden je zwei Bits miteinander verknüpft. Das Ergebnis besteht wiederum in einem Bit, das entweder gesetzt (1) oder aber gelöscht (0) ist.

2.18 Logische Verknüpfung

Zuerst beschäftigen wir uns mit der "AND"-Verknüpfung (UND-Verknüpfung). Zwei Bits, die mit AND verknüpft werden, ergeben nur dann das Resultat 1, wenn beide gesetzt sind.

```
0 AND 0 => Ergebnis 0
0 AND 1 => Ergebnis 0
1 AND 0 => Ergebnis 0
1 AND 1 => Ergebnis 1
```

Nach dieser Regel werden alle acht Bits (Bit 0-7) zweier Bytes miteinander verknüpft. Nehmen wir an, wir laden den Wert $03 in den Akkumulator und verknüpfen ihn mit der Zahl $80.

```
      00000011 (=$03)
AND   10000000 (=$80)
-------------------
      00000000 (=$00)
```

Das Ergebnis ist null, da kein Bit in beiden Zahlen zugleich gesetzt ist. Ein Gegenbeispiel:

```
      11000111 (=$C7)
AND   10011110 (=$9E)
-------------------
      10000110 (=$86)
```

In beiden Zahlen sind die Bits 1,2 und 7 zugleich gesetzt. Nach den Regeln der AND-Verknüpfung werden diese drei Bits auch im Ergebnis gesetzt.

Alle anderen Bits werden gelöscht. Im Akkumulator befindet sich nach erfolgter Verknüpfung der Wert $86.

Der Sinn des AND-Befehls besteht darin, gezielt bestimmte Bits einer Zahl zu löschen. Dazu verknüpfen wir die Zahl mit einer sogenannten "Maske".

Ein Beispiel: wir wollen Bit 6 und Bit 7 einer bestimmten Zahl löschen. Dazu laden wir die betreffende Zahl zuerst in den Akkumulator und verknüpfen sie anschließend mit der "Maske" 00111111, das heißt mit dem Wert $3F.

```
      00000000        10101010        11111111
AND   00111111  AND   00111111  AND   00111111
-------------   -------------   -------------
      00000000        00101010        00111111
```

Die drei Beispiele zeigen, daß die Bits 6 und 7 im Ergebnis immer gelöscht werden, egal welche Zahl mit der Maske verknüpft wird. Alle übrigen Bits bleiben unverändert erhalten.

Schema zum gezielten Löschen von Bits:

1. Das Byte, in dem bestimmte Bits gelöscht werden sollen, wird in den Akkumulator geladen.
2. Der Akkumulator wird mit einer Maske AND-verknüpft, in der die zu löschenden Bits gelöscht und alle übrigen Bits gesetzt sind.
3. Nach der Verknüpfung befindet sich das Ergebnis im Akkumulator. Bis auf die gezielt gelöschten Bits entspricht es exakt dem Ausgangswert.

```
Inverses A auf Bildschirm schreiben und auf Taste warten
      A C000 A9 81            LDA #$81     ;BILDSCHIRMCODE A INVERS
      A C002 8D 00 04         STA $0400    ;NACH $0400 SCHREIBEN
      A C005 20 E4 FF         JSR $FFE4    ;MIT GETIN AUF
      A C008 F0 FB            BEQ $C005    ;TASTE WARTEN
Inverses A mit AND-Verknüpfung normalisieren
      A C00A AD 00 04         LDA $0400    ;ZEICHEN AN ADRESSE
                                           ;$0400 LESEN
      A C00D 29 7F            AND #$7F     ;AND-VERKNÜPFUNG MIT $7F
      A C00F 8D 00 04         STA $0400    ;ERGEBNIS ZURÜCKSCHREIBEN
      A C012 00               BRK
```

Dieses kleine Programm zeigt eine praktische Anwendung der AND-Verknüpfung. Wie wir wissen, unterscheiden sich die inversen Bildschirm- und ASCII-Codes von den normalen Zeichen durch ein gesetztes siebtes Bit.

$81 (=%10000001) ist der Bildschirmcode für ein inverses A. Dieses Zeichen wird in die obere linke Bildschirmecke geschrieben ($0400) und anschließend mit GETIN auf eine Taste gewartet.

Nach dieser Vorbereitung erfolgt die eigentliche Demonstration. Wenn Sie eine beliebige Taste betätigen, wird das Programm fortgesetzt. Der Inhalt von Speicherzelle $0400 wird in den Ak-

kumulator gelesen und mit dem Wert $7F (%01111111) AND-verknüpft. Anschließend wird das Ergebnis zurückgeschrieben und auf dem Bildschirm sehen Sie anstelle des inversen A's ein normal dargestelltes A.

```
      10000001 (=$81)   Inverses A
AND   01111111 (=$7F)   Maske zum Löschen von Bit 7
-------------------
      00000001 (=$01)   Normales A
```

Die ODER-Verknüpfung

Die "ORA"-Verknüpfung (ODER-Verknüpfung) ist das Gegenteil der AND-Verknüpfung. Werden zwei Bits mit AND verknüpft, ist das Ergebnis immer dann ein gesetztes Bit, wenn entweder ein oder aber beide Bits gesetzt waren ("inclusives" ODER).

```
0 ORA 0 => 0
0 ORA 1 => 1
1 ORA 1 => 1
1 ORA 1 => 1
```

In der Praxis bedeutet diese Eigenschaft der ORA-Verknüpfung, daß wir gezielt Bits setzen können. Wir verknüpfen das betreffende Byte mit einer Maske, in der das gewünschte Bit gesetzt und alle anderen Bits gelöscht sind. Das folgende Schema zeigt, wie mit der Maske %11000000 in beliebigen Zahlen die Bits 6 und 7 gesetzt werden.

```
      00000000        10101010        11111111
ORA   11000000   ORA  11000000   ORA  11000000
-------------------------------------------------
      11000000        11101010        11111111
```

Schema zum gezielten Setzen von Bits:

1. Wir laden den betreffenden Wert in den Akkumulator.

2. Wir verknüpfen den Akkumulator mit einer Maske, in der alle Bits gelöscht sind, außer jenen, die gesetzt werden sollen.
3. Als Ergebnis erhalten wir im Akkumulator den Originalwert, wobei jedoch alle Bits gesetzt sind, die auch in der Maske gesetzt waren. Alle anderen Bits bleiben unverändert erhalten.

Mit ORA können wir zum Beispiel Zeichen invertieren, ohne $80 addieren zu müssen. Wir setzen einfach das siebte Bit, indem wir den jeweiligen Bildschirmcode mit der Maske %10000000 ($80) verknüpfen.

```
A C000 A2 00          LDX #$00      ;ZÄHLER INITIAL.
A C002 BD 00 04       LDA $0400,X   ;ZEICHEN LESEN
A C005 09 80          ORA #$80      ;ORA-VERKNÜPFUNG MIT $80
A C007 9D 00 04       STA $0400,X   ;ZURÜCKSCHREIBEN
A C00A CA             DEX           ;ZÄHLER DEKREMENT.
A C00B D0 F5          BNE $C002     ;FERTIG?
A C00D 00             BRK
```

Dieses Programm invertiert die ersten 256 Zeichen auf dem Bildschirm. Zeichen für Zeichen wird gelesen und mit der Maske $80 (=%10000000) OR-verknüpft. Bis auf das gesetzte siebte Bit unverändert werden die - invertierten - Zeichen zurückgeschrieben.

Die EXKLUSIV-ODER-Verknüpfung

Mit der AND- und der ORA-Verknüpfung sind wir in der Lage, unser Programm "blinkende Bildschirmzeile" einfacher zu gestalten. Es geht jedoch noch weitaus komfortabler, wenn wir die EOR-Verknüpfung ("Exklusic-ODER") verwenden.

Werden zwei Bits mit EOR verknüpft, resultiert nur dann ein gesetztes Bit, wenn entweder das eine oder das andere Bit gesetzt war.

Beachten Sie bitte die Formulierung "entweder oder". Sind beide Bits gesetzt, ist das "Ergebnis-Bit" im Gegensatz zur ORA-Verknüpfung nicht gesetzt!

```
0 EOR 0 => 0
0 EOR 1 => 1
1 EOR 0 => 1
1 EOR 1 => 0
```

EOR setzt man häufig zur "Umkehrung" von Bits ein. Wird ein beliebiger Wert mit der Maske %11111111 EOR-verknüpft, kehren sich alle Bitzustände um.

```
    00000000        10101010        11111111
EOR 11111111    EOR 11111111    EOR 11111111
------------    ------------    ------------
    11111111        01010101        00000000
```

Wollen Sie nur ein bestimmtes Bit "umdrehen", verknüpfen Sie es (mit EOR) mit einer Maske, in der nur dieses Bit gesetzt und alle anderen gelöscht sind. Beispielsweise können mit der Maske %11000000 die Bits 6 und 7 in beliebigen Zahlen "umgedreht" werden.

```
    00000000        10101010        11111111
EOR 11000000    EOR 11000000    EOR 11000000
------------    ------------    ------------
    11000000        01101010        00111111
```

Schema zur gezielten "Umkehrung" von Bits:

1. Laden Sie den betreffenden Wert in den Akkumulator.
2. Verknüpfen Sie den Wert (EOR-Verknüpfung) mit einer Maske, in der nur die umzukehrenden Bits gesetzt und alle anderen Bits gelöscht sind.
3. Im Akkumulator befindet sich nun der Originalwert, wobei jedoch die in der Maske gesetzten Bits gegenüber dem Original umgedreht sind.

EOR bietet uns daher eine fantastische Möglichkeit, unsere "blinkende Bildschirmzeile" so einfach wie möglich zu verwirklichen: wir kehren Bit 7 einfach bei jedem Aufruf der Routine um, indem wir dann die betreffenden Bildschirmcodes mit der Maske %10000000 (=$80) "eoren". Ist Bit 7 gesetzt, wird es gelöscht und umgekehrt.

Um das Ganze etwas abwechslungsreicher zu gestalten, lassen wir im Gegensatz zu früheren Programmen diesmal den kompletten Bildschirm blinken.

```
Blinkender Bildschirm
Vorbereitung
      A C000 A9 00          LDA #$00      ;LOW-BYTE VON $0400
      A C002 85 FB          STA $FB       ;NACH $FB
      A C004 A9 04          LDA #$04      ;HIGH-BYTE VON $0400
      A C006 85 FC          STA $FC       ;NACH $FB
      A C008 A2 04          LDX #$04      ;BLOCKZÄHLER MIT $04 LADEN
      A C00A A0 00          LDY #$00      ;ZÄHLER INNEN
                                          ;INITIALISIEREN
Bildschirm invertieren/normalisieren
      A C00C B1 FB          LDA ($FB),Y   ;ZEICHEN LESEN
      A C00E 49 80          EOR #$80      ;BIT 7 UMDREHEN
      A C010 91 FB          STA ($FB),Y   ;ZEICHEN ZURÜCKSCHREIBEN
      A C012 88             DEY           ;ZÄHLER INNEN DEKREMENT.
      A C013 D0 F7          BNE $C00C     ;EIN BLOCK FERTIG?
      A C015 E6 FC          INC $FC       ;ZEIGER: HIGH-BYTE
                                          ;INKREMENTIEREN
      A C017 CA             DEX           ;BLOCKZÄHLER DEKREMENT.
      A C018 D0 F2          BNE $C00C     ;ALLE 4 BLOCKS FERTIG?
Warteschleifen
      A C01A 88             DEY           ;ZÄHLER INNEN DEKREMENT.
      A C01B D0 FD          BNE $C01A     ;256 DURCHGÄNGE?
      A C01D CA             DEX           ;ZÄHLER AUSSEN DEKREMENT.
      A C01E D0 FA          BNE $C01A     ;256 DURCHGÄNGE?
      A C020 4C 00 C0       JMP $C000     ;NEUSTART DES PROGRAMMS
```

Wenn Sie dieses Programm eingeben, wundern Sie sich eventuell, welche Effekte mit einem derartig kurzen Programm zu erzielen sind. Das Prinzip der Bildschirmbehandlung mit der indirekt-in-

dizierten Adressierung werde ich nicht erläutern, da der verwendete Schleifenaufbau inzwischen hinreichend bekannt sein dürfte.

Die innere Schleife ist dank EOR extrem kurz. Wir müssen nicht mehr überprüfen, ob ein Zeichen invertiert oder normalisiert ist. EOR erledigt die Umkehrung des jeweiligen Zustandes automatisch.

Interessant sind die ebenfalls extrem kurzen ineinander geschachtelten Warteschleifen am Programmende (unbedingt nötig, da die Blinkfrequenz sonst viel zu hoch ist).

Diese Warteschleife ist so kurz, da die Tatsache ausgenutzt wird, daß sowohl X als auch Y nach dem Verlassen der Hauptschleife den Wert $00 haben und eine Neuinitialisierung (LDX #$00 und LDY #$00) daher überflüssig ist.

Wahrheitstabellen und Adressierungsarten

Um die Auswirkungen bestimmter logischer Verknüpfungen zu überprüfen, benutzt man normalerweise "Wahrheitstabellen", wie sie nachstehend für alle drei Verknüpfungsarten abgebildet sind.

AND	0	1
0	0	0
1	0	1

ORA	0	1
0	0	1
1	1	1

EOR	0	1
0	0	1
1	1	0

Um diese Tabellen anzuwenden, prüfen Sie, ob sich am Kreuzungspunkt der jeweiligen "Bitzustände" eine null oder eine eins befindet.

Beispiel: Sie verknüpfen zwei gesetzte Bits mit EOR. In der EOR-Tabelle finden Sie am Kreuzungspunkt der beiden Bitzustände 1 (=gesetzt) den Wert null vor. Die Bedeutung: wird ein gesetztes Bit mit einem ebenfalls gesetzten Bit EOR-verknüpft, resultiert als Ergebnis ein gelöschtes Bit.

Bei allen drei Verknüpfungen können die folgenden Adressierungsarten verwendet werden:

```
Unmittelbar:          AND #$01
Absolut:              AND $0400
Zeropage:             AND $FF
X-indiziert:          AND $0400,X
Y-indiziert:          AND $0400,Y
Indirekt-indiziert:   AND ($FB),Y
Indiziert-indirekt:   AND ($FB,X)
Zeropage-X-indiziert: AND $FB,X
```

Stören Sie sich bitte nicht an den beiden zuletzt aufgeführten und noch unbekannten Adressierungsarten, die relativ selten verwendet werden.

2.19 Die Schiebe- und Rotierbefehle

Wir sahen, wie Bits gezielt gesetzt, gelöscht oder umgedreht werden. Die "Schiebebefehle" ASL, LSR, ROL und ROR ermöglichen es uns, das "Bitmuster", aus dem ein Byte besteht, beliebig zu verschieben.

ASL (Arithmetisches Nach-links-Schieben)

ASL bedeutet "arithmetic shift left" oder "Arithmetisches Nach-links-Schieben". Wird keine bestimmte Adresse nach ASL angegeben, bezieht sich der Befehl - ebenso wie bei allen anderen Schiebebefehlen - auf den Inhalt des Akkumulators ("Akkumulator-Adressierung").

Alle Bits des im Akkumulator enthaltenen Wertes wandern um eine Stelle nach links. Aus Bit 0 wird Bit 1, aus Bit 1 wird Bit 2 und so weiter. Das freigewordene Bit 0 wird immer mit 0 (gelöschtes Bit) gefüllt.

```
        11110000              01010101              00011011
ASL=>   11100000      ASL=>   10101010      ASL=>   00110110
```

Bit 7 verschwindet bei dieser "Schiebung". Es wandert in das "neunte" Bit des Akkumulators, ins Carry-Bit, und kann mit BCS und BCC getestet werden.

Der Sinn dieser Linksverschiebung? Betrachten Sie bitte den Wert der drei Dualzahlen vor und nach ASL. Bei Dualzahlen bedeutet Verschiebung um eine Stelle nach links Verdopplung. Entsprechend verdoppelt sich der Zahlenwert der drei als Beispiele verwendeten Bitmuster.

```
Ausgangswert:   00000001
ASL       =>    00000010
ASL       =>    00000100
ASL       =>    00001000
ASL       =>    00010000
ASL       =>    00100000
ASL       =>    01000000
ASL       =>    10000000
ASL       =>    00000000
```

Diese Reihe von ASL-Befehlen zeigt, wie das Bitmuster nach links verschoben und von rechts eine null ins Bit 0 nachgeschoben wird. Jede Verschiebung entspricht einer Verdoppelung der Zahl.

Nach dem letzten ASL-Befehl erfolgte ein "Übertrag" des zuvor gesetzten siebten Bits ins Carry-Bit. Ob das ins Carry-Bit geschobene Bit gesetzt oder aber gelöscht war, können Sie nach einem ASL-Befehl mit BCS ("verzweige, wenn Carry gesetzt") und BCC ("verzweige, wenn Carry gelöscht") abfragen.

Wie erläutert, bezieht sich ASL ohne nähere Angabe auf den Inhalt des Akkumulators. Außer dieser "Akkumulator-Adressierung" sind folgende Adressierungsarten möglich:

```
Absolut:              ASL $0400
Zeropage:             ASL $FF
X-indiziert:          ASL $0400,X
Zeropage-X-indiziert: ASL $FF,X
```

Angenommen, sie wollen eine Zahl, die sich in Speicherzelle $033C befindet, mit vier multiplizieren. Dann schieben Sie den Inhalt von $033C zweimal nach links.

```
ASL $033C   ;MAL 2
ASL $033C   ;NACHMAL MAL 2
```

Komplizierter liegt der Fall jedoch, wenn Sie eine Zahl zum Beispiel mit 23 multiplizieren wollen. In diesem Fall ist ein wenig Nachdenken gefordert.

Die Frage ist, wie man den Faktor 21 am geschicktesten in "Binärpotenzen" zerlegen kann, also in 1, 2, 4, 8, 16, 32, 64 und 128.

```
21 = 16 + 4 + 1
```

Diese Zerlegung zeigt, daß es möglich ist, die Multiplikation einer Zahl mit 21 zu ersetzen, indem die Zahl zuerst mit 16 (Zahl * 16) und dann mit vier multipliziert wird (Zahl * 4), die Ergebnisse addiert und zum Schluß die Zahl selbst noch einmal addiert wird (Zahl * 1).

Gehen wir von der Zahl $0A aus, die sich im Akkumulator befinden soll.

```
A C000 A9 0A          LDA #$0A     ;AUSGANGSZAHL $0A
A C002 8D 3C 03       STA $033C    ;NACH $033C BRINGEN
A C005 0A             ASL          ;=> ZAHL * 2
A C006 0A             ASL          ;=> ZAHL * 4
A C007 8D 3D 03       STA $033D    ;ZAHL * 4 NACH
                                   ;$033D BRINGEN
A C00A 0A             ASL          ;=> ZAHL * 8
A C00B 0A             ASL          ;=> ZAHL * 16
A C00C 18             CLC          ;ADDITION VORBEREITEN
A C00D 6D 3D 03       ADC $033D    ;ZAHL * 4 ADDIEREN
A C010 6D 3C 03       ADC $033C    ;ZAHL * 1 ADDIEREN
A C013 00             BRK          ;ERGEBNIS: ZAHL * 21
                                   ;IM AKKU
```

Die Zahl $0A befindet sich im Akkumulator und wird zuerst nach $033C "gerettet". Die beiden folgenden ASL-Befehle entsprechen einer Multiplikation mit vier. Das Ergebnis ZAHL*4 kommt nach $033D.

Zwei weitere ASL-Befehle entsprechen wieder einer Multiplikation mit vier. ZAHL*4 mulipliziert mit vier ergibt das zweite Ergebnis, ZAHL*16. Nun wird das erste Ergebnis ZAHL*4 (in $033D) und zum Schluß der Ausgangswert ZAHL (in $033C) addiert.

Nach dem Aufruf des Monitors (BRK) gibt die Registeranzeige den Wert $D2 (=dezimal 210) als Inhalt des Akkumulators aus, das 21fache des Ausgangswerts $0A (=dezimal 10).

LSR (Logisches Nach-rechts-Schieben)

LSR ("logical shift right", "logisches Nach-Rechts-Schieben") ist die Umkehrung des Befehls ASL.

Während ASL das angegebene Bitmuster (im Akkumulator oder einer Speicherzelle) nach links schiebt, das Bit 7 ins Carry-Bit schiebt und als neues Bit 0 immer ein gelöschtes Bit einfügt, arbeitet LSR umgekehrt.

Das Bitmuster wird nach rechts geschoben, wobei Bit 0 "herausfällt" und ins Carry-Bit kommt. Als neues siebtes Bit wird 0 eingefügt.

```
        11110000            01010101            00011011
LSR=>   01111000    LSR=>   00101010    LSR=>   00001101

Ausgangswert:   10000000
LSR         =>  01000000
LSR         =>  00100000
LSR         =>  00010000
LSR         =>  00001000
LSR         =>  00000100
LSR         =>  00000010
```

```
LSR     =>   00000001
LSR     =>   00000000
```

Die Tabelle zeigt das zugrundeliegende Schema. Jeder ASL-Befehl entspricht einer Division durch zwei. Die ersten sieben Befehle schieben jeweils eine 0 ins Carry-Bit (Carry-Flag gelöscht).

Nach dem achten ASL-Befehl erhalten wir als Ergebnis den Wert null, das zuvor in der Zahl gesetzte Bit 0 befindet sich im Carry-Bit, das nun gesetzt ist. Wie üblich kann der Zustand des Carry-"Bits" oder -"Flags" mit BCS und BCC getestet werden.

ROR und ROL (nach rechts/links rotieren)

ROR ("rotate right", "rotiere nach links") und ROL ("rotate left", "rotiere nach rechts") besitzen eine etwas andere Funktionsweise als ASL und LSR.

Beide Befehle führen sogenannte "Ringverschiebungen" durch. Der Unterschied zu ASL und LSR: in die freiwerdende Stelle beim "Links-" beziehungsweise "Rechtsschieben" wird nicht automatisch eine null nachgeschoben, sondern das aktuelle Carry-Bit (null oder eins).

```
CLC=> Carry löschen
Ausgangswert:  10111000
       ROL=>  11100000  (Carry gesetzt)
       ROL=>  11000001  (Carry gesetzt)
```

Der Befehl CLC ist in diesem Beispiel notwendig, um vor einem Rotierbefehl für einen "definierten Ausgangszustand" des Carry-Bits zu sorgen (gelöscht). Im Beispiel wird das Bitmuster mit ROL nach links geschoben.

Als Bit 0 wird das momentane Carry-Bit (0 wegen CLC) "nachgeschoben". Das siebte Bit (1) wandert ins Carry-Bit, das nun gesetzt ist. Der folgende ROL-Befehl schiebt das Bitmuster wieder um eine Stelle nach links, diesmal wird als Bit 0 jedoch eine

1 nachgeschoben, da das Carry-Bit aufgrund des vorangegangenen ROR-Befehls gesetzt ist. Das Carry-Bit wird wiederum gesetzt, da auch dieser ROL-Befehl ein gesetztes siebtes Bit ins Carry-Bit schiebt.

Daraus geht hervor, daß neun ROL-Befehle den ursprünglichen Zustand des Ausgangswerts wiederherstellen (Voraussetzung: vor dem ersten ROL-Befehl wird das Carry-Bit mit CLC gelöscht).

```
Ausgangswert:   11110000
CLC       =>                (Carry gelöscht)
ROL       =>    11100000    (Carry gesetzt)
ROL       =>    11000001    (Carry gesetzt)
ROL       =>    10000011    (Carry gesetzt)
ROL       =>    00000111    (Carry gesetzt)
ROL       =>    00001111    (Carry gelöscht)
ROL       =>    00011110    (Carry gelöscht)
ROL       =>    00111100    (Carry gelöscht)
ROL       =>    01111000    (Carry gelöscht)
ROL       =>    11110000    (Carry gelöscht)
```

ROR arbeitet genau entgegengesetzt. Wie bei LSR findet eine Linksverschiebung statt, in das freigewordene siebte Bit wird nicht wie bei LSR eine null, sondern ebenso wie bei ROL das aktuelle Carry-Bit nachgeschoben.

```
CLC=> Carry löschen
Ausgangswert:  10100111
       ROR=>   01010011  (Carry gesetzt)
       ROR=>   10101001  (Carry gesetzt)
```

Ebenso wie bei ROL kann mit neun Ringverschiebungen der ursprüngliche Zustand wiederhergestellt werden.

```
Ausgangswert:   11110000
CLC       =>                (Carry gelöscht)
ROR       =>    01111000    (Carry gelöscht)
ROR       =>    00111100    (Carry gelöscht)
ROR       =>    00011110    (Carry gelöscht)
ROR       =>    00001111    (Carry gelöscht)
```

```
ROR     =>   00000111   (Carry gesetzt)
ROR     =>   10000011   (Carry gesetzt)
ROR     =>   11000001   (Carry gesetzt)
ROR     =>   11100000   (Carry gesetzt)
ROR     =>   11110000   (Carry gesetzt)
```

Alle Schiebe- und Rotierbefehle können mit den bereits bei ASL erläuterten Adressierungsarten eingesetzt werden. Nicht nur der Inhalt des Akkumulators, sondern beliebiger Speicherzellen kann somit "geschoben" und "rotiert" werden.

Wie die folgenden Abschnitte zeigen, sind ROL und ROR vor allem in Verbindung mit den Schiebebefehlen ASL und LSR sehr effektiv einzusetzen.

Gezieltes Testen von Bits

Angenommen, wir müssen unbedingt wissen, ob ein bestimmtes Bit in einer Speicherzelle gesetzt oder aber gelöscht ist. Beim siebten Bit ist die einfachste Lösung das Laden des Akkumulators mit der jeweiligen Speicherzelle und das Testen auf positiv oder negativ mit BMI und BPL.

```
LDA $033C       ;WERT LADEN
BMI (ADRESSE)   ;SPRUNG, WENN NEGATIV (BIT 7 GESETZT)
```

Dabei wird jedoch der Inhalt des Akkumulators verändert und muß - wenn benötigt - zuvor auf den Stack gebracht und anschließend wieder geholt werden. Einfacher ist die folgende Möglichkeit:

```
ASL $033C       ;INHALT VON $033C EINE STELLE NACH LINKS SCHIEBEN
BCS (ADRESSE)   ;SPRUNG, WENN BIT 7 GESETZT WAR (NUN IM CARRY)
```

Um Bit 0 zu testen, verwenden wir LSR:

```
LSR $033C       ;INHALT VON $033C EINE STELLE NACH RECHTS SCHIEBEN
BCS (ADRESSE)   ;SPRUNG, WENN BIT 0 GESETZT WAR (NUN IM CARRY)
```

Die Schiebebefehle ASL und LSR werden wir immer dann verwenden, wenn der ursprüngliche Inhalt der Speicherzelle nicht mehr benötigt wird.

Wollen wir hingegen gezielt Bits testen, den Wert selbst jedoch wiederherstellen, wenn der Test negativ ausfiel, verwenden wir (zuvor Carry-Bit mit CLC löschen!) eine Kombination aus ASL beziehungsweise LSR, und ROR beziehungsweise ROL:

```
ASL $033C      ;INHALT VON $033C EINE STELLE NACH LINKS SCHIEBEN
BCS (ADRESSE) ;SPRUNG, WENN BIT 7 GESETZT WAR (NUN IM CARRY)
ROR $033C      ;WIEDER NACH RECHTS SCHIEBEN INCLUSIVE CARRY
```

ASL schiebt das Bitmuster eine Stelle nach links, wobei das siebte Bit im Carry-Bit "landet". Nach dem Test schieben wir den Wert mit ROR wieder eine Stelle nach rechts zurück. Der alte Zustand wird wiederhergestellt, da ROR nicht einfach eine null, sondern das Carry-Bit in die freiwerdende Stelle schiebt. Und im Carry-Bit befindet sich ja nach ASL das ursprüngliche siebte Bit des betreffenden Wertes.

Um Bit 0 zu testen, verwenden wir die umgekehrte Kombination (LSR und ROL):

```
LSR $033C      ;INHALT VON $033C EINE STELLE NACH RECHTS SCHIEBEN
BCS (ADRESSE) ;SPRUNG, WENN BIT 7 GESETZT WAR (NUN IM CARRY)
ROL $033C      ;WIEDER NACH LINKS SCHIEBEN INCLUSIVE CARRY
```

Merken Sie sich: sollen nur Bits getestet werden, ist die einfachste Möglichkeit die Verwendung von LSR und ASL. Benötigen wir den Wert jedoch anschließend unverändert, schieben wir das jeweilige Bit mit ASL oder LSR ins Carry-Bit, testen mit BCS oder BCC dieses Bit und stellen anschließend den ursprünglichen Wert mit ROR oder ROL wieder her.

Um andere Bits (0,1,..,6,7) zu testen, schieben wir so lange, bis sich das gewünschte Bit im Carry-Bit befinden und testen dieses mit BCC oder BCS. Soll anschließend der ursprüngliche Wert wiederhergestellt werden, verwenden wir entsprechend viele Rotierbefehle. Beachten Sie jedoch unbedingt, daß nur die erste

"Schiebung" mit ASL oder LSR durchgeführt werden darf, alle folgenden mit ROL oder ROR, da sonst Bits unwiederbringlich "verlorengehen".

```
Bit 5 testen
ASL $033C       ;BIT 7 INS CARRY
ROL $033C       ;BIT 6 INS CARRY
ROL $033C       ;BIT 5 INS CARRY
BCS (ADRESSE)   ;BIT 5 TESTEN
ROR $033C       ;NACH RECHTS INCLUSIVE CARRY
ROR $033C       ;NACH RECHTS INCLUSIVE CARRY
ROR $033C       ;NACH RECHTS INCLUSIVE CARRY
```

Angenommen, in $033C befindet sich der Wert %10111001 (=$B9). Dann ergibt sich folgender Ablauf:

```
Ausgangszustand: $033C=%10111001, Carry: unbekannt
Befehl      $033C        Carry
ASL         01110010     GESETZT
ROL         11100101     GELÖSCHT
ROL         11001010     GESETZT
BCS           KEINE VERÄNDERUNG
ROR         11100101     GELÖSCHT
ROR         01110010     GESETZT
ROR         10111001     GELÖSCHT
```

Weitere Einsatzmöglichkeiten

Wir wissen nun, wir wir mit den Schiebebefehlen einen 8-Bit-Wert mit beliebigen Potenzen zur Basis zwei (2^1=2, 2^2=4, 2^3=8,...) multiplizieren können. Wir schieben ihn ein-, zwei-, dreimal oder auch öfters nach links. Wenn nötig, speichern wir Zwischenergebnisse und können mit deren Hilfe auch "ungerade" Multiplikationen durchführen.

Wenn wir jedoch eine 16-Bit-Zahl auf diese Weise mit einem Wert multiplizieren wollen, bekommen wir Probleme. Gehen wir von der Zahl $1082 aus, die wir im "Adreßformat" (zuerst Low-, dann High-Byte) in $033C und $033D ablegen.

```
$033D (High-Byte $10)     $033C (Low-Byte $82)
    %00001010                 %10000010
```

Theoretisch wäre es nun möglich, zuerst das Low- und anschließend das High-Byte nach links zu schieben. Leider bleibt dabei der Übertrag unberücksichtigt, der beim Linksschieben des Low-Bytes entsteht und ins Carry-Bit wandert.

Dieser muß im High-Byte anschließend mitberücksichtigt und zu dessen Bit 0 werden.

```
$033D (High-Byte)     $033C (Low-Byte)
  %00010100  <=  1  <=  %00000100
```

Das Problem wird durch den kombinierten Einsatz von ASL und ROL gelöst. Mit ASL werden die acht Bits des Low-Bytes ($033C) nach links verschoben, wobei Bit 7 ins Carry-Bit wandert.

Anschließend wird das High-Byte nicht mit ASL, sondern mit ROL ebenfalls um eine Stelle nach links geschoben. Dabei wird das Carry-Bit - das im Beispiel gesetzt ist - zum Bit 0 des High-Bytes. Ein eventueller Übertrag im Carry-Bit wird durch die Anwendung von ROL auf das High-Byte automatisch berücksichtigt.

```
ASL $033C ;LOW-BYTE * 2
ROL $033D ;HIGH-BYTE * 2 (UNTER BERÜCKSICHTIGUNG EINES ÜBERTRAGS)
```

Interruptprogrammierung (SEI, CLI, RTI und BRK)

Interrupt-Programme werden oft als die "Krönung" der Assembler-Programmierung bezeichnet. Mit keiner anderen Programmiertechnik lassen sich derart effektvolle Programme erstellen, deren Auswirkungen dem Laien oftmals völlig unerklärlich sind.

Bevor wir uns diesem "Leckerbissen" zuwenden, muß ich jedoch die Frage klären, was unter einem "Interrupt" zu verstehen ist. Im Grunde genommen nichts anderes als eine "Unterbrechung".

Der Rechner unterbricht seine laufende Arbeit (zum Beispiel die Ausführung eines BASIC-Programms) für einen Moment, um sich anderen Beschäftigungen zuzuwenden.

Vielleicht fragen Sie sich manchmal, wieso im Direkt-Modus der Cursor ständig blinkt, welches "Heinzelmännchen" im Rechner ununterbrochen eine Bildschirm-Speicherzelle abwechselnd invertiert und wieder normalisiert.

Oder wer ständig die rechnerinterne Uhr (die Sie in BASIC mit den Variablen TI und TI$ abfragen können) weiterzählt und in jedem Moment die Tastatur abfragt?

Nun, für alle diese grundlegenden Aufgaben sind sogenannte Interrupt-Programme zuständig. Ebenso wie im täglichen Leben (das Telefon klingelt und unterbricht Ihre augenblickliche Tätigkeit) werden Interrupt-Programme immer dann aufgerufen, wenn eine "Unterbrechung" auftritt.

Die CPU kennt nun verschiedene Unterbrechungsarten und demzufolge verschiedene Interrupts. Die für unsere Zwecke wichtigsten Interrupts heißen IRQ und NMI.

IRQ und NMI

Der IRQ ("interrupt request", "Unterbrechungsanforderung") besitzt eine andere "Priorität" als die zweite Interrupt-Sorte, der NMI ("non maskable interrupt", "nicht maskierbarer Interrupt").

Um zu verstehen, was unter Priorät gemeint ist, greife ich wieder auf ein alltägliches Beispiel zurück. Angenommen, Ihre momentane Tätigkeit besteht im Lesen dieses Buches. Nun kommt ein IRQ, eine "maskierbare Unterbrechunganforderung". Und zwar klingelt das Telefon.

Sie werden Ihre Tätigkeit - das Lesen - wahrscheinlich unterbrechen und ans Telefon gehen. Nun tritt jedoch ein NMI auf, eine "nicht maskierbare Unterbrechung", Sie hören, daß in der Küche das Wasser überkocht.

Der NMI ("Wasser kocht") besitzt zweifellos eine höhere Priorität als der IRQ ("Telefon klingelt"). Die Unterbrechung wird ihrerseits durch eine wichtigere Unterbrechung unterbrochen und Sie werden sofort in die Küche rennen.

Wenn der NMI erledigt ist (Sie nahmen den Topf von der Herdplatte) können Sie sich wieder dem IRQ zuwenden, der weniger wichtigen Tätigkeit des Telefonierens.

Erst wenn auch diese Unterbrechung erledigt ist, wenden Sie sich wieder Ihrer normalen - und sehr sinvollen - Beschäftigung zu, dieses Buch zu lesen.

Dieses Beispiel beschreibt hervorragend die Reaktion der CPU auf eine Unterbrechungsanforderung. Die Frage ist nun, wie eine solche Unterbrechung erzeugt, wie ein Interrupt "ausgelöst" wird.

Unser Rechner besitzt die verschiedensten "Interruptquellen", vor allem die sogenannten CIA-Bausteine, deren "Register" unter anderem für die Farben des Bildschirmrahmens ($D020) und -hintergrundes zuständig sind.

Uns interessiert nun vor allem eine der verschiedenen Unterbrechungsanforderungen, die dieser Baustein erzeugt. Er enthält eine interne Uhr, für die spezielle Register zuständig sind, die sogenannten "Timer".

Nach jeder sechzigstel Sekunde lösen diese Timer einen IRQ aus, einen maskierbaren Interrupt. Eine weitere IRQ-Quelle ist der BRK-Befehl. Dieser Befehl löst ebenfalls einen IRQ aus, wobei zugleich das sogenannte "BRK-Flag" oder "Break-Flag" im Status-Register (Bit 3) gesetzt wird, damit beim folgenden Ablauf zwischen einem durch die Timer und einem durch einen BRK-Befehl ausgelösten IRQ unterschieden werden kann.

Der Prozessor reagiert auf einen IRQ folgendermaßen:

1. Der Inhalt des Programmzählers - der ja die Adresse des nächsten zu bearbeitenden Befehls enthält - und des Status-Registers werden auf den Stack "gerettet".
2. Im Status-Register wird das Bit 1 gesetzt. Solange dieses Bit gesetzt ist, werden keine weiteren IRQ's bearbeitet (nur ein NMI kann einen IRQ unterbrechen).
3. Am Speicherende ($FFFE/$FFFF) befindet sich eine Adresse in der üblichen Form Low-Byte/High-Byte im ROM, also im nicht veränderbaren Speicherteil. Der Programmzähler wird mit dieser Adresse geladen. Die weitere Programmbearbeitung erfolgt daher ab der jeweiligen Adresse.

Punkt 3 entspricht dem bereits bekannten Sprung über einen Vektor (JMP (VEKTOR)). Die Sprungadresse befindet sich unveränderbar in den Speicherzellen $FFFE/$FFFF.

Beim C64 befindet sich in diesen Speicherzellen die Adresse $FF48, die Startadresse einer Routine, die folgende Aufgaben besitzt:

1. Die aktuellen Registerinhalte werden auf den Stack "gerettet".
2. Anhand des Status-Registers, das auf den Stack gebracht wurde (siehe oben), wird geprüft, ob der IRQ durch die Timer oder durch einen BRK-Befehl ausgelöst wurde.
3. Bei einem BRK-IRQ erfolgt ein Sprung über den Vektor $0316/$0317 (JMP ($0316)).
4. Bei einem Timer-IRQ erfolgt ebenfalls ein indirekter Sprung, jedoch über den Vektor $0314/$0315, der normalerweise die Adresse $EA31 enthält (JMP ($0314)).

Punkt 3 erklärt den Rücksprung in den Monitor mit dem BRK-Befehl. Jeder Monitor "verstellt" nach dem Laden und Starten zunächst einmal den BRK-Vektor an Adresse $0316/$0317.

Dieser Vektor wird so verändert, daß er anschließend mitten in das Monitor-Programm weist, genauer: auf die Registeranzeige.

Wenn wir mit einem BRK-Befehl unser Programm beenden, wird beim Erreichen dieses Befehls ein IRQ ausgelöst und anschließend über den BRK-Vektor zur Registeranzeige des Monitors gesprungen. Ein Geheimnis wäre damit "gelüftet".
Interessanter für unsere Zwecke ist jedoch der Timer-IRQ und der resultierende Sprung über den Vektor $0314/$0315 zur Adresse $EA31. Ab Adresse $EA31 befindet sich beim C64 die sogenannte "Service-Routine", die folgende Aufgaben besitzt:

1. Die Uhr wird weitergezählt.
2. Der Cursor wird invertiert, wenn er gerade normalisiert ist beziehungsweise normalisiert, wenn er momentan invertiert ist.
3. Die Tastatur wird abgefragt, vor allem die STOP-Taste (nun wissen Sie, warum Sie ein BASIC-Programm in jedem Augenblick mit STOP unterbrechen können).
4. Die ursprünglichen Inhalte der Register und des Programmzählers werden wieder vom Stack geholt.
5. Mit dem Befehl RTI ("return from interrupt", "beende die Unterbrechung") wird dem Prozessor angezeigt, daß die Unterbrechung beendet ist.

Der Prozessor setzt die Bearbeitung des unterbrochenen Programms mit dem nächsten Befehl (der alte Inhalt des Programmzählers wurde wieder vom Stack geholt) fort, als hätte die Unterbrechung niemals stattgefunden.

Aus diesen Erläuterungen geht hervor, wie wir eigene Programme in den "Systeminterrupt" - der jede sechzigstel Sekunde stattfindet - "einbinden" können. Wir verändern den IRQ-Vektor $0314/$0315, so daß er statt auf die normale Service-Routine ab $EA31 auf die Startadresse unseres eigenen Programms weist, das jede sechzigstel Sekunde aufgerufen werden soll.

Eines ist dabei jedoch zu beachten: zu jeder "ordnungsgemäßen" Einbindung einer Interrupt-Routine gehört als letzter Programmbefehl ein Sprung zur Service-Routine (JMP $EA31). Erfolgt dieser Sprung nicht, wird weder die STOP-Taste abgefragt, noch die Uhr weitergezählt oder der Cursor kontinuierlich invertiert und normalisiert (=> der Cursor blinkt).

Alle grundlegenden "Systemfunktionen" sind ausgeschaltet, wenn wir die Service-Routine einfach umgehen. Daher werden eigene Interrupt-Routinen in Form einer "Verlängerung" eingebunden.

Der IRQ-Vektor wird auf unser Programm verstellt, dieses ausgeführt, und zum Schluß die normale Service-Routine aufgerufen.

Angenommen, unsere Interrupt-Routine beginnt an Adresse $C00D. Dann verstellen wir den IRQ-Vektor auf diese Adresse mit den Befehlen:

```
LDA #$0D    ;LOW-BYTE VON $C00D
STA $0314   ;NACH $0314
LDA #$C0    ;HIGH-BYTE VON $C00D
STA $0315   ;NACH $0315
BRK         ;ENDE DER INITIALISIERUNG
```

Falsch!!!
Zwar nicht allzu oft, jedoch ab und zu wird der Rechner "abstürzen", wenn wir so vorgehen. Theoretisch könnte genau dann ein Timer-IRQ stattfinden, wenn wir das Low-Byte des Vektors geändert haben.

Da das High-Byte noch unverändert ist ($EA), weist der IRQ-Vektor auf die Adresse $EA0D. Der indirekte Sprung beim Timer-IRQ führt somit zu dieser Adresse.

Was sich dort befindet, wissen wir jedoch nicht. Höchstwahrscheinlich kein allzu sinnvolles Programm.

Bevor wir diesen Vektor verändern, müssen wir daher sicherstellen, daß kein IRQ erfolgen kann. Den IRQ kann man (im Gegensatz zum NMI) "ausschalten", und zwar mit dem Befehl SEI ("set interrupt mask", "setze die Interrupt-Maske").

Das "Interrupt-Flag" des Status-Registers (Bit 2) wird auf 1 gesetzt, was bedeutet, daß folgende Timer- oder BRK-IRQ's nicht ausgeführt werden. Wir haben den IRQ "gesperrt" und können den IRQ-Vektor verändern, ohne befürchten zu müssen, daß

genau in diesem Moment ein Interrupt erfolgt (zumindest kein IRQ, höchstens ein NMI, der jedoch über einen anderen Vektor "springt" und daher nicht zum Absturz führt).

```
SEI          ;IRQ VERHINDERN
LDA #$0D     ;LOW-BYTE VON $C00D
STA $0314    ;NACH $0314
LDA #$C0     ;HIGH-BYTE VON $C00D
STA $0315    ;NACH $0315
CLI          ;IRQ WIEDER ZULASSEN
BRK
```

Nach dem "Verbiegen" des Vektors wird mit dem Befehl CLI ("clear interrupt mask", "Unterbrechungsmaske löschen") das Interrupt-Flag gelöscht und damit IRQ's wieder zugelassen.

Nun wissen wir auch, warum die zweite Interrupt-Sorte, der NMI, eine höhere Priorität besitzt als der IRQ: weil ein NMI nicht verhindert werden kann (NMI = "non maskable interrupt").

Doch zurück zu unserer Interrupt-Routine. Beim nächsten IRQ wird nun zur Adresse $C00C gesprungen. Dort befindet sich jedoch noch kein Programm.

Also überlegen wir uns, welche Programmfunktion jede sechzigstel Sekunde ausgeführt werden soll. Zum "Aufwärmen" zuerst etwas sehr einfaches: jede sechzigstel Sekunde wird das Zeichen A in die obere linke Bildschirmecke geschrieben. Geben Sie bitte ein:

```
Die erste Interrupt-Routine

IRQ-Vektor "verbiegen"
      A C000 78               SEI          ;IRQ SPERREN
      A C001 A9 0D            LDA #$0D     ;LOW-BYTE VON $C00D
      A C003 8D 14 03         STA $0314    ;NACH IRQ-VEKTOR LOW
      A C006 A9 C0            LDA #$C0     ;HIGH-BYTE VON $C00D
      A C008 8D 15 03         STA $0315    ;NACH IRQ-VEKTOR HIGH
      A C00B 58               CLI          ;IRQ ZULASSEN
      A C00C 00               BRK          ;ENDE
```

```
Interrupt-Routine
      A C00D A9 01            LDA #$01     ;ASCII-CODE VON 'A'
      A C00F 8D 00 04         STA $0400    ;IN OBERE LINKE ECKE
      A C012 4C 31 EA         JMP $EA31    ;=>SERVICE-ROUTINE
```

Die eigentliche Interrupt-Routine beginnt ab $C00D, unmittelbar hinter der Initialisierung, die den IRQ-Vektor auf die Adresse der Routine verbiegt.

Die Interrupt-Routine gibt ein A in der oberen linken Bildschirmecke aus ($0400) und springt anschließend zur normalen Service-Routine an Adresse $EA31.

Um das Programm zu starten, rufen Sie die Initialisierungs-Routine mit G C000 auf. Die Interrupt-Routine ist nun eingebunden und wird automatisch jede sechzigstel Sekunde ausgeführt.

Das Resultat: das Zeichen A befindet sich immer in der ersten Spalte der ersten Bildschirmzeile, selbst wenn Sie versuchen, es zu überschreiben oder mit der Taste CLR den kompletten Bildschirm löschen.

Der Grund: da das A jede sechzigstel Sekunde - also in extrem kurzen Abständen - ausgegeben wird, ist es für das menschliche Auge sofort nach dem Löschen wieder vorhanden.

Bereits mit diesem wohl denkbar anspruchlosen Interrupt-Programm lassen sich Uneingeweihte verblüffen, denen Sie die Aufgabe stellen, das A zu löschen.

Um die Interrupt-Routine wieder auszuschalten, drücken Sie gleichzeitig die Tasten STOP und RESTORE, mit denen eine Betriebssystem-Routine aufruft, die verschiedene Zeiger (unter anderem den IRQ-Vektor) neu einrichtet und somit unseren Eingriff rückgängig macht.

Der NMI wurde bisher vernachlässigt. Diese Unterbrechungsanforderung ist für uns in der Praxis weniger interessant als der IRQ, der von den Timern automatisch jede sechzigstel Sekunde

ausgelöst wird. Eine "NMI-Quelle" ist zum Beispiel die RESTORE-Taste. Diese Taste löst bei jeder Betätigung einen NMI aus.

In den NMI können Sie eigene Routinen "einbinden", indem Sie den NMI-Vektor verbiegen ($0318/$0319).

Am Ende Ihrer Routine muß sich wiederum ein Sprung zur normalen NMI-Routine befinden, nämlich zu $8E60.

Mit Hilfe des NMI können Sie jederzeit eine eigene Routine aufrufen, auf die der NMI-Vektor zeigt.

Übungsprogramme zur Interrupt-Programmierung

Für die folgenden Übungsprogramme verwenden wir zwei Routinen zum "Verbiegen" des IRQ-Vektors. Eine Routine, die unsere Interrupt-Routine einschaltet, und eine zweite, die sie ausschaltet, indem sie den ursprünglichen Inhalt des IRQ-Vektors wiederherstellt.

Um beide Routinen von BASIC aus mit SYS aufrufen zu können, enden Sie nicht mit BRK, sondern mit RTS.

```
Ein-/Ausschalten von Interrupt-Routinen

Interrupt-Routine einschalten
      A C000 78               SEI          ;IRQ SPERREN
      A C001 A9 1A            LDA #$1A     ;VEKTOR IN
      A C003 8D 14 03         STA $0314    ;$0314/$0315
      A C006 A9 C0            LDA #$C0     ;AUF
      A C008 8D 15 03         STA $0315    ;$C01A VERBIEGEN
      A C00B 58               CLI          ;IRQ ZULASSEN
      A C00C 60               RTS
Interrupt-Routine ausschalten
      A C00D 78               SEI          ;IRQ SPERREN
      A C00E A9 31            LDA #$31     ;VEKTOR IN
      A C010 8D 14 03         STA $0314    ;$0314/$0315
      A C013 A9 EA            LDA #$EA     ;WIEDER AUF
```

```
      A C015 8D 15 03          STA $0315    ;$EA31 (SERVICE-ROUTINE)
      A C018 58                CLI          ;IRQ ZULASSEN
      A C019 60                RTS
Interrupt-Routine
      A C01A
```

Mit SYS 49152 ($C000) wird die nun ab $C01A beginnende Interrupt-Routine eingeschaltet und mit SYS 49165 ($C00D) wieder ausgeschaltet.

Blinkende Bildschirmzeile

Was hätten Sie den nun gerne jede sechzigstel Sekunde ausgeführt? Wie wär's - um "gemütlich" zu beginnen - mit der altbekannten "blinkenden Bildschirmzeile", diesmal jedoch als Interrupt-Routine und damit um einiges effektvoller?

Dieses Programm demonstriert ein typisches Problem aller Interrupt-Routinen. Die Invertierung/Normalisierung wird wie üblich mit einer Schleife und dem EOR-Befehl vorgenommen.

```
            LDX #$27
(ADRESSE)   LDA $0400,X
            EOR #$80
            STA $0400,X
            DEX
            BPL (ADRESSE)
            JMP $EA31
```

Obwohl es nur jede sechzigstel Sekunde aufgerufen wird, ist dieses Programm jedoch viel zu schnell. Die Bildschirmzeile blinkt nicht, sie flimmert.

Bei einem "normalen" Programm würden wir dieses Problem mit einer Warteschleife lösen, nicht jedoch bei einer Interrupt-Routine!

Interrupt-Routinen dürfen nicht zu lang sein (Zeitdauer, nicht Programmlänge). Sonst wird noch während des Interrupts der

nächste Timer-IRQ ausgelöst, und daß dann "Unfug" entstehen muß, ist einsichtig. Also müssen wir unser Geschwindigkeitsproblem auf andere Weise lösen, am besten so, wie es uns die Service-Routine beim Blinken des Cursors vormacht.

Der Cursorzustand (invertiert/normalisiert) wird nicht bei jedem IRQ "umgedreht", sondern nur bei jedem zwanzigsten.

Wir verwenden eine Speicherzelle als Zähler, und zwar $033C. Bei jedem Aufruf unserer Interrupt-Routine wird $033C dekrementiert. Ist das Ergebnis ungleich null, überspringen wir die Schleife und springen sofort zur Service-Routine nach $EA31.

Ist das Ergebnis hingegen null, initialisieren wir die Speicherzelle mit dem Wert 20 (=$14) und durchlaufen die "Blinkschleife".

IRQ-Routine "Blinkende Bildschirmzeile"

```
A C01A CE 3C 03        DEC $033C     ;$033C DEKREMENTIEREN
A C01D D0 12           BNE $C031     ;SPRUNG, WENN
                                     ;UNGLEICH NULL
A C01F A9 14           LDA #$14      ;SONST MIT $14
A C021 8D 3C 03        STA $033C     ;NEU INITIALISIEREN
A C024 A2 27           LDX #$27      ;UND DIE OBERSTE
A C026 BD 00 04        LDA $0400,X   ;BILDSCHIRMZEILE MIT $80
A C029 49 80           EOR #$80      ;'EOREN', DAS HEISST
A C02B 9D 00 04        STA $0400,X   ;'UMDREHEN' (INVERTIEREN
A C02E CA              DEX           ;BZW.NORMALISIEREN)
A C02F 10 F5           BPL $C026
A C031 4C 31 EA        JMP $EA31     ;=> SERVICE-ROUTINE
```

Geben Sie das Programm ab Adresse $C01A ein, im Anschluß an den Initialisierungsteil.

Nach der Einbindung in den Systeminterrupt mit SYS 49152 vergehen eventuell bis zu vier Sekunden, bis die Routine "loslegt", da ja der Ausgangswert des Zählers $033C beim ersten Durchgang unbestimmt ist.

Enthält er den Maximalwert $FF, wird die Bildschirmzeile erst nach dem 255ten IRQ invertiert (255*1/60 sec = 4.25 sec). Danach jedoch blinkt die Zeile im gleichen zeitlichen Rhytmus wie der Cursor.

Wenn Sie wollen, können Sie die Initialisierungs-Routine ändern, so daß in dieser der Zähler $033C mit $01 geladen wird. Dann erfolgt die erste Invertierung unmittelbar nach dem Aufruf.

Die geradezu "märchenhaften" Eigenschaften von Interrupt-Routinen werden deutlich, wenn Sie nun zum Beispiel ein BASIC-Programm schreiben und starten. Dies alles stört die blinkende Bildschirmzeile nicht im geringsten.

Während "im Hintergrund" ein Interrupt-Programm läuft, können Sie "im Vordergrund" ganz normal mit dem Rechner arbeiten. Für den ahnungslosen Benutzer hat es den Anschein, als würden zwei Programme gleichzeitig bearbeitet werden.

Sie wissen nun jedoch, daß das "Hintergrund-Programm" tatsächlich nur jede sechzigstel Sekunde "im Vorbeigehen" bearbeitet wird. Ich kenne keine andere Methode, die die unglaubliche Geschwindigkeit von Assembler so eindrucksvoll dokumentiert wie eine Interrupt-Routine.

Belegung der Funktionstasten

Bei Betätigung einer Funktionstaste wird sofort die zugehörige Zeichenkette ab der aktuellen Cursorposition ausgegeben. Diese "Funktionstastenbelegung" ist sehr nützlich, da ein Tastendruck genügt, um häufig benötigte Befehle wie LOAD, SAVE, RUN oder LIST einzugeben.

Mit einem geeigneten Interrupt-Programm können wir dem C64 diese Fähigkeit verleihen. Die Interrupt-Routine wird bei jedem Timer-IRQ aufgerufen und fragt die Tastatur ab. Wurde eine Funktionstaste betätigt, wird die zugehörige Zeichenkette ausgegeben.

Bevor wir mit der Programmerstellung beginnen, sollte jedoch noch ein Problem gelöst werden: Angenommen, wir schreiben ein BASIC-Programm, das die Funktionstasten zum Auslösen bestimmter Funktionen benutzt. Ein Beispiel:

```
100 GET A$:IF A$="" THEN GOTO 100:REM WARTESCHLEIFE
110 IF A$=CHR$(133) THEN GOTO 500:REM F1 GEDRUECKT
110 IF A$=CHR$(134) THEN GOTO 500:REM F2 GEDRUECKT
...
...
...
```

Die Funktionstasten werden beim C64 innerhalb eines Programms benutzt, indem die ihnen zugeordneten ASCII-Codes 133 bis 140 wie im Beispiel abgefragt werden. Die im Hintergrund "lauernde" Interrupt-Routine schreibt wie erläutert bei jeder Betätigung einer Funktionstaste die zugeordnete Zeichenkette auf den Bildschirm, was beim Ablauf eines BASIC-Programms äußerst störend wäre.

Um die normale Funktion dieser Tasten nicht zu beeinflussen, wird im folgenden Programm immer eine Tastenkombination abgefragt. Die Interrupt-Routine wird nur dann "aktiv", wenn gleichzeitig mit einer Funktionstaste die "Commodore-Taste" betätigt wird. Die "einfache" Betätigung einer Funktionstaste läuft daher weiterhin ab wie gewohnt, das heißt der ASCII-Code der gedrückten Taste wird übergeben (GET A$), weitere Aktionen erfolgen nicht.

Auf diese Weise wird ausgeschlossen, daß die Interrupt-Routine ein BASIC-Programm "stört", das die Funktionstasten wie gezeigt zur Steuerung des Programmablaufs verwendet.

Die zu erstellende Routine ist leider recht komplex. Zur Tastaturabfrage können wir nicht einfach wie gewohnt die Routine GETIN verwenden. Auch BSOUT zur Ausgabe der jeweiligen Zeichenkette ist nicht zulässig. Beide Betriebssystem-Routinen sind zu "langsam", um in einer Interrupt-Routine verwendet zu werden. Selbst wenn wir GETIN benutzen könnten, wären wir

dadurch noch nicht in der Lage, festzustellen, ob der Benutzer gleichzeitig die Commodore-Taste betätigt.

Die Tastaturabfrage lösen wir wie folgt:

1. Abfrage der Commodore-Taste: Der Inhalt der Speicherzelle $028D gibt an, ob momentan eine der Sondertasten SHIFT, CTRL oder die Commodore-Taste gedrückt ist. Wenn die Commmdore-Taste gedrückt wird, enthält $028D den Wert $02 (bei SHIFT den Wert $01 und bei CTRL den Wert $04).
   Wird unsere IRQ-Routine aufgerufen, vergleichen wir somit $028D mit $02. Enthält $028D einen anderen Wert, ist die Commodore-Taste nicht gedrückt und es wird zur normalen Service-Routine verzweigt.
2. Abfrage der Funktionstasten: Wenn die Commodore-Taste betätigt wird, prüfen wir, ob gleichzeitig auch eine Funktionstaste gedrückt wird. Die Speicherzelle $C5 gibt uns über die momentan gedrückte Taste Auskunft.
   Der Inhalt $03 entspricht der Taste F7, $04 der Taste F1, $05 der Taste F3 und $06 der Funktionstaste F5.

Wie 2. zeigt, beschränken wir uns auf die Abfrage - und Belegung - von vier Funktionstasten: F1, F3, F5 und F7.

```
Tastaturabfrage
      A C01A AD 8D 02       LDA $028D     ;'CTRL-FLAG' HOLEN
      A C01D C9 02          CMP #$02      ;INHALT $02 = CTRL
                                          ;GEDRUECKT
      A C01F D0 30          BNE $C051     ;SPRUNG, WENN NICHT
                                          ;GEDRUECKT
      A C021 A5 C5          LDA $C5       ;SONST TASTE HOLEN
                                          ;UND PRUEFEN,
      A C023 C9 03          CMP #$03      ;OB ES SICH UM EINE
                                          ;FUNKTIONSTASTE
      A C025 90 2A          BCC $C051     ;HANDELT ($03, $04, $05
      A C027 C9 07          CMP #$07      ;ODER $06)
      A C029 B0 26          BCS $C051     ;SPRUNG, WENN KEINE
                                          ;FUNKTIONSTASTE
```

```
A C02B CD 57 C0          CMP $C057    ;MIT LETZTER TASTE
                                      ;VERGLEICHEN
A C02E F0 21             BEQ $C051    ;SPRUNG, WENN GLEICH
```

Die Tastaturabfrage verzweigt in verschiedenen Fällen zu $C051, wo sich der Befehl JMP $EA31 befindet, also der Einsprung in die Service-Routine des Betriebssystems.

Die Verzweigung erfolgt, wenn $028D nicht den Wert $02 enthält, also CTRL nicht betätigt wird.

Sie erfolgt ebenfalls, wenn $C5 einen Wert enthält, der kleiner ist als $03 oder größer als $07. In beiden Fällen wird keine der vier Funktionstasten betätigt.

Die beiden letzten Befehle (CMP $C057 : BEQ $C051) können Sie nicht verstehen, ohne zu wissen, daß am Ende unserer Routine der Inhalt von $C5 - die momentan betätigte Taste - in die Speicherzelle $C057 kopiert wird. In $C057 befindet sich somit bei jedem Aufruf der Code jener Taste, die während des letzten Aufrufs gedrückt wurde.

Ist die momentan gedrückte Taste mit der zuletzt betätigten identisch, wird ebenfalls sofort zur Service-Routine verzweigt, obwohl momentan sowohl eine Funktions- als auch die Commodore-Taste betätigt wird.

Der Grund: Wie wir wissen, wird eine IRQ-Routine jede sechzigstel Sekunde aufgerufen. Da eine Taste normalerweise jedoch länger betätigt wird, wird während einer Tastenbetätigung unsere Routine mehrmals aufgerufen und gibt die zugehörige Zeichenkette ebenfalls mehrmals aus.

Daher überprüfen wir diesen Spezialfall - beim vorigen Aufruf gedrückte Taste ist mit der momentan gedrückten identisch - und übergehen die Ausgabe der Zeichenkette, wenn wir wissen, daß diese offenbar bereits beim letzten Aufruf ausgegeben wurde.

Der folgende Programmteil soll die auszugebenden Zeichen ermitteln. Die Zeichenketten selbst befinden sich im ASCII-Format am Programmende und beginnen an Adresse $C058.

Jede Zeichenkette wird von der vorhergehenden durch die "Endemarke" $00 getrennt.

Der zweite Programmteil liest diese Zeichenketten ein, bis die Startposition der auszugebenden Zeichen gefunden wurde.

Um den folgenden Programmteil zu verstehen, müssen Sie wissen, daß im "Funktionstastenspeicher" am Programmende als erstes die der Taste F7 zugeordnete Zeichenkette abgelegt ist und dieser folgend die den Tasten F1, F3 und F5 zugeordneten Zeichenketten.

```
Position der Zeichenkette ermitteln
      A C030 A0 00            LDY #$00     ;STARTWERT FUER Y-REGISTER
      A C032 A2 03            LDX #$03     ;STARTWERT FUER X-REGISTER
      A C034 E4 C5            CPX $C5      ;X MIT ZEICHENCODE
                                           ;VERGLEICHEN
      A C036 F0 09            BEQ $C041    ;SPRUNG, WENN GLEICH
      A C038 C8               INY          ;SONST Y ERHOEHEN
      A C039 B9 58 C0         LDA $C058,Y  ;ZEICHEN LESEN
      A C03C D0 FA            BNE $C038    ;UND WEITERLESEN,
                                           ;WENN NICHT $00
      A C03E E8               INX          ;WENN $00 GELESEN,
                                           ;X INKREMENTIEREN
      A C03F D0 F3            BNE $C034    ;UND VERZWEIGEN
```

Bei diesem zweiten Programmabschnitt handelt es sich um eine relativ schwer zu duchschauende Schachtelung zweier Schleifen. Zu Beginn der äußeren Schleife wird Y mit $00 und X mit $03 (also dem Code der Funktionstaste F7) geladen.

X wird mit dem Inhalt von $C5 verglichen. Wird Gleichheit festgestellt - enthält also $C5 den Wert $03 - wird verzweigt. Zeichenketten am Programmende werden nicht weiter durch-

sucht, da sich die Zeichenkette, die der Taste F7 zugeordnet ist, am Anfang des "Funktionstastenspeichers" befindet.

Ansonsten wird der "Funktionstastenspeicher" bis zum Ende der aktuellen Zeichenkette abgesucht, bis zur "Endemarke" $00. Wird diese Endemarke erreicht, ist die Bedingung BNE ("verzweige, wenn ungleich null") nicht erfüllt und die innere Schleife, die Zeichen für Zeichen liest, wird verlassen.

Der folgende INX-Befehl inkrementiert das X-Register, bevor zum Beginn der äußeren Schleife verzweigt wird. X enthält nun den Wert $04, der dem Code entspricht, mit dem die Taste F1 in der Speicherzelle $C5 dargestellt wird. Das Y-Register wurde bei jedem Durchgang der inneren Schleife inkrementiert und weist nun auf den Beginn der zweiten Zeichenkette, genauer: Auf das Trennbyte $00, das sich vor dem ersten Zeichen der zweiten Zeichenkette befindet.

Wenn der Inhalt von $C5 mit dem X-Wert $04 identisch ist, wird die "Suchschleife" verlassen und nach $C041 verzweigt, ansonsten wird die momentan untersuchte Zeichenkette wiederum bis zu ihrem Ende gelesen und unser "Funktionstastenzähler" X inkrementiert.

Diese sehr komplexe Routine führt letztendlich zu folgendem Ergebnis: Y weist auf das Trennbyte $00, das sich vor jener Zeichenkette befindet, die der gedrückten Funktionstaste zugeordnet ist.

Der dritte Programmteil gibt diese Zeichenkette auf dem Bildschirm aus. BSOUT können wir wie erläutert nicht verwenden. Direkt in den Bildschirmspeicher zu schreiben, bereitet ebenfalls Probleme, da wir zuvor die aktuelle Cursorposition ermitteln müßten.

Eine recht trickreiche Lösung besteht darin, die Zeichenkette in den "Tastaturpuffer" des C64 zu schreiben. In diesem Tastaturpuffer speichert unser Rechner alle noch nicht verarbeiteten Zeichen. Also "tun wir so", als wären die auszugebenden Zeichen

zwar vom Benutzer eingetippt, jedoch noch nicht verarbeitet und auf dem Bildschirm ausgegeben worden.

Der Tastaturpuffer beginnt beim C64 ab Adresse $0277.

Der folgende Programmteil liest ab der durch den Inhalt des Y-Registers angegebenen Position in unserem "Funktionstastenspeicher" Zeichen für Zeichen ein und schreibt sie in den Tastaturpuffer.

```
Zeichenkette in Tastaturpuffer übertragen

A C041 A2 00          LDX #$00      ;STARTWERT 1 FUER DAS
A C043 E8             INX           ;X-REGISTER
A C044 C8             INY           ;Y AUF 1.ZEICHEN
                                    ;HINTER $00
A C045 B9 58 C0       LDA $C058,Y   ;ZEICHEN LESEN UND IN DEN
A C048 9D 76 02       STA $0276,X   ;TASTATURPUFFER SCHREIBEN
A C04B D0 F6          BNE $C043     ;FERTIG, WENN $00 GELESEN
```

Der letzte Programmteil teilt dem Betriebssystem mit, daß sich im Tastaturpuffer 'X' noch auszugebende Zeichen befinden. Die Speicherzelle $C6 gibt an, wie viele noch zu bearbeitende Zeichen der Tastaturpuffer enthält. In diese Speicherzelle wird der aktuelle X-Wert übertragen.

Wie bereits erwähnt wurde, wird zuletzt der in $C5 enthaltene Tastencode für den nächsten Aufruf der IRQ-Routine nach $C057 kopiert, bevor mit JMP $EA31 die normale Service-Routine aufgerufen wird.

```
Programm beenden

A C04D 86 C6          STX $C6       ;ANZAHL ZEICHEN
                                    ;IM TAST.PUFFER
A C04F A5 C5          LDA $C5       ;AKTUELLEN TASTENCODE NACH
A C051 8D 57 C0       STA $C057     ;$C057 KOPIEREN
A C054 4C 31 EA       JMP $EA31     ;=> SERVICE-ROUTINE
```

Das Programm selbst ist beendet, es fehlen jedoch noch die Zeichenketten, mit denen die Funktionstasten belegt sind. Diese beginnen ab Adresse $C058 oder dezimal 49240.

Verwenden Sie zur Ablage der Zeichenketten das folgende BASIC-Programm.

```
100 A$=CHR$(0)+"RUN"+CHR$(0)+"LIST"+CHR$(0)+"LOAD"+CHR$(0)+"SAVE"+
CHR$(0)
110 FOR I=1 TO LEN(A$)
120 : POKE I+49239,ASC(MID$(A$,I,1))
130 NEXT
```

Dieses Programm belegt F7 mit der Zeichenkette "RUN", F1 mit "LIST", F3 mit "LOAD" und F5 mit "SAVE". Die Belegung können Sie selbstverständlich ändern. Dabei müssen Sie jedoch berücksichtigen, daß der Tastaturpuffer nur zehn Zeichen umfaßt und daher keine Zeichenkette länger als zehn Zeichen sein darf.

Nach dem Starten des BASIC-Programms rufen Sie bitte wieder den Monitor auf und speichern Sie den Bereich $C000 bis $C0A0 unter dem Namen "F-KEYS" ab. Außer dem eigentlichen Programm wird auch die Funktionstastenbelegung gespeichert.

Bei einer späteren Verwendung der Routine laden Sie das Assembler-Programm und initialisieren Sie die Interrupt-Routine mit SYS 49152. Mit SYS 49165 kann die Belegung jederzeit ausgeschaltet werden.

Denken Sie bitte daran, daß es nicht ausreicht, eine der vier Funktionstasten zu betätigen. Gleichzeitig müssen Sie die Commodore-Taste drücken.

Bevor Sie das folgende vollständige Programm-Listing eingeben, erlauben Sie mir noch eine Bemerkung. Dieses Programm ist zweifellos erheblich schwerer zu verstehen als alle vorangegangenen Demoprogramme. "Pädagogisch gesehen" gehört es zweifellos nicht in ein Lehrbuch, das mit den Grundlagen der Maschinensprache beginnt.

Der Sinn dieses Programms besteht vor allem darin, Sie zu eigenen Experimenten mit Interrupt-Routinen zu motivieren. Das letzte Demoprogramm bewirkte das Blinken einer Bildschirmzeile. Dieses Programm war problemlos zu verstehen und zeigte die bei der Interrupt-Programmierung zu beachtenden Prinzipien auf.

Eine sinnvolle Anwendung war es jedoch nicht, sondern fiel eher unter die Rubrik "Spielerei".

In diese Abteilung fällt ein großer Teil der in diversen Fachzeitschriften veröffentlichten IRQ-Routinen, zum Beispiel die immer wiederkehrenden "Laufschrift"-Programme.

Das Demoprogramm "Funktionstastenbelegung" sollte Ihnen zeigen, daß mit ein wenig Phantasie äußerst nützliche IRQ-Routinen vorstellbar sind. Das vorgestellte Programm erleichtert die BASIC-Programmierung derart, daß Sie sich schnell angewöhnen werden, es nach jedem Einschalten des Rechners als erstes Programm zu laden.

Noch ein Tip: Da dieses Programm vor allem bei der BASIC-Programmierung verwendet wird und daher nur in seltenen Fällen zuvor ein Monitor geladen wurde, müssen Sie die Routine "von BASIC aus" laden.

Geben Sie nach dem Einschalten des Rechners (oder wann immer Sie die Routine laden wollen) ein: LOAD"F-KEYS",8,1 und nach dem Beenden des Ladevorgangs NEW. Mit diesen beiden Befehlen wird der LOAD-Befehl des Monitors ersetzt, das Programm kann von BASIC aus geladen werden (vor dem BASIC-Programm, das durch NEW gelöscht würde).

Nachfolgend finden Sie das Listing des kompletten Programms. Nicht abgedruckt ist die im vorigen Kapitel erläuterte Initialisierungs-Routine, die ab $C000 einzugeben ist.

Das vollständige Programm-Listing:

```
Tastaturabfrage
      A C01A AD 8D 02           LDA $028D     ;'CTRL-FLAG' HOLEN
      A C01D C9 02              CMP #$02      ;INHALT $02 =
                                              ;CTRL GEDRUECKT
      A C01F D0 30              BNE $C051     ;SPRUNG, WENN NICHT
                                              ;GEDRUECKT
      A C021 A5 C5              LDA $C5       ;SONST TASTE HOLEN
                                              ;UND PRUEFEN,
      A C023 C9 03              CMP #$03      ;OB ES SICH UM EINE
                                              ;FUNKTIONSTASTE
      A C025 90 2A              BCC $C051     ;HANDELT ($03, $04, $05
      A C027 C9 07              CMP #$07      ;ODER $06)
      A C029 B0 26              BCS $C051     ;SPRUNG, WENN
                                              ;KEINE FUNKTIONSTASTE
      A C02B CD 57 C0           CMP $C057     ;MIT LETZTER TASTE
                                              ;VERGLEICHEN
      A C02E F0 21              BEQ $C051     ;SPRUNG, WENN GLEICH
Position der Zeichenkette ermitteln
      A C030 A0 00              LDY #$00      ;STARTWERT FUER Y-REGISTER
      A C032 A2 03              LDX #$03      ;STARTWERT FUER X-REGISTER
      A C034 E4 C5              CPX $C5       ;X MIT ZEICHENCODE
                                              ;VERGLEICHEN
      A C036 F0 09              BEQ $C041     ;SPRUNG, WENN GLEICH
      A C038 C8                 INY           ;SONST Y ERHOEHEN
      A C039 B9 58 C0           LDA $C058,Y   ;ZEICHEN LESEN
      A C03C D0 FA              BNE $C038     ;UND WEITERLESEN, WENN
                                              ;NICHT $00
      A C03E E8                 INX           ;WENN $00 GELESEN,
                                              ;X INKREMENTIEREN
      A C03F D0 F3              BNE $C034     ;UND VERZWEIGEN
Zeichenkette in Tastaturpuffer übertragen
      A C041 A2 00              LDX #$00      ;STARTWERT 1 FUER DAS
      A C043 E8                 INX           ;X-REGISTER
      A C044 C8                 INY           ;Y AUF 1.ZEICHEN
                                              ;HINTER $00
      A C045 B9 58 C0           LDA $C058,Y   ;ZEICHEN LESEN UND IN DEN
      A C048 9D 76 02           STA $0276,X   ;TASTATURPUFFER SCHREIBEN
      A C04B D0 F6              BNE $C043     ;FERTIG, WENN $00 GELESEN
```

```
Programm beenden
      A C04D 86 C6            STX $C6      ;ANZAHL ZEICHEN
                                           ;IM TAST.PUFFER
      A C04F A5 C5            LDA $C5      ;AKTUELLEN TASTENCODE NACH
      A C051 8D 57 C0         STA $C057    ;$C057 KOPIEREN
      A C054 4C 31 EA         JMP $EA31    ;=> SERVICE-ROUTINE
```

Teil 4: Zwei Welten kommen sich näher: BASIC und Assembler

Mehrmals in diesem Buch versprach ich Ihnen, auf ein Thema einzugehen, das leider in kaum einem Assembler-Lehrbuch beschrieben wird, auf die Zusammenarbeit von Assembler-Routinen und BASIC-Programmen. In folgenden Teil wird Ihnen das nötige Handwerkszeug vermittelt, um zum Beispiel Eingabe- oder Sortierroutinen in Assembler zu schreiben, die von einem BASIC-Programm genutzt werden.

Die in diesem dritten Teil entwickelten Programme sind deutlich komplexer als die vorangegangenen und werden deshalb in verschiedenen Teilschritten entwickelt und erläutert. Am Ende eines jeden Abschnitts wird das Listing des kompletten Programms vorgestellt.

Vor der Entwicklung diverser Programme ist jedoch ein wenig Theorie unumgänglich, da Sie unbedingt darüber Bescheid wissen müssen, wie der BASIC-Interpreter einen BASIC-Text liest und Variablen organisiert.

2.20 Wie BASIC-Programme abgelegt werden

BASIC-Programme legt der Interpreter in einer sehr speziellen Form im BASIC-RAM ab. Beim C64 beginnt der BASIC-Speicher ab $0400. An dieser Adresse befindet sich das Byte $00, an dem das folgende BASIC-Programm erkannt wird. Den Beginn jeder Programmzeile bilden zwei "Link-Bytes", die die Adresse (in der Form Low-Byte/High-Byte) der Link-Bytes der nächsten Programmzeile enthalten.

Über diese Link-Bytes kann sich der Interpreter sehr schnell durch ein BASIC-Programm "hangeln", ohne jeweils die komplette Programmzeile lesen zu müssen.

Den Link-Bytes folgt die Zeilennummer (ebenfalls zwei Bytes), der wiederum der eigentliche Programmtext folgt.

Der Programmtext wird keineswegs Zeichen für Zeichen im Speicher abgelegt, sondern platzsparender. Jedem BASIC-Befehle wird ein Ein-Byte-Wert zugeordnet, das sogenannte "Token", der den Befehl eindeutig identifiziert.

Statt PRINT wird daher das zugehörige Token abgelegt.

Abgesehen von den BASIC-Befehlen wird der restliche Programmtext in ASCII-Form Zeichen für Zeichen abgelegt. Das Ende einer Programmzeile wird immer am Code $00 erkannt.

Auf diese Weise wird Zeile für Zeile im Speicher abgelegt. Das Programmende markiert eine zusätzliche null ($00 $00).

Zum Beispiel wird beim C64 das Programm

```
100 PRINT "A"
110 PRINT "B"
```

wie folgt abgelegt:

```
0800   00 0B 08 64 00 99 20 22 => Link-B./Zeilennr./PRINT "
0808   41 22 00 15 08 6E 00 99 => A"/00/Link-B.Zeilennr./PRINT
0810   20 22 42 22 00 00       =>  "B"/00/00
```

Für Sie ist die Art und Weise, wie der Interpreter ein BASIC-Programm ablegt, interessant, wenn Sie zum Beispiel Hilfsprogramme wie einen "REM-Killer" schreiben wollen, ein Programm, das automatisch alle REM-Zeilen aus einem BASIC-Programm entfernt.

Variablenbehandlung durch den Interpreter

Die Art und Weise, wie der Interpreter ein BASIC-Programm im Speicher ablegt, wird uns nicht weiter beschäftigen. Außerordentlich wichtig für uns ist jedoch die Art und Weise, wie der Interpreter Variablen behandelt. Ohne dieses Wissen können wir niemals eine Sortierroutine schreiben, die zum Beispiel ein Stringarray bearbeitet (oder wissen Sie bereits, wo das Assembler-Programm die Strings zu suchen hat?).

Wir müssen drei Variablentypen unterscheiden: Integer-, Fließkomma- und Stringvariablen. Bei allen drei Typen sind wiederum einfache und Feld- oder "Array"-Variablen zu unterscheiden.

Einfache Variablentypen

Unmittelbar hinter einem BASIC-Programm befindet sich die sogenannte "Variablentabelle". Diese Tabelle enthält alle einfachen und dimensionierten numerischen Variablen (Integer/Fließkomma).

Jeder "Eintrag" in dieser Tabelle ist bei nicht dimensionierten Variablen genau sieben Byte lang. Die ersten beiden Byte (Byte 1 und 2) enthalten den jeweiligen Variablennamen und kennzeichnen zugleich den Variablentyp.

Integervariablen: Bit 7 der beiden "Namensbytes" sind gesetzt.
Fließkommavariablen: Bit 7 beider "Namensbytes" ist gelöscht.
Stringvariablen: Bit 7 ist nur im zweiten "Namensbyte" gesetzt.
Die restlichen fünf Bytes (Byte 3-7) besitzen folgende Bedeutung:

```
1. Integervariablen
Byte 3 und 4: Zahlenwert in der Form High-Byte/Low-Byte.
Byte 5-7: ungenutzt.
```

Beachten Sie bitte die ungewohnte Form, in der eine Integerzahl gespeichert wird. Zuerst kommt das High-Byte, anschließend das Low-Byte, nicht umgekehrt wie beim "Adreßformat".

```
2. Fließkommavariablen
Byte 3-7: Zahlenwert im sogenannten "Fließkomma-Format", in dem
eine Zahl durch fünf Byte dargestellt wird.

3. Stringvariablen
Byte 3: Stringlänge (0-255)
Byte 4-5: Zeiger auf die Adresse des Strings
Byte 6-7: ungenutzt
```

Die Strings selbst sind nicht in der Variablentabelle enthalten, sondern befinden sich am Ende des BASIC-RAM's.

Der erste String wird am Ende des BASIC-Speichers angelegt, der nächste String unmittelbar vor diesem und so weiter.

Man spricht auch vom "String-Stack", der nach unten - in Richtung niedrigerer Adressen - "wächst".

Die Variablentabelle enthält ein Byte (Byte 3), das Auskunft über die Länge des Strings gibt, und zwei weitere Byte (4 und 5), die einen Zeiger auf den String darstellen (im Adreßformat Low-Byte/High-Byte).

Daß die Strings nicht ebenso wie numerische Variablen in der Variablentabelle enthalten sind, sondern nur die sogenannten "Stringdescriptoren", hat einen einleuchtenden Grund.

Wie erläutert beginnt die Variablentabelle unmittelbar hinter dem BASIC-Programm. Angenommen, eine Programmzeile wird gelöscht. Dann muß die gesamte Tabelle nach "unten" verschoben werden, hinter das neue Ende des BASIC-Programms, eine langwierige Aufgabe, wenn die Tabelle eine Vielzahl großer Zeichenketten enthält.

Daher befinden sich die Strings am "entgegengesetzten" Ende des Speichers, um bei Änderungen der Variablentabelle oder des BASIC-Programms Zeitprobleme durch das Verschieben umfangreicher Zeichenketten zu vermeiden.

Arrayvariablen

Arrayvariablen werden in einer "komprimierten" Form gespeichert. Da die Namen von Arrayvariablen (bis auf den Index) immer identisch sind, wäre die Speicherung von jeweils zwei "Namensbyte" für jede Arrayvariable eine enorme Platzverschwendung.

Es bietet sich an, nach einem DIM-Befehl sofort Platz für alle Variablen in der Tabelle zu "reservieren" und nur einmal - am Anfang des reservierten Bereichs - den Namen des Arrays zu speichern.

Genau so geht der BASIC-Interpreter tatsächlich vor. Der Name wird nur einmal am Arrayanfang vermerkt, wobei für die beiden "Namensbyte" die gleichen Konventionen wie bei einfachen Variablen gelten. Der "Arraykopf" enthält außer dem Arraynamen noch zwei Byte, die den Speicherplatz angeben, den das Array belegt, ein Byte mit der jeweiligen Dimensionierung, und zwei Byte, die die Anzahl der Arrayelemente enthalten. Erst nach dieser "Arraybeschreibung" kommen die eigentlichen Arrayvariablen.

1. Integervariablen
Für dimensionierte Integervariablen werden nur zwei Byte benötigt, das High-Byte und das Low-Byte der betreffenden Zahl. Diesen zwei Byte folgt unmittelbar die nächste Integervariable des Arrays.

2. Fließkommavariablen
Den ersten fünf Byte für die erste Fließkommavariable folgt sofort die zweite Variable und so weiter.

3. Stringvariablen
Dimensionierte Stringvariablen benötigen pro Eintrag in der Variablentabelle drei Byte, ein Byte für die Stringlänge und zwei Byte für den Zeiger auf den String.

Interpreter-Routinen zur Parameterübergabe

Soviel zur Theorie, wenden wir uns nun der Praxis zu. Sollen Assembler-Routinen mit BASIC-Programm zusammenarbeiten, muß das BASIC-Programm meist "Parameter" übergeben.

Ein Beispiel: eine oftmals sehr nützliche Assembler-Routine ist ein "Invertier-Routine", die in der Lage ist, beliebige Bildschirmausschnitte zu invertieren und wieder zu normalisieren.

Das BASIC-Programm muß angeben können, wo die Invertierung/Normalisierung beginnen soll (Spalte/Zeile).

Als weiterer Parameter ist die "Invertierlänge" anzugeben, die Anzahl der folgenden Bildschirmzellen, die zu behandeln sind.

Erst nach Übergabe dieser Parameter kann das Assembler-Programm seine Arbeit aufnehmen.

```
SYS (STARTADRESSE),(SPALTE),(ZEILE),(ZEICHENANZAHL)
```

Ein typischer Aufruf: SYS 49152,5,10,100

Dieser Aufruf bedeutet für die Assembler-Routine, daß ab Spalte fünf von Zeile zehn die folgenden 100 Zeichen zu invertieren sind (beziehungsweise zu normalisieren, wenn sie bereits invertiert sind).

Der BASIC-Interpreter stellt uns mehrere Routinen zur Verfügung, mit denen wir Zahlenwerte aus einem BASIC-Text einlesen können. Wie bei allen weiteren Routinen des Interpreters sind die "Einsprungadressen" - im Gegensatz zu den Betriebssystem-Routinen - von Rechner zu Rechner unterschiedlich.

Soweit sie mir bekannt sind, gebe ich für alle Rechner die betreffenden Adressen an.

GETBYT und CHKKOM

Die grundlegenden Routinen zur Parameterübergabe nennen sich CHKKOM und GETBYT. Mit diesen Routinen können wir bereits einfachere Assembler-Routinen für BASIC-Programme schreiben.

```
CHKKOM ($AEFD)
```

Während ein BASIC-Programm bearbeitet wird, läuft ständig ein Zeiger mit, der die Adresse des Zeichens angibt, das gerade gelesen wurde.

Nach dem Aufruf der Assembler-Routine mit SYS (START-ADRESSE) weist dieser Zeiger auf das der Adresse folgende Komma. Die Routine CHKKOM liest das nächste Zeichen aus dem BASIC-Text und prüft, ob es sich um ein Komma handelt. Wenn ja, wird der "Textzeiger" inkrementiert und weist auf das nächste Zeichen. Wenn nein, wird die Fehlermeldung "Syntax error" ausgegeben.

Mit CHKKOM können wir daher sehr elegant Kommata zur Trennung der verschiedenen Übergabe-Parameter verwenden. Nach dem Aufruf der Assembler-Routine und nach dem Einlesen eines Parameters lesen wir mit CHKKOM das Trennzeichen. Tippfehler des Benutzers (fehlendes Komma, anderes Trennzeichen etc.) werden automatisch mit einem "SYNTAX ERROR" quittiert.

GETBYT ($B79E)

GETBYT liest einen Ein-Byte-Wert aus dem BASIC-Text und übergibt ihn im X-Register. Um den im Beispiel verwendeten ersten Parameter fünf einzulesen, würden wir daher die Befehlsfolge verwenden:

```
JSR $AEFD ;CHKKOM AUFRUFEN
JSR $B79E ;GETBYT AUFRUFEN
```

CHKKOM liest das Komma ein und inkrementiert den Textzeiger. GETBYT liest den folgenden Bytewert und übergibt ihn unserem Programm im X-Register (X=$05). Da GETBYT den Textzeiger nach jedem gelesenen Zeichen ebenfalls inkrementiert, weist er nach der Übergabe des Wertes auf das folgende Komma und wir können im Anschluß an GETBYT sofort wieder CHKKOM aufrufen.

Invertier-Routine für BASIC-Programme

Wir kennen nun bereits die benötigten Interpreter-Routinen, um alle Parameter aus dem BASIC-Text zu lesen, die die Invertier-Routine benötigt. Der Aufruf:

```
SYS (STARTADRESSE),(SPALTE),(ZEILE),(ZEICHENANZAHL)
```

```
Spalte und Zeile lesen, Cursor auf Ausgangsposition setzen
      A C000 20 FD AE          JSR $AEFD    ;KOMMA LESEN
      A C003 20 9E B7          JSR $B79E    ;SPALTE NACH X HOLEN
      A C006 8A                TXA          ;SPALTE NACH AKKU BRINGEN
      A C007 48                PHA          ;UND AUF STACK RETTEN
      A C008 20 FD AE          JSR $AEFD    ;KOMMA LESEN
      A C00B 20 9E B7          JSR $B79E    ;ZEILE NACH X HOLEN
      A C00E 68                PLA          ;SPALTE VOM STACK ZIEHEN
      A C00F A8                TAY          ;UND NACH Y BRINGEN
      A C010 18                CLC          ;'PLOT' VORBEREITEN
      A C011 20 F0 FF          JSR $FFF0    ;CURSOR SETZEN
```

Dieser erste Programmteil soll den Cursor auf die Ausgangsposition der Invertierung/Normalisierung setzen. Wie wir wissen, benötigt die PLOT-Routine als Parameter die Spalte im Y-Register und die Zeile im X-Register.

Zuerst liest CHKKOM das Komma hinter der Startadresse, anschließend wird der erste Ein-Byte-Wert (Spalte) mit GETBYT aus dem BASIC-Text geholt. GETBYT übergibt den gelesenen Wert immer im X-Register. Da die Registerinhalte durch die

folgenden Aufrufe von CHKKOM und GETBYT verändert werden, retten wir den übergebenen Parameter auf den Stack.

Anschließend wird mit CHKKOM und GETBYT der nächste Parameter gelesen, die Startzeile. Die Zeilennummer befindet sich somit im X-Register. Nun wird die Spaltennummer vom Stack gezogen und ins Y-Register gebracht, das Carry-Bit gelöscht und die PLOT-Routine aufgerufen. Der Cursor befindet sich danach an der Ausgangsposition für die folgenden Invertierung/Normalisierung.

Diese Invertierung/Normalisierung wird jedoch ein wenig problematisch aufgrund der Tatsache, daß wir die Adresse der Startposition nicht kennen. In früheren ähnlichen Routinen begannen wir mit der Behandlung des Bildschirms prinzipiell ab $0400, der ersten Bildschirmzelle. Diesmal schreiben wir jedoch eine flexible Routine und müssen die Adresse der Ausgangsposition zuerst ermitteln.

Eine Möglichkeit besteht darin, in einer Schleife zur Adresse $0400 mehrmals - je nach Startzeile - 40 zu addieren und zum Schluß die Ausgangsspalte zu addieren. Im Folgenden verwende ich eine elegantere Lösung.

Die Zeropage enthält Informationen über Spalte und Zeile des Cursors und einen Zeiger auf die Zeile, in der er sich befindet. Diese Informationen werden nach jedem Aufruf von PLOT oder BSOUT - das heißt nach jeder Änderung der Cursorposition - aktualisiert und geben somit immer Auskunft über die momentane Position.

```
$D3: Cursorspalte (0-39)      $D6: Cursorzeile (0-24)
```

$D1/$D2: Zeiger auf die Adresse des ersten Zeichens der aktuellen Bildschirmzeile ($D1=Low-Byte/$D2=High-Byte). Interessant sind für uns $D3, die Cursorspalte, und $D1/$D2, der Zeiger auf das erste Zeichen jener Zeile, in der sich der Cursor momentan befindet. Wenn wir zu diesem Zeiger die Spaltennummer addieren, erhalten wir einen Zeiger, der exakt auf die aktuelle Cursorposition weist.

```
Zeiger auf Cursorposition einrichten
      A C014 18          CLC         ;ADDITION VORBEREITEN
      A C015 A5 D1       LDA $D1     ;SPALTE UND LOW-BYTE DES
      A C017 65 D3       ADC $D3     ;ZEILENZEIGERS ADDIEREN
      A C019 85 D1       STA $D1     ;UND SPEICHERN
      A C01B 90 02       BCC $C01F   ;SPRUNG, WENN
                                     ;KEIN ÜBERTRAG
      A C01D E6 D2       INC $D2     ;SONST ZEIGER-HIGH-BYTE
                                     ;INKREMENTIEREN
```

Dieser Programmteil nimmt die geschilderte Addition vor. Zum High-Byte des Zeigers ($D1) wird die Cursorspalte addiert. Wichtig ist die anschließende Berücksichtigung eines eventuellen Übertrags in das High-Byte des Zeigers (wenn das Ergebnis 255 überschreitet).

Getestet wird das Ergebnis mit BCC. Wenn das Carry-Bit immer noch gelöscht ist, wird der folgende Befehl übersprungen. Ist das Carry-Bit jedoch gesetzt, fand ein Übertrag statt und das High-Byte $D2 muß inkrementiert werden, um ein korrektes Ergebnis zu erhalten. Sollte dieses Vorgehensweise unklar sein, bitte ich Sie, die Kapitel über die Schiebe- und Rotierbefehle noch einmal zu studieren.

Wir besitzen nun in $D1/$D2 einen Zeiger auf die Ausgangsposition der Invertierung/Dekrementierung und können diese mit einer sehr gewöhnlichen Schleife vornehmen. Zuvor müssen wir jedoch den letzten Parameter aus dem BASIC-Text lesen, die Anzahl der Zeichen, die ab der Ausgangsposition zu behandeln sind.

```
Zeichenanzahl holen
      A C01F 20 FD AE    JSR $AEFD   ;CHKKOM AUFRUFEN
      A C022 20 9E B7    JSR $B79E   ;GETBYT AUFRUFEN
      A C025 8A          TXA         ;ZEICHENANZAHL ÜBER AKKU
      A C026 A8          TAY         ;NACH Y BRINGEN
```

Wie üblich wird zuerst das Komma und anschließend mit GETBYT der Parameter selbst gelesen, der sich nun im X-Register befindet.

Da wir jedoch mit einem Zeiger (indirekt-indizierte Adressierung) und Y als Indexregister in der folgenden Schleife arbeiten werden, übertragen wir die Zeichenanzahl - über den "Akkumulator-Umweg" - ins Y-Register.

```
Invertieren/Normalisieren
      A C027 B1 D1          LDA ($D1),Y  ;ZEICHEN LESEN
      A C029 49 80          EOR #$80     ;'UMDREHEN'
      A C02B 91 D1          STA ($D1),Y  ;UND ZURÜCKSCHREIBEN
      A C02D 88             DEY          ;ZÄHLER DEKREMENTIEREN
      A C02E 10 F7          BPL $C027    ;SPRUNG, WENN
                                         ;NOCH NICHT FERTIG
      A C030 60             RTS          ;ZURÜCK NACH BASIC
```

Dieser Programmteil bietet nichts Neues. In einer Schleife wird ab der Startadresse mit indirekt-indizierter Adressierung auf die einzelnen Zeichen zugegriffen, nach dem "eoren" mit $80 wird das "umgedrehte" Zeichen wieder zurückgeschrieben.

Das Programm besitzt zwei "Schönheitsfehler":

- Ein Zeichen mehr als angegeben wird "umgedreht". Beispiel: als Zeichenanzahl wurde zwei angegeben. Die Schleife bearbeitet jedoch der Reihe nach die Zeichen ($D1),2; ($D1),1 und ($D1),0. Entweder geben Sie beim Aufruf die Zeichenanzahl minus eins an (SYS 49152,5,10,9 wenn zehn Zeichen behandelt werden sollen), oder aber Sie fügen vor der Schleife einen DEY-Befehl ein.
- Da die Schleife auch das Zeichen an der aktuellen Cursorposition behandeln soll (Schleife inclusive null!), verwende ich BPL, damit die Schleife auch mit dem Zählerwert $00 ein letztes Mal durchlaufen wird.
  Daher ist die maximale Zeichenanzahl auf 129 beschränkt. Enthält Y beim Eintritt in die Schleife einen größeren Wert als $80 (=dezimal 128), erkennt BPL nach der Dekrementierung mit DEY einen "negativen" Wert (größer als $79) und verzweigt nicht.
  Wenn Sie diese Einschränkung stört, verwenden Sie BNE mit einem zusätzlichen Vergleichsbefehl (CPY #$FF und

BNE $C027). Nach einer entsprechenden Änderung können Sie mit dem Programm bis zu 256 Zeichen behandeln.

Das komplette Programm-Listing:

```
Spalte und Zeile lesen, Cursor auf Ausgangsposition setzen
      A C000 20 FD AE          JSR $AEFD     ;KOMMA LESEN
      A C003 20 9E B7          JSR $B79E     ;SPALTE NACH X HOLEN
      A C006 8A                TXA           ;SPALTE NACH AKKU BRINGEN
      A C007 48                PHA           ;UND AUF STACK RETTEN
      A C008 20 FD AE          JSR $AEFD     ;KOMMA LESEN
      A C00B 20 9E B7          JSR $B79E     ;ZEILE NACH X HOLEN
      A C00E 68                PLA           ;SPALTE VOM STACK ZIEHEN
      A C00F A8                TAY           ;UND NACH Y BRINGEN
      A C010 18                CLC           ;'PLOT' VORBEREITEN
      A C011 20 F0 FF          JSR $FFF0     ;CURSOR SETZEN
Zeiger auf Cursorposition einrichten
      A C014 18                CLC           ;ADDITION VORBEREITEN
      A C015 A5 D1             LDA $D1       ;SPALTE UND LOW-BYTE DES
      A C017 65 D3             ADC $D3       ;ZEILENZEIGERS ADDIEREN
      A C019 85 D1             STA $D1       ;UND SPEICHERN
      A C01B 90 02             BCC $C01F     ;SPRUNG, WENN
                                             ;KEIN ÜBERTRAG
      A C01D E6 D2             INC $D2       ;SONST ZEIGER-HIGH-BYTE
                                             INKREMENTIEREN
Zeichenanzahl holen
      A C01F 20 FD AE          JSR $AEFD     ;CHKKOM AUFRUFEN
      A C022 20 9E B7          JSR $B79E     ;GETBYT AUFRUFEN
      A C025 8A                TXA           ;ZEICHENANZAHL ÜBER AKKU
      A C026 A8                TAY           ;NACH Y BRINGEN
Invertieren/Normalisieren
      A C027 B1 D1             LDA ($D1),Y   ;ZEICHEN LESEN
      A C029 49 80             EOR #$80      ;'UMDREHEN'
      A C02B 91 D1             STA ($D1),Y   ;UND ZURÜCKSCHREIBEN
      A C02D 88                DEY           ;ZÄHLER DEKREMENTIEREN
      A C02E 10 F7             BPL $C027     ;SPRUNG, WENN NOCH
                                             ;NICHT FERTIG
      A C030 60                RTS           ;ZURÜCK NACH BASIC
```

Aufrufbeispiel: SYS 49152,5,10,100 invertiert 100 Zeichen ab Spalte fünf von Zeile 10.

FRMNUM, ADRFOR und ADRBYT

Angenommen, wir schreiben ein Programm, dem Parameter zu übergeben sind, die größer sind als 255. Mit GETBYT können wir nur Werte zwischen null und 255 lesen.

Eine Möglichkeit besteht darin, daß unser BASIC-Programm den Zwei-Byte-Wert in ein High- und ein Low-Byte zerlegt und diese einzeln mit GETBYT gelesen werden.

```
100 X=1000
110 XH=1000/256
120 XL=XH-256*XH
130 SYS 49152,5,10,XL,XH
```

Wie dieses Beispiel zeigt, verarbeitet GETBYT nicht nur Konstanten, sondern auch die Übergabe von Variablen. GETBYT holt in einem solchen Fall den zugehörigen Wert aus der Variablentabelle und übergibt ihn wie üblich im X-Register.

Diese Methode ist jedoch viel zu umständlich. Wozu haben wir den BASIC-Interpreter mit all seinen komfortablen Übergabe-Routinen?

FRMNUM ($AD8A)

FRMNUM wertet einen beliebigen numerischen Ausdruck aus und übergibt das Ergebnis im Fließkomma-Format in einem speziellen Speicherbereich der Zeropage.

Das wir uns mit diesem Format jedoch nicht auskennen, wird FRMNUM für uns erst zusammen mit einer weiteren Interpreter-Routine interessant.

ADRFOR ($B7F7)

Die Routine ADRFOR wandelt eine Fließkommazahl, die sich in dem reservierten Zeropage-Bereich befindet, in das Adreßformat, also in einen Zwei-Byte-Wert mit einem Low- und einem High-Byte.

ADRFOR übergibt das Low-Byte im Y-Register und das High-Byte im Akkumulator. Zusätzlich wird das Ergebnis noch im Adreßformat (Low-Byte/High-Byte) in der Zeropage abgelegt ($14/$15).

Ein Beispiel für den Einsatz beider Routinen:

```
SYS 49152,1000   oder

SYS 49152,10*X+33
A C000 JSR $AEFD ;CHKKOM AUFRUFEN
A C003 JSR $AD8A ;FRMNUM AUFRUFEN
A C006 JSR $B7F7 ;ADRFOR AUFRUFEN
```

In den Beispielen wird einmal eine Konstante übergeben und im zweiten Aufruf ein komplexer numerischer Ausdruck. In beiden Fällen wertet FRMNUM den Ausdruck aus und ADRFOR wandelt die resultierende Fließkommazahl in einen Zwei-Byte-Wert, wobei sich anschließend im Y-Register das Low-Byte und im Akkumulator das High-Byte befindet.

Der erste Aufruf übergibt die Konstante 1000 (=$03E8). Im Y-Register befindet sich nach dem Aufruf der beiden Routinen der Wert $03 und im Akkumulator der Wert $E8. Wie erläutert, wird das Ergebnis zusätzlich in der Zeropage abgelegt.

ADRBYT ($B7EB)

ADRBYT stellt eine Kombination der Routinen FRMNUM, ADRFOR, CHKKOM und GETBYT dar. Die Routine ADRBYT ruft zuerst nacheinander FRMNUM und ADRFOR auf und liest damit einen Zwei-Byte-Wert, der in den beschriebenen Zeropage-Adressen $14/$15 abgelegt wird.

Anschließend werden CHKKOM und GETBYT aufgerufen und somit ein folgender Ein-Byte-Wert gelesen und im X-Register übergeben.

Achtung: den zuerst gelesenen Zwei-Byte-Wert finden Sie anschließend zwar noch in der Zeropage wieder, jedoch nicht mehr in den Registern, da die folgenden Aufrufe von CHKKOM und GETBYT sowohl das Y-Register als auch den Akkumulator beeinflussen.

Ein Beispiel:

```
SYS 49152,1000,100
A C000 JSR $AEFD ;CHKKOM AUFRUFEN
A C003 JSR $B7EB ;ADRBYT AUFRUFEN

Ergebnis: - $14/$15: 1000 ($03E8) im Format Low-Byte/High-Byte
          - X-Register: 100 ($64)
```

Invertieren/Normalisieren, Version 2

Mit diesen Kenntnissen können wir unsere Invertier-/Normalisier-Routine noch flexibler gestalten und eine beliebige Zeichenanzahl übergeben. Wollen wir jedoch die Routine ADRBYT einsetzen, müssen wir den Aufruf "umdrehen", da ADRBYT zuerst einen Zwei-Byte-Wert und anschließend einen Ein-Byte-Wert liest.

```
SYS (STARTADRESSE),(ZEICHENANZAHL),(SPALTE),(ZEILE)
```

Beispiel:

```
SYS 49152,500,5,10

Parameterübergabe
      A C000 20 FD AE          JSR $AEFD    ;CHKKOM AUFRUFEN
      A C003 20 EB B7          JSR $B7EB    ;ADRBYT AUFRUFEN
      A C006 8A                TXA          ;SPALTE AUF
      A C007 48                PHA          ;STACK BRINGEN
```

```
A C008 20 FD AE     JSR $AEFD    ;CHKKOM AUFRUFEN
A C00B 20 9E B7     JSR $B79E    ;GETBYT (=>ZEILE IN X)
A C00E 68           PLA          ;SPALTE VOM STACK
A C00F A8           TAY          ;NACH Y BRINGEN
A C010 18           CLC          ;'PLOT' VOEBREITEN
A C011 20 F0 FF     JSR $FFF0    ;CURSOR AUF
                                 ;AUSGANGSPOSITION
```

Nach dem Aufruf von CHKKOM und ADRBYT befindet sich in $14/$15 die Zeichenanzahl in der Form Low-Byte/High-Byte und im X-Register die Ausgangsspalte. Die Spalte wird auf den Stack gebracht und anschließend mit CHKKOM und GETBYT die Ausgangszeile gelesen. Die Spalte wird vom Stack geholt und ins Y-Register kopiert, bevor der Cursor mit der PLOT-Routine auf die Ausgangsposition gesetzt wird.

```
Zeiger auf Ausgangsposition erzeugen
      A C014 18          CLC          ;ADDITION VORBEREITEN
      A C015 A5 D1       LDA $D1      ;ZUM ZEIGER AUF DEN
                                      ;ZEILENANFANG
      A C017 65 D3       ADC $D3      ;DIE CURSORSPALTE
      A C019 85 D1       STA $D1      ;ADDIEREN UND EINEN
      A C01B 90 02       BCC $C01F    ;EVENTUELLEN ÜBERTRAG
      A C01D E6 D2       INC $D2      ;BERÜCKSICHTIGEN
```

Der Zeiger auf diese Ausgangsposition wird analog zum vorigen Programm erzeugt, indem die aktuelle Cursorspalte (nach PLOT!) zum Zeiger auf das erste Zeichen der Cursorzeile addiert wird (mit Berücksichtigung eines eventuellen Übertrags!).

```
Invertier-/Normalisier-Schleife
      A C01F A6 15       LDX $15      ;X=BLOCKZÄHLER (HIGH-BYTE)
      A C021 F0 14       BEQ $C037    ;NULL 256-BYTE-BLÖCKE?
      A C023 A0 FF       LDY #$FF     ;SCHLEIFENZÄHLER INITIAL.
      A C025 B1 D1       LDA ($D1),Y  ;ZEICHEN LESEN
      A C027 48 80       EOR #$80     ;UMDREHEN
      A C029 91 D1       STA ($D1),Y  ;UND ZURÜCKSCHREIBEN
      A C02B 88          DEY          ;SCHLEIFENZÄHLER
                                      ;DEKREMENTIEREN
```

```
A C02C C0 FF        CPY #$FF     ;UND MIT $FF VERGLEICHEN
A C02E D0 F5        BNE $C025    ;SPRUNG, WENN UNGLEICH
A C030 E6 D2        INC $D2      ;SONST ZEIGER-HIGH
                                 ;INKREMENTIEREN
A C032 CA           DEX          ;BLOCKZÄHLER
                                 ;DEKREMENTIEREN
A C033 30 06        BMI $C03B    ;UND FERTIG,
                                 ;WENN X=$FF (NEGATIV)
A C035 D0 EC        BNE $C023    ;NOCH EIN
                                 ;KOMPLETTER BLOCK?
A C037 A4 14        LDY $14      ;NEIN, DANN Y MIT
                                 ;REST LADEN
A C039 D0 F0        BNE $C02B    ;UND IN SCHLEIFE SPRINGEN
A C03B 60           RTS          ;ZURÜCK NACH BASIC
```

Diese Schleife ähnelt sehr stark jener, die wir zur Ausgabe der Fehlermeldungen verwendeten. Auch hier sind wiederum (eventuell) mehrere komplette 256-Byte-Blöcke und ein Rest zu behandeln.

Das X-Register fungiert als Blockzähler und wird mit dem High-Byte ($15) der Zeichenanzahl geladen. Im Unterschied zur Ausgabe der Fehlermeldungen wird nun jedoch geprüft, ob dieses High-Wert den Wert null besitzt. Wenn ja, soll kein einziger kompletter Block behandelt werden (Zeichenanzahl kleiner als 255). In diesem Fall wird sofort nach Adresse $C037 verzweigt, wo Y mit dem Rest geladen wird.

Wie üblich werden komplette Blöcke behandelt, indem Y mit $FF initialisiert und bis $00 heruntergezählt wird. Nach jedem Block wird das High-Byte des Zeigers $D1/$D2 inkrementiert und der Blockzähler X dekrementiert, bis X den Wert $00 enthält und somit alle kompletten Blöcke behandelt sind.

Y wird nun mit dem Rest (dem Low-Byte der Zeichenanzahl geladen) und dann folgt ein kleiner "Trick". Um den "Schönheitsfehler" aus Version 1 zu vermeiden (beim Aufruf muß die Zeichenanzahl minus eins angegeben werden), wird nicht zum

Beginn der Schleife verzweigt ($C025), sondern zum Schleifenende, wo Y dekrementiert wird ($C02B).

Der Zähler wird dadurch um eins korrigiert, bevor die restliche Zeichenanzahl behandelt wird. Nach dem Verlassen der inneren Schleife wird der Blockzähler wieder dekrementiert und es erfolgt ein Unterlauf (DEX => X=$FF). Die Bedingung BMI ("verzweige, wenn negativ") ist erfüllt und BMI $C03B verzeigt zum Programmende, wo die Rückkehr in das aufrufende BASIC-Programm erfolgt.

Mit diesem Programm besitzen Sie eine flexible Invertier-/Normalisier-Routine, die Sie wie im folgenden Beispiel zusammmen mit beliebigen BASIC-Programmen einsetzen können.

```
100 PRINT CHR$(147):REM BILDSCHIRM LOESCHEN
110 PRINT "BLINKENDER BILDSCHIRMAUSSCHNITT"
120 INPUT "STARTZEILE";Z
130 INPUT "STARTSPALTE";S
140 INPUT "ZEICHENANZAHL";A
150 SYS 49152,A,S,Z:REM AUFRUF DER ROUTINE
160 GET A$:IF A$="" THEN 160:REM AUF TASTE WARTEN
170 GOTO 150
```

Nach dem Starten fragt Sie dieses BASIC-Programm nach den benötigten Parametern und ruft anschließend die Assembler-Routine auf. Die Bildschirmdarstellung im angegebenen Bereich wird "umgedreht".

Das Programm läuft in einer Endlosschleife. Immer, wenn Sie eine Taste betätigen, wird die Assembler-Routine erneut aufgerufen und der aktuelle Zeichenzustand "umgekehrt" (Endlosschleife).

Das komplette Programm-Listing:

```
Parameterübergabe
      A C000 20 FD AE          JSR $AEFD     ;CHKKOM AUFRUFEN
      A C003 20 EB B7          JSR $B7EB     ;ADRBYT AUFRUFEN
      A C006 8A                TXA           ;SPALTE AUF
```

```
A C007 48          PHA          ;STACK BRINGEN
A C008 20 FD AE    JSR $AEFD    ;CHKKOM AUFRUFEN
A C00B 20 9E B7    JSR $B79E    ;GETBYT (=>ZEILE IN X)
A C00E 68          PLA          ;SPALTE VOM STACK
A C00F A8          TAY          ;NACH Y BRINGEN
A C010 18          CLC          ;'PLOT' VOEBREITEN
A C011 20 F0 FF    JSR $FFF0    ;CURSOR AUF
                                ;AUSGANGSPOSITION
Zeiger auf Ausgangsposition erzeugen
A C014 18          CLC          ;ADDITION VORBEREITEN
A C015 A5 D1       LDA $D1      ;ZUM ZEIGER AUF DEN
                                ;ZEILENANFANG
A C017 65 D3       ADC $D3      ;DIE CURSORSPALTE
A C019 85 D1       STA $D1      ;ADDIEREN UND EINEN
A C01B 90 02       BCC $C01F    ;EVENTUELLEN ÜBERTRAG
A C01D E6 D2       INC $D2      ;BERÜCKSICHTIGEN
Invertier-/Normalisier-Schleife
A C01F A6 15       LDX $15      ;X=BLOCKZÄHLER (HIGH-BYTE)
A C021 F0 14       BEQ $C037    ;NULL 256-BYTE-BLÖCKE?
A C023 A0 FF       LDY #$FF     ;SCHLEIFENZÄHLER INITIAL.
A C025 B1 D1       LDA ($D1),Y  ;ZEICHEN LESEN
A C027 48 80       EOR #$80     ;UMDREHEN
A C029 91 D1       STA ($D1),Y  ;UND ZURÜCKSCHREIBEN
A C02B 88          DEY          ;SCHLEIFENZÄHLER
                                ;DEKREMENTIEREN
A C02C C0 FF       CPY #$FF     ;UND MIT $FF VERGLEICHEN
A C02E D0 F5       BNE $C025    ;SPRUNG, WENN UNGLEICH
A C030 E6 D2       INC $D2      ;SONST ZEIGER-HIGH
                                ;INKREMENTIEREN
A C032 CA          DEX          ;BLOCKZÄHLER
                                ;DEKREMENTIEREN
A C033 30 06       BMI $C03B    ;UND FERTIG,
                                ;WENN X=$FF (NEGATIV)
A C035 D0 EC       BNE $C023    ;NOCH EIN
                                ;KOMPLETTER BLOCK?
A C037 A4 14       LDY $14      ;NEIN, DANN Y MIT
                                ;REST LADEN
A C039 D0 F0       BNE $C02B    ;UND IN SCHLEIFE SPRINGEN
A C03B 60          RTS          ;ZURÜCK NACH BASIC
```

GETPOS ($B08B)

GETPOS ist eine der flexibelsten Interpreter-Routinen. Diese Routine holt einen Zeiger auf eine beliebige Variable, die im BASIC-Text angegeben wird. Der Zeiger wird im Akku (Low-Byte) und Y-Register (High-Byte) übergeben.

Geben Sie bitte zuerst das abgebildete kleine Assembler-Programm ein, verlassen Sie den Monitor mit X, löschen Sie den BASIC-Speicher mit NEW, geben Sie das BASIC-Programm ein und starten Sie es.

```
A C000 20 FD AE          JSR $AEFD    ;CHKKOM AUFRUFEN
A C003 20 8B B0          JSR $B08B    ;GETPOS AUFRUFEN
A C006 00                BRK

100 A%=513
110 SYS 49152,A%
```

Das Assembler-Programm liest zuerst mit CHKKOM das der Adresse folgende Komma und ruft danach GETPOS auf. Die Registeranzeige des Monitors gibt anschließend $1F als Inhalt des Akkumulators und $08 als Inhalt des Y-Registers aus.

GETPOS übergibt uns somit einen Zeiger auf die Adresse $081F. Geben Sie bitte ein: M 081F.

```
081F  02 01 .....
```

Uns interessieren nur die ersten beiden Bytes ab dieser Adresse. Sie enthalten die Werte $02 und $01, also die Zahl $0201 (=dezimal 513) im "umgekehrten Adreßformat".

GETPOS lieferte uns einen Zeiger auf die Variable <A%>, der unmittelbar hinter die beiden "Namensbyte" weist, also auf das erste Byte der Integerzahl.

Entsprechendes gilt für andere Variablentypen. Unabhängig vom Variablentyp liefert GETPOS immer einen Zeiger, der entweder - bei numerischen Variablen - auf das erste Byte der Zahl, oder

aber - bei Stringvariablen - auf das erste Byte der "Stringdescriptoren", auf das Längenbyte, weist.

Diese Arbeitsweise von GETPOS gilt gleichermaßen für einfache und für Arrayvariablen.

Dank dem übergebenen Zeiger können wir nicht nur lesend, sondern auch schreibend auf die von BASIC übergebenen Variablen zugreifen. Angenommen, wie erstellen eine Assembler-Routine, die eine Berechnung ausführt. Das Ergebnis ist ein Zwei-Byte-Wert, der an das BASIC-Programm zurückübergeben werden soll.

Eine "primitive" Möglichkeit zur Rückübergabe besteht darin, den Wert in zwei Speicherzellen zu schreiben, die das BASIC-Programm anschließend mit PEEK ausliest.

```
Zahlenübergabe an BASIC, Version 1
Assembler-Programm
      ...
      ...
      STA $FB ;LOW-BYTE ÜBERGEBEN
      STX $FC ;HIGH-BYTE ÜBERGEBEN
      RTS
BASIC-Programm
      ...
      ...
      SYS 49152
      PRINT PEEK(251)+256*PEEK(252)
      ...
      ...
```

Dank GETPOS wählen wir jedoch den komfortableren Weg und übergeben das Ergebnis direkt in einer Integervariablen.

```
Zahlenübergabe an BASIC, Version 2
Zeiger auf Integervariable holen
      A C000 20 FD AE          JSR $AEFD    ;CHKKOM AUFRUFEN
      A C003 20 8B B0          JSR $B08B    ;GETPOS => ZEIGER
                                            ;AUF VARIABLE
```

```
      A C006 85 FB              STA $FB      ;ZEIGER LOW NACH $FB
      A C008 84 FC              STY $FC      ;ZEIGER HIGH NACH $FC
Integerwert $0406 (=dezimal 1030) übergeben
      A C00A A0 00              LDY #$00     ;Y INITIALISIEREN
      A C00C A9 04              LDA #$04     ;ERGEBNIS: HIGH-BYTE LADEN
      A C00E 91 FB              STA ($FB),Y  ;UND NACH ($FB),0
      A C010 C8                 INY          ;Y INKREMENTIEREN
      A C011 A9 06              LDA #$06     ;ERGEBNIS: LOW-BYTE LADEN
      A C013 91 FB              STA ($FB),Y  ;UND NACH ($FB),1
      A C015 60                 RTS

BASIC-Programm
      100 SYS 49152,X%
      110 PRINT X%
```

Wenn Sie - nach der Eingabe des Assembler-Programms - das BASIC-Programm starten, erhalten Sie als Inhalt der Variablen <X%> den Wert 1030, der dieser BASIC-Variablen vom Assembler-Programm zugewiesen wurde.

Der Ablauf: Nach CHKKOM wird mit GETPOS ein Zeiger auf die angegebene Integervariable geholt, der sich anschließend im Akkumulator (Low-Byte) und im Y-Register (High-Byte) befindet.

Da über diesen Zeiger im weiteren Programmverlauf mit indirekt-indizierter Adressierung auf die betreffende Variable zugegriffen wird, muß er in der Zeropage abgelegt werden. Wie üblich wird der Zeiger in $FB/$FC gespeichert.

Der Zeiger in $FB/$FC weist auf das erste Byte der Integervariablen nach den "Namensbytes", also auf das High-Byte der Zahl.

In dieses High-Byte wird der Wert $04 geschrieben, Y wird inkrementiert und danach der Wert $06 in das Low-Byte der Integerzahl geschrieben. Die Variable <X%> besitzt nun den Wert $04 $06 (High-Byte/Low-Byte), also 1030 ($0406), jenen Wert, den der Befehl PRINT X% ausgibt.

GETPOS besitzt übrigens einen unschätzbaren Vorteil: wenn die angegebene Variable noch nicht existiert, wird Sie durch GETPOS angelegt (und erhält den Ausgangswert null). Daher ist es nicht nötig, daß das BASIC-Programm die übergebene Variable vor dem Aufruf der Assembler-Routine selbst anlegt (zum Beispiel mit X%=0).

Mit GETBYT, FRMNUM und ADRFOR konnten wir Ein- oder Zwei-Byte-Werte aus einem BASIC-Text lesen. Mit GETPOS können wir jedoch direkt auf BASIC-Variablen zugreifen, lesen oder vor allem auch schreibend und somit von Assembler aus BASIC-Variablen anlegen, wie wir soeben sahen.

Auch der Lesezugriff auf eine Variable kann sehr nützlich sein, wie wir anhand von Stringvariablen sehen werden. Der Zugriff auf Stringvariablen ist jedoch ein wenig komplex, da GETPOS nicht direkt einen Zeiger auf den String selbst, sondern einen Zeiger auf die Stringdescriptoren in der Variablentabelle liefert.

Um einen String zu lesen, müssen wir wie folgt vorgehen:

1. Mit GETPOS im Akkumulator und Y-Register einen Zeiger auf das erste Descriptorenbyte holen, die Stringlänge.
2. Diesen Zeiger in der Zeropage ablegen, um anschließend mit indirekt-indizierter Adressierung auf die Stringdescriptoren zuzugreifen.
3. Die drei Descriptorenbytes (Stringlänge, Zeiger auf String) lesen und ebenfalls in der Zeropage ablegen.
4. Über den Zeiger auf den String kann nun - wiederum mit indirekt-indizierter Adressierung - auf den String selbst zugegriffen werden.

Übungsprogramme zur Parameter-Übergabe

Im Folgenden stelle ich mehrere Übungsprogramme vor. Da die Materie doch etwas komplex ist, werden die betreffenden Programme "abschnittsweise" erläutert und am Ende der Kapitel das komplette Programm-Listing abgebildet.

BASIC-String ausgeben

Das vorgestellte Schema zum Zugriff auf Stringvariablen will ich an einem Demoprogramm veranschaulichen, das in der Lage ist, auf einen beliebigen BASIC-String zuzugreifen und diesen auf dem Bildschirm auszugeben. Der Aufruf der Assembler-Routine kann zum Beispiel so aussehen:

```
100 A$="DIES IST EIN TEST"
110 SYS 49152,A$
```

Für die Ablage des Zeigers und der Stringdescriptoren in der Zeropage benötigen wir fünf Byte. Der beim C64 reservierte Zeropage-Bereich $FB-$FE ist daher um genau ein Byte zu klein.

Wir verwenden für die Stringdescriptoren den Bereich der "Fließkomma-Akkumulatoren", der beim C64 bei $61 beginnt und bei $70 endet. Dieser Bereich ist vorm Überschreiben geschützt, bis das Assembler-Programm mit RTS endet (=> Rückkehr nach BASIC) und der BASIC-Interpreter wieder "regiert", der diesen Bereich für eigene Zwecke benötigt.

Der Zeiger auf die übergebene Variable - im BASIC-Demoprogramm die Stringvariable <A$> - wird wie in den vorigen Programmen mit CHKKOM und GETPOS in die Register geholt und anschließend in der Zeropage in $FB/$FC abgelegt.

```
Zeiger auf Variable erzeugen
      A C000 20 FD AE           JSR $AEFD     ;CHKKOM
      A C003 20 8B B0           JSR $B08B     ;GETPOS => ZEIGER
                                              ;AUF DESCRIPTOREN
      A C006 85 FB              STA $FB       ;ZEIGER LOW NACH $FB
      A C008 84 FC              STY $FC       ;ZEIGER HIGH NACH $FC
```

Anschließend wird in einer Schleife auf die Stringdescriptoren zugegriffen, auf die der Zeiger $FB/$FC weist. Die drei Descriptorbytes (Stringlänge/Adresse Low/Adresse High) werden in $61 (Stringlänge) und $62/$63 (Zeiger auf String) abgelegt.

Achten Sie bitte darauf, daß die drei Descriptorbytes mit STA $0061,Y in der Zeropage abgelegt werden. Die "Y-indizierte-Zeropage-Adressierung" STA $61,Y existiert leider nicht (im Gegensatz zur "X-indizierten-Zeropage-Adressierung" STA $61,X).

```
Stringdescriptoren holen
      A C00A A0 02            LDY #$02      ;SCHLEIFENZÄHLER
                                            ;INITIALISIEREN
      A C00C B1 FB            LDA ($FB),Y   ;STRINGDESCRIPTOR HOLEN
      A C00E 99 61 00         STA $0061,Y   ;UND NACH $0061,Y BRINGEN
      A C011 88               DEY           ;ZÄHLER DEKREMENTIEREN
      A C012 10 F8            BPL $C00C     ;FERTIG, WENN DREI
                                            ;BYTE BEARBEITET
```

Nun folgt der letzte Programmteil, der Zugriff auf den String und die Ausgabe der einzelnen Zeichen mit BSOUT. Der Zugriff erfolgt über den Zeiger in $62/$63 in einer aufwärts zählenden Schleife. Die Zählrichtung der Schleife ist wichtig, da eine abwärts zählende Schleife zuerst das letzte, dann das vorletzte Zeichen ausgeben würde, also den String in "Spielgelschrift" ausgibt. Die Schleife ist beendet, wenn der Zähler Y mit der Stringlänge (in $61) identisch ist.

```
Stringausgabe
      A C014 A0 00            LDY #$00      ;ZÄHLER INTIALISIEREN
      A C016 B1 62            LDA ($62),Y   ;STRINGZEICHEN LESEN
      A C018 20 D2 FF         JSR $FFD2     ;UND MIT 'BSOUT' AUSGEBEN
      A C01B C8               INY           ;ZÄHLER INKREMENTIEREN
      A C01C C4 61            CPY $61       ;UND MIT STRINGLÄNGE
                                            ;VERGLEICHEN
      A C01E D0 F6            BNE $C016     ;SPRUNG, WENN UNGLEICH
      A C020 60               RTS           ;SONST STRING KOMPLETT
                                            ;AUSGEGEBEN
```

Das vollständige Programm-Listing

```
Zeiger auf Variable erzeugen
      A C000 20 FD AE         JSR $AEFD     ;CHKKOM
```

```
A C003 20 8B B0          JSR $B08B    ;GETPOS => ZEIGER AUF
                                      ;DESCRIPTOREN
A C006 85 FB             STA $FB      ;ZEIGER LOW NACH $FB
A C008 84 FC             STY $FC      ;ZEIGER HIGH NACH $FC
;Stringdescriptoren holen
A C00A A0 02             LDY #$02     ;SCHLEIFENZÄHLER
                                      ;INITIALISIEREN
A C00C B1 FB             LDA ($FB),Y  ;STRINGDESCRIPTOR HOLEN
A C00E 99 61 00          STA $0061,Y  ;UND NACH $0061,Y BRINGEN
A C011 88                DEY          ;ZÄHLER DEKREMENTIEREN
A C012 10 F8             BPL $C00C    ;FERTIG, WENN DREI
                                      ;BYTE BEARBEITET
;Stringausgabe
A C014 A0 00             LDY #$00     ;ZÄHLER INTIALISIEREN
A C016 B1 62             LDA ($62),Y  ;STRINGZEICHEN LESEN
A C018 20 D2 FF          JSR $FFD2    ;UND MIT 'BSOUT' AUSGEBEN
A C01B C8                INY          ;ZÄHLER INKREMENTIEREN
A C01C C4 61             CPY $61      ;UND MIT STRINGLÄNGE
                                      ;VERGLEICHEN
A C01E D0 F6             BNE $C016    ;SPRUNG, WENN UNGLEICH
A C020 60                RTS          ;SONST STRING KOMPLETT
                                      ;AUSGEGEBEN
```

Stringarray ausgeben

Das im letzten Kapitel vorgestellte Programm kann ohne größeren Aufwand zu einer in der Praxis sehr nützlichen Routine ausgebaut werden.

Oftmals soll ein BASIC-Programm komplette Teile eines Stringarrays auf dem Bildschirm ausgeben. Ein Beispiel: sie schreiben in BASIC eine Adreßverwaltung. Die einzelnen Adressen befinden sich in dem Stringarray <A$(...)>. Auf dem Bildschirm befindet sich immer ein Ausschnitt dieser "Liste", zum Beispiel jeweils 20 Strings (= Adressen).

Wenn das Programm wirklich komfortabel ist, sollte es dem Benutzer möglich sein, mit einem Tastendruck in der Adreßliste nach vor- beziehungsweise rückwärts zu blättern.

Konkret: momentan befinden sich die Adressen 30 bis 49 auf dem Bildschirm. Mit einem Druck auf die Taste CURSOR RIGHT wird die Liste "vorwärts" geblättert und die Adressen 31 bis 50 ausgegeben.

Der Haken an diesem BASIC-Programm: angenommen, der Benutzer gibt "Dauerfeuer" mit der Taste CURSOR RIGHT, weil er die Liste um zum Beispiel 15 Adressen nach vorne blättern will. Das BASIC-Programm muß in diesem Fall 15mal den jeweiligen Listenausschnitt ausgeben, jeweils 20 Strings.

Dieser Vorgang ist mit einem BASIC-Programm quälend langsam. Die Liste wird geradezu im Zeitlupentempo "gerollt". Optimal wäre eine Assembler-Routine, die blitzartig den aktuellen Listenausschnitt auf dem Bildschirm ausgibt.

Zu einer solchen Routine können wir unsere String-Ausgabe problemlos erweitern. Anstelle eines werden beispielsweise 20 Strings ausgegeben. Der Aufruf soll lauten:

```
SYS (STARTADRESSE),(1.STRING),(STRINGANZAHL)
zum Beispiel: SYS 49152,A$(30),20
```

Mit GETPOS holen wir die Adresse der angegebenen Variablen <A$(30)> und geben in einer Schleife 20 die Strings <A$(30)> bis <A$(49)> aus.

Der erste Programmteil holt mit GETPOS den Zeiger auf die übergebene Stringvariable und anschließend mit GETBYT die Anzahl der auszugebenden Strings.

```
Parameter aus BASIC-Text holen
      A C000 20 FD AE           JSR $AEFD     ;CHKKOM
      A C003 20 8B B0           JSR $B08B     ;GETPOS => ZEIGER
                                              ;AUF VARIABLE
      A C006 85 FB              STA $FB       ;ZEIGER LOW NACH $FB
      A C008 84 FC              STY $FC       ;ZEIGER HIGH NACH $FC
      A C00A 20 FD AE           JSR $AEFD     ;CHKKOM
      A C00D 20 9E B7           JSR $B79E     ;GETBYT => X=STRINGANZAHL
```

Im folgenden Programmteil lesen wir über den Zeiger in $FB/$FC die drei Descriptorenbytes und speichern sie in $61-$63.

```
Stringdescriptoren holen
      A C010 A0 02          LDY #$02      ;DREI DURCHGÄNGE
      A C012 B1 FB          LDA ($FB),Y   ;DESCRIPTORBYTE LESEN
      A C014 99 61 00       STA $0061,Y   ;UND NACH $0061,Y
      A C017 88             DEY           ;ZÄHLER DEKREMENTIEREN
      A C018 10 F8          BPL $C012     ;SPRUNG, WENN NOCH
                                          ;NICHT FERTIG
```

Der dritte Programmabschnitt bietet ebenfalls nicht Neues. Der String wird in einer Schleife ausgegeben, beginnend mit dem ersten Zeichen. Die Schleife wird verlassen, wenn das letzte Zeichen ausgegeben wurde und der Zähler Y mit der Stringlänge in $61 identisch ist.

```
String ausgeben
      A C01A A0 00          LDY #$00      ;ZÄHLER INITIAL.
      A C01C B1 62          LDA ($62),Y   ;STRINGZEICHEN LESEN
      A C01E 20 D2 FF       JSR $FFD2     ;UND AUSGEBEN
      A C021 C8             INY           ;NÄCHSTES ZEICHEN
      A C022 C4 61          CPY $61       ;STRING KOMPLETT
                                          ;AUSGEGEBEN?
      A C024 D0 F6          BNE $C01C     ;SPRUNG, WENN NICHT FERTIG
```

Der letzte Programmabschnitt nach Verlassen der inneren Schleife beinhaltet die eigentliche Neuerung. Da in jeder Zeile ein String auszugeben ist, wird zuerst ein Zeilenvorschub bewirkt, indem der ASCII-Code der Taste RETURN mit BSOUT ausgegeben wird.

Und nun folgt der eigentlich interessante Teil. Wir wissen, daß in einem Stringarray die Stringdescriptoren jeweils drei Byte umfassen. Diesen drei Byte folgen die entsprechenden Descriptoren des nächsten Arraystrings.

Angenommen, die Descriptoren des Strings <A$(1)> beginnen ab Adresse $1000. Der folgende Abschnitt der Variablentabelle ist dann gemäß dem folgenden Schema aufgebaut:

```
$1000-$1002: Descriptoren von <A$(1)>
$1003-$1005: Descriptoren von <A$(2)>
$1006-$1008: Descriptoren von <A$(3)>
$1009-$100B: Descriptoren von <A$(4)>
...
...
```

Wenn der Aufruf aus dem BASIC-Programm lautet: SYS 49152,A$(1),20, befindet sich nach dem ersten Teil des Programms in $FB/$FC ein Zeiger auf das erste Descriptorbyte von <A$(1)>.

Nachdem dieser String ausgegeben wurde, kann der Zeiger auf die Descriptoren von <A$(2)> "verbogen" werden, indem der Wert $03 addiert wird.

Der letzte Programmteil addiert demnach $03 zu dem Zeiger in $FB/$FC (unter Berücksichtigung eines eventuell entstehenden Übertrags!). Anschließend wird der Stringzähler - das X-Register - dekrementiert.

Wurden noch nicht alle Strings ausgegeben, erfolgt ein Sprung zum Programmteil "Stringdescriptoren holen". Die drei Descriptorbytes des nächsten Strings (auf den der um $03 erhöhte Zeiger in $FB/$FC weist) werden geholt und dieser ebenfalls ausgegeben.

```
Nächster String
      A C026 A9 0D            LDA #$0D     ;ASCII-CODE VON 'RETURN'
      A C028 20 D2 FF         JSR $FFD2    ;AUSGEBEN
                                           ;(=>ZEILENVORSCHUB)
      A C02B 18               CLC          ;ADDITION VORBEREITEN
      A C02C A5 FB            LDA $FB      ;UND $FB/$FC UNTER
      A C02E 69 03            ADC #$03     ;BERÜCKSICHTIGUNG EINES
      A C030 85 FB            STA $FB      ;EVENTUELL ENTSTEHENDEN
      A C032 90 02            BCC $C036    ;ÜBERTRAGS UM DREI
```

```
A C034 E6 FC            INC $FC      ;ERHÖHEN.
A C036 CA               DEX          ;STRINGZÄHLER
                                     ;DEKREMENTIEREN
A C037 D0 D7            BNE $C010    ;=> NÄCHSTEN STRING
                                     ;AUSGEBEN
A C039 60               RTS          ;ZURÜCK NACH BASIC
```

Um Ihnen die Anwendung der Routine zu demonstrieren, erstellte ich ein kleines BASIC-Programm, das mit dieser Routine wie beschrieben ein Stringarray "durchblättert".

BASIC-Demoprogramm

```
100 DIM A$(100)
110 FOR I=1 TO 100
120 : A$(I)="DIES IST DER STRING NUMMER"+STR$(I)
130 NEXT
140 :
150 NR=1:REM START MIT <A$(1)>
160 PRINT CHR$(147);:REM BILDSCHIRM LOESCHEN
170 SYS 49152,A$(NR),24:REM ROUTINE AUFRUFEN
180 GET A$:IF A$="" THEN 180
190 IF A$=CHR$(29) THEN IF NR<81 THEN NR=NR+1
200 IF A$=CHR$(157) THEN IF NR>1 THEN NR=NR-1
210 GOTO 160:REM ENDLOSSCHLEIFE
```

Das BASIC-Programm besteht aus einem vorbereitenden Teil (Zeile 100-130), der ein Stringarray <A$(1)>-<A$(100)> mit den Strings

```
A$(1)="DIES IST DER STRING NUMMER 1"
A$(2)="DIES IST DER STRING NUMMER 2"
A$(3)="DIES IST DER STRING NUMMER 3"
A$(4)="DIES IST DER STRING NUMMER 4"
...
...
A$(100)="DIES IST DER STRING NUMMER 100"
```

erzeugt, und einem Hauptteil, der die Liste verwaltet. Der Hauptteil löscht vor jeder Ausgabe den Bildschirm und ruft anschließend die "Ausgabe-Routine" auf. Ausgegeben werden je-

weils 24 Strings ab String Nummer <NR>. Da <NR> mit dem Wert eins initialisiert wurde, gibt die Routine beim ersten Aufruf die Strings <A$(1)> bis <A$(20)> aus.

In Zeile 180 befindet sich eine Warteschleife, die erst verlassen wird, wenn Sie eine Taste betätigen.

Je nach betätigter Cursortaste (CURSOR RIGHT oder CURSOR LEFT) wird <NR> um eins erhöht oder vermindert und die Ausgabe-Routine erneut aufgerufen.

Sie können die Liste problemlos und vor allem schnell durchblättern, etwa doppelt bis dreimal so schnell wie mit den entsprechenden BASIC-Befehlen:

```
170 FOR I=NR TO NR+23:PRINT A$(I):NEXT
```

Die Routine kann um ein Vielfaches beschleunigt werden, wenn Sie die Zeichen direkt in den Bildschirmspeicher schreiben, anstatt die relativ langsame Ausgabe-Routine BSOUT zu verwenden.

Beachten Sie jedoch, daß Strings im ASCII-Format gespeichert werden, passend für die Ausgabe mit BSOUT. Wollen Sie die Zeichen direkt in das Video-RAM schreiben, müssen Sie die ASCII-Codes in den Bildschirmcode wandeln.

Sollen ausschließlich "normale" Zeichen ausgegeben werden und keine Graphikzeichen, ist die Umwandlung recht einfach. Bei Kleinbuchstaben ist im ASCII-Code Bit 6 gesetzt, im Bildschirmcode dagegen gelöscht. Bei Großbuchstaben ist liegt der Unterschied zwischen beiden Codes im gesetzten (ASCII-Code) beziehungsweise gelöschten (Bildschirmcode) Bit 7. Daraus ergibt sich folgende Wandlungsmethode:

```
            LDA (ZEICHEN)  ;ASCII-CODE DES ZEICHENS LADEN
            BMI (ADRESSE)  ;BIT 7 GESETZT => VERZWEIGEN
            AND #$BF       ;SONST BIT 6 LOESCHEN
(ADRESSE)   AND #$7F       ;IMMER BIT 7 LOESCHEN
```

Das Prinzip: Wenn im ASCII-Code des betreffenden Zeichens Bit 7 gesetzt ist (Großbuchstabe), verzweigt BMI zu ADRESSE und Bit 7 wird durch eine AND-Verknüpfung mit $7F gelöscht.

Ansonsten handelt es sich um einen Kleinbuchstaben mit gesetztem Bit 6. Dieses Bit wird durch AND #$BF gelöscht. Der folgende Befehl AND #$7F zum Löschen des siebten Bits hat keinerlei Auswirkungen, da Bit 7 bei Kleinbuchstaben ja sowieso gelöscht ist.

```
Das komplette Programmlisting
Parameter aus BASIC-Text holen
      A C000 20 FD AE          JSR $AEFD     ;CHKKOM
      A C003 20 8B B0          JSR $B08B     ;GETPOS => ZEIGER
                                             ;AUF VARIABLE
      A C006 85 FB             STA $FB       ;ZEIGER LOW NACH $FB
      A C008 84 FC             STY $FC       ;ZEIGER HIGH NACH $FC
      A C00A 20 FD AE          JSR $AEFD     ;CHKKOM
      A C00D 20 9E B7          JSR $B79E     ;GETBYT => X=STRINGANZAHL
Stringdescriptoren holen
      A C010 A0 02             LDY #$02      ;DREI DURCHGÄNGE
      A C012 B1 FB             LDA ($FB),Y   ;DESCRIPTORBYTE LESEN
      A C014 99 61 00          STA $0061,Y   ;UND NACH $0061,Y
      A C017 88                DEY           ;ZÄHLER DEKREMENTIEREN
      A C018 10 F8             BPL $C012     ;SPRUNG, WENN NOCH
                                             ;NICHT FERTIG
String ausgeben
      A C01A A0 00             LDY #$00      ;ZÄHLER INITIAL.
      A C01C B1 62             LDA ($62),Y   ;STRINGZEICHEN LESEN
      A C01E 20 D2 FF          JSR $FFD2     ;UND AUSGEBEN
      A C021 C8                INY           ;NÄCHSTES ZEICHEN
      A C022 C4 61             CPY $61       ;STRING KOMPLETT
                                             ;AUSGEGEBEN?
      A C024 D0 F6             BNE $C01C     ;SPRUNG, WENN NICHT FERTIG
Nächster String
      A C026 A9 0D             LDA #$0D      ;ASCII-CODE VON 'RETURN'
      A C028 20 D2 FF          JSR $FFD2     ;AUSGEBEN
                                             ;(=>ZEILENVORSCHUB)
      A C02B 18                CLC           ;ADDITION VORBEREITEN
      A C02C A5 FB             LDA $FB       ;UND $FB/$FC UNTER
```

```
A C02E 69 03        ADC #$03     ;BERÜCKSICHTIGUNG EINES
A C030 85 FB        STA $FB      ;EVENTUELL ENTSTEHENDEN
A C032 90 02        BCC $C036    ;ÜBERTRAGS UM DREI
A C034 E6 FC        INC $FC      ;ERHÖHEN.
A C036 CA           DEX          ;STRINGZÄHLER
                                 ;DEKREMENTIEREN
A C037 D0 D7        BNE $C010    ;=> NÄCHSTEN STRING
                                 ;AUSGEBEN
A C039 60           RTS          ;ZURÜCK NACH BASIC
```

2.21 Ein großes Programmprojekt: Die INPUT#-Routine

Sie wissen nun alles, was für den Lesezugriff auf beliebige Variablentypen benötigt wird. Mit GETPOS können Sie zudem numerische Variablen anlegen und mit beliebigen Werten beschreiben (Integervariablen, da wir uns mit dem Fließkomma-Format nicht auskennen).

Das letzte Kapitel bei der Variablenbehandlung ist jedoch noch offen, das Anlegen von Strings. In diesem Abschnitt werden außer diesem Thema noch weitere, häufig sehr nützliche Routinen des Betriebssystems behandelt.

Alle diese Routinen gipfeln im "krönenden Abschluß" dieses Buches, einer "INPUT#"-Routine, die nicht nur nützlich ist, sondern alles bisher Erlernte anwendet.

Wenn Sie mit diesem Programm - und meinen Erläuterungen - zurecht kommen, haben Sie das unseren Assembler-Kurs "bestanden". In diesem Fall dürfen Sie mit Recht behaupten, ein fortgeschrittener Assembler-Programmierer zu sein.

Dennoch sollten Sie auch die folgenden Kapitel noch lesen, in denen Ihnen nicht nur eine weitere, sehr nützliche Interpreter-Routine gezeigt wird, sondern auch auf die "allerletzten" - allerdings nur sehr selten in der Praxis verwendeten - Assembler-Befehle eingegangen wird.

STRRES und FRETOP

Den BASIC-Befehl INPUT# kennen Sie sicherlich. Er dient zum Einlesen von Daten von Diskette. Beim Einlesen von Strings besitzt dieser Befehl jedoch einige Schwächen:

- Die zu lesenden Strings dürfen nicht länger sein als 80 Zeichen.
- Die Strings dürfen verschiedene Sonderzeichen nicht enthalten (":", ";", ","), auf die in der Praxis jedoch kaum verzichtet werden kann.

Eine Möglichkeit, dieses Problem zu lösen, besteht in der Verwendung von GET#. Mit GET# kann - mit einer Schleife - ein beliebig langer String mit beliebigen Zeichen gelesen werden. Leider ist dieser Vorgang jedoch extrem langsam. Es bietet sich daher an, INPUT# durch eine Assembler-Routine zu ersetzen.

Zwei Probleme müssen wir zuvor lösen, den Zugriff auf Cassette/Diskette von Assembler aus, und die Frage, wie wir einen String anlegen.

STRRES ($B4F4)

Die Routine STRRES besitzt mehrere Aufgaben. Sie prüft, ob ausreichend Platz für den anzulegenden String vorhanden ist. Die Stringlänge (Zeichenanzahl) wird im Akkumulator übergeben, wenn STRRES aufgerufen wird.

Wenn nicht ausreichend Platz vorhanden ist, führt STRRES eine "Garbage Collection" durch, das heißt, überflüssiger "Stringmüll" wird entfernt, der String-Stack wird "aufgeräumt". Ist anschließend immer noch zuwenig Platz für den neu anzulegenden String vorhanden, wird die Fehlermeldung "OUT OF MEMORY ERROR" ausgegeben. STRRES sollte bereits aus diesem Grund immer aufgerufen, bevor ein String angelegt und dadurch eventuell die Variablentabelle überschrieben wird (erinnern Sie sich: die Variablentabelle wächst "aufwärts", der String-Stack "abwärts").

FRETOP ($33/$34)

FRETOP ist keine Interpreter-Routine, sondern ein Zeiger in der Zeropage, der auf das momentane - untere - Ende des String-Stacks weist. Jeder neue String wird "unterhalb" des letzten Strings angelegt.

STRRES vermindert den Zeiger FRETOP nach dem Aufruf automatisch um die Länge des anzulegenden Strings. Ein Beispiel: FRETOP weist auf die Adresse $5009. An $5009 befindet sich somit das erste Zeichen des zuletzt angelegten Strings. Wir rufen nun STRRES auf und übergeben im Akkumulator den Wert $05, da der anzulegende String aus fünf Zeichen besteht. Wenn ausreichend Speicherplatz vorhanden ist, vermindert STRRES den Zeiger FRETOP um den übergebenen Wert $05.

FRETOP weist nun auf die Adresse $5004, das erste Zeichen des neu anzulegenden Strings. Den String selbst können wir mit Hilfe dieses Zeigers anschließend problemlos in den reservierten Bereich ($5004-$5008) kopieren.

```
      LDY #$04
LOOP  LDA (STRING),Y
      STA (FRETOP),Y
      DEY
      BPL LOOP
```

Wenn wir wie geplant einen String von Diskette/Cassette einlesen, wissen wir leider noch nicht, wie lang der einzulesende String ist. Daher reservieren wir vorsichtshalber Platz in der maximalen Stringlänge von 255 Zeichen.

```
Stringplatz reservieren
      A C000 A9 FF            LDA #$FF     ;PLATZ FÜR EINEN
                                           ;255 ZEICHEN
      A C002 20 F4 B4         JSR $B4F4    ;LANGEN STRING RESERVIEREN
```

Wir können nun Zeichen für Zeichen einlesen und über den Zeiger FRETOP im reservierten Bereich speichern (STA (FRETOP),0 bis STA (FRETOP),$FF).

BASIN($FFCF), CHKIN($FFC6), CHKOUT($FFC9) und CLRCH($FFCC)

Zum Schreiben/Lesen von/auf Casette/Diskette benötigen wir verschiedene Betriebssystem-Routinen. Wie bei allen Routinen des Betriebssystems sind die Adressen bei allen Commdore-Rechnern identisch.

Die Routine BASIN liest ein Zeichen von einer "geöffneten logischen Datei". Diese logische Datei öffnen wir noch in unserem BASIC-Programm vor dem Aufruf der Assembler-Routine, zum Beispiel mit:

```
200 OPEN 2,8,2"TEST,S,R":REM LOGISCHE DATEI OEFFNEN
210 SYS 49152,2,A$:REM INPUT#-ROUTINE AUFRUFEN
```

Wie Sie von BASIC her wissen, wird beim Öffnen einer logischen Datei mit dem OPEN-Befehl eine "Filenummer" (auch "Dateinummer" genannt) angegeben (im Beispiel die Filenummer 2), auf die sich anschließende Lesebefehle beziehen müssen (INPUT#2,A$).

Unsere Assembler-Routine wird analog arbeiten. Der Startadresse folgt die im OPEN-Befehl verwendete logische Filenummer. Diese Filenummer wird mit GETBYT eingelesen und im X-Register der Routine CHKIN übergeben.

CHKIN "leitet Eingaben auf die logische Datei um". Zum Verständnis müssen Sie wissen, daß sich die Ausgabe-Routine BSOUT (Ausgabe eines Zeichens) und die Eingabe-Routine BASIN (Eingabe eines Zeichens) immer auf die sogenannten "Standardgeräte" beziehen, also auf Bildschirm und Tastatur. Wenn wir die Routine BASIN aufrufen, wird diese daher ein Zeichen von der Tastatur einlesen (ebenso wie BSOUT normalerweise ein Zeichen immer auf dem Bildschirm ausgibt).

Wollen wir ein Zeichen nicht von der Tastatur, sondern von Floppy oder Diskette lesen, öffnen wir eine logische Datei und geben der Routine CHKIN die verwendete Filenummer an. Alle folgenden Eingaben werden nun nicht mehr von der Tastatur, sondern von der betreffenden Datei erwartet, in unserem Fall

von der - vom BASIC-Programm geöffneten - Disketten- oder Cassetten-Datei. Wenn keine weiteren Lesezugriffe mehr stattfinden, wird die Routine CLRCH aufgerufen, die alle geöffneten "Übertragungs-Kanäle" schließt und die Ein-/Ausgabe wieder auf die Standardgeräte umleitet.

Übrigens: auch die Standard-Ausgaberoutine BSOUT können wir verwenden, um Zeichen auf ein beliebiges Peripherie-Gerät auszugeben. In diesem Fall öffnen wir ebenfalls zuvor (von BASIC aus) eine logische Datei und lenken mit der Routine CHKOUT ($FFC9) die Ausgabe auf diese Datei um (Filenummer im X-Register übergeben).

Jede Zeichenausgabe mit BSOUT bezieht sich anschließend auf die betreffende Datei.

Wir können nun den zweiten Programmteil erstellen, der den String einliest und Zeichen für Zeichen ab jener Adresse ablegt, auf die der Zeiger FRETOP ($33/$34) weist.

Zuerst wird mit CHKKOM und GETBYT die angegebene Filenummer gelesen und CHKIN aufgerufen, um die Eingabe auf die betreffende Datei umzuleiten.

Anschließend wird der String zeichenweise eingelesen und im reservierten Bereich abgelegt. Das Stringende wird an dem Zeichen $0D erkannt, dem RETURN-Code, den der BASIC-Befehl PRINT# automatisch an einen zu schreibenden String "anhängt".

Nach dem Einlesen des Strings enthält der Zähler Y die Stringlänge, die in eine Zeropage-Speicherzelle kopiert wird, da wir die Anzahl der eingelesenen Zeichen noch mehrmals benötigen werden.

```
String einlesen
      A C005 20 FD AE           JSR $AEFD     ;CHKKOM
      A C008 20 9E B7           JSR $B79E     ;GETBYT (=>X=FILENUMMER)
      A C00B 20 C6 FF           JSR $FFC6     ;CHKIN (MIT FILENUMMER
                                              ;IN X)
      A C00E A0 00              LDY #$00      ;ZÄHLER INITIALISIEREN
```

```
A C010 20 CF FF          JSR $FFCF    ;BASIN => ZEICHEN LESEN
A C013 C9 0D             CMP #$0D     ;UND MIT 'RETURN'
                                      ;VERGLEICHEN
A C015 F0 05             BEQ $C01C    ;FERTIG, WENN GLEICH
                                      ;'RETURN'
A C017 91 33             STA ($33),Y  ;SONST ZEICHEN SPEICHERN
A C019 C8                INY          ;ZÄHLER INKREMENTIEREN
A C01A D0 F4             BNE $C010    ;UND NÄCHSTES ZEICHEN
                                      ;LESEN
A C01C 84 FB             STY $FB      ;STRINGLÄNGE NACH $FB
                                      ;'RETTEN'
A C01E 20 CC FF          JSR $FFCC    ;UND CHKIN AUFRUFEN
```

Der String ist nun komplett eingelesen und befindet sich im reservierten, 255 Byte großen Bereich. Da der gelesene String jedoch wahrscheinlich weitaus kürzer als 255 Zeichen ist, betreiben wir eine enorme Platzverschwendung (stellen Sie sich vor, daß 255 Zeichen pro zu lesendem String reserviert werden und diese dann im Durchschnitt aus vielleicht 30 Zeichen bestehen!).

Daher werden wir den String nach dem Einlesen "nach oben" verschieben, an das Ende des reservierten Bereichs, und den Bereich darunter wieder "freigeben", indem wir FRETOP erhöhen, so daß dieser Zeiger anschließend auf den tatsächlichen Stringanfang weist.

Um den String zu verschieben, ermitteln wir die Differenz zwischen den reservierten 255 Byte und der echten Stringlänge und erzeugen eine Zeiger, der auf die Adresse weist, zu der der String verschoben werden muß.

```
Zeiger auf "echten" Stringanfang erzeugen
      A C021 A5 34           LDA $34      ;FRETOP-HIGH NACH
      A C023 85 FD           STA $FD      ;$FD KOPIEREN.
      A C025 A9 FF           LDA #$FF     ;DIFFERENZ ZWISCHEN $FF
                                          ;UND
      A C027 38              SEC          ;DER TATSÄCHLICHEN
                                          ;STRINGLÄNGE
      A C028 E5 FB           SBC $FB      ;ERMITTELN.
```

```
A C02A 18          CLC          ;DIESE DIFFERENZ
                                ;ZUM ZEIGER
A C02B 65 33       ADC $33      ;FRETOP ADDIEREN, IN
A C02D 85 FC       STA $FC      ;$FC/$FD EINEN ZEIGER
                                ;AUF DEN
A C02F 90 02       BCC $C033    ;KORREKTEN STRINGANFANG
                                ;ERZEUGEN
A C031 E6 F9       INC $FD      ;(ÜBERTRAG
                                ;BERÜCKSICHTIGEN).
```

Halten wir fest: FRETOP ($33/$34) weist auf den Beginn des reservierten 255-Byte-Bereichs, ab dem unser String abgelegt wurde. In $FB befindet sich die Stringlänge, und in $FC/$FD legten wir soeben einen Zeiger an, der auf jene Adresse weist, an der der String gespeichert werden sollte, um Platzverschwendungen zu vermeiden. Dieser Zeiger ergab sich, indem zu dem Inhalt von FRETOP die Differenz zwischen 255 und der tatsächliche Stringlänge addiert wurde. Der nächste Schritt besteht im Übertragen des Strings an jene Adresse, auf die unser Zeiger ($FC/$FD) weist.

```
String an endgültige Adresse verschieben
A C033 A4 FB       LDY $FB      ;ZÄHLER MIT STRINGLÄNGE
                                ;INITIALISIEREN
A C035 88          DEY          ;UND KORREKTUR VORNEHMEN
A C036 B1 33       LDA ($33),Y  ;ZEICHEN LESEN
A C038 91 FB       STA ($FB),Y  ;UND NACH ($FB),Y KOPIEREN
A C03A 88          DEY          ;ZÄHLER DEKREMENTIEREN
A C03B C0 FF       CPY #$FF     ;STRING KOMPLETT
                                ;VERSCHOBEN?
A C03D D0 F7       BNE $C036    ;SPRUNG, WENN NEIN
```

Der String befindet sich nun zwar an der korrekten Adresse, der BASIC-Interpreter "weiß" jedoch noch nichts von diesem String, da die String-Descriptoren fehlen.

Im letzten Programmteil müssen diese Descriptoren angelegt werden. Mit CHKKOM und GETPOS wird ein Zeiger auf die Stringvariable geholt, die im BASIC-Programm angegeben wurde

(SYS 49152,2,A$). Der Zeiger befindet sich im Akkumulator und im Y-Register und könnte nun wie üblich in der Zeropage abgelegt werden.

Was ich Ihnen jedoch bisher noch nicht verriet: nach dem Aufruf von GETPOS befindet sich der Zeiger nicht nur in den Registern. GETPOS legt den Zeiger zugleich in den Zeropage-Adressen $47/$48 ab.

Im folgenden Teil verwenden wir den bereits vorhandenen Zeiger auf die String-Descriptoren in $47/$48.

Der letzte Programmteil erzeugt die Stringdescriptoren, indem mit indirekt-indizierter Adressierung auf die drei Descriptor-Bytes zugegriffen wird (STA ($47),0; STA ($47),1); STA ($47), 2).

In das erste Descriptor-Byte - den Längendescriptor - wird die in $FB enthaltene Stringlänge kopiert. In die beiden folgenden Descriptor-Bytes - Zeiger auf den String - wird der Inhalt des Zeigers $FC/$FD kopiert, der die echte Anfangsadresse des Strings enthält.

Diese echte Anfangsadresse wird ebenfalls nach $33/$34 kopiert, also in FRETOP, den Zeiger auf das Ende des String-Stacks, der noch noch nicht korrigiert wurde und auf jene Position weist, an der sich der String vor dem Verschieben befand.

```
String-Descriptoren und FRETOP behandeln
      A C03F 20 FD AE           JSR $AEFD     ;CHHKOM
      A C042 20 8B B0           JSR $B08B     ;ZEIGER AUF STRINGVARIABLE
                                              ;($47/$48)
      A C045 A0 00              LDY #$00      ;ZEIGER Y AUF
                                              ;LÄNGENDESCRIPTOR LOW
      A C047 A5 FB              LDA $FB       ;STRINGLÄNGE IN
      A C049 91 47              STA ($47),Y   ;LÄNGENDESCRIPTOR KOPIEREN
      A C04B C8                 INY           ;ZEIGER AUF
                                              ;2.DESCRIPTORBYTE
      A C04C A5 FC              LDA $FC       ;STRINGADRESSE-LOW NACH
```

```
A C04E 91 47          STA ($47),Y  ;ADRESSENDESCRIPTOR
                                   ;LOW UND
A C050 85 33          STA $33      ;FRETOP-LOW KOPIEREN
A C052 C8             INY          ;ZEIGER Y AUF
                                   ;LÄNGENDESCRIPTOR HIGH
A C053 A5 FD          LDA $FD      ;STRINGADRESSE-HIGH NACH
A C055 91 47          STA ($47),Y  ;ADRESSENDESCRIPTOR
                                   ;HIGH UND
A C057 85 34          STA $34      ;FRETOP-HIGH KOPIEREN
A C059 60             RTS          ;ZURÜCK NACH BASIC
```

Die komplette INPUT#-Routine

```
Stringplatz reservieren
      A C000 A9 FF          LDA #$FF     ;PLATZ FÜR EINEN
                                         ;255 ZEICHEN
      A C002 20 F4 B4       JSR $B4F4    ;LANGEN STRING RESERVIEREN
String einlesen
      A C005 20 FD AE       JSR $AEFD    ;CHKKOM
      A C008 20 9E B7       JSR $B79E    ;GETBYT (=>X=FILENUMMER)
      A C00B 20 C6 FF       JSR $FFC6    ;CHKIN (MIT FILENUMMER
                                         ;IN X)
      A C00E A0 00          LDY #$00     ;ZÄHLER INITIALISIEREN
      A C010 20 CF FF       JSR $FFCF    ;BASIN => ZEICHEN LESEN
      A C013 C9 0D          CMP #$0D     ;UND MIT 'RETURN'
                                         ;VERGLEICHEN
      A C015 F0 05          BEQ $C01C    ;FERTIG, WENN GLEICH
                                         ;'RETURN'
      A C017 91 33          STA ($33),Y  ;SONST ZEICHEN SPEICHERN
      A C019 C8             INY          ;ZÄHLER INKREMENTIEREN
      A C01A D0 F4          BNE $C010    ;UND NÄCHSTES ZEICHEN
                                         ;LESEN
      A C01C 84 FB          STY $FB      ;STRINGLÄNGE NACH $FB
                                         ;'RETTEN'
      A C01E 20 CC FF       JSR $FFCC    ;UND CHKIN AUFRUFEN
Zeiger auf "echten" Stringanfang erzeugen
      A C021 A5 34          LDA $34      ;FRETOP-HIGH NACH
      A C023 85 FD          STA $FD      ;$FD KOPIEREN.
      A C025 A9 FF          LDA #$FF     ;DIFFERENZ ZWISCHEN
```

```
                                        ;$FF UND
      A C027 38           SEC           ;DER TATSÄCHLICHEN
                                        ;STRINGLÄNGE
      A C028 E5 FB        SBC $FB       ;ERMITTELN.
      A C02A 18           CLC           ;DIESE DIFFERENZ ZUM
                                        ;ZEIGER
      A C02B 65 33        ADC $33       ;FRETOP ADDIEREN UND
                                        ;SOMIT IN
      A C02D 85 FC        STA $FC       ;$FC/$FD EINEN ZEIGER
                                        ;AUF DEN
      A C02F 90 02        BCC $C033     ;KORREKTEN STRINGANFANG
                                        ;ERZEUGEN
      A C031 E6 F9        INC $FD       ;(ÜBERTRAG
                                        ;BERÜCKSICHTIGEN).
String an endgültige Adresse verschieben
      A C033 A4 FB        LDY $FB       ;ZÄHLER MIT STRINGLÄNGE
                                        ;INITIALISIEREN
      A C035 88           DEY           ;UND KORREKTUR VORNEHMEN
      A C036 B1 33        LDA ($33),Y   ;ZEICHEN LESEN
      A C038 91 FB        STA ($FB),Y   ;UND NACH ($FB),Y KOPIEREN
      A C03A 88           DEY           ;ZÄHLER DEKREMENTIEREN
      A C03B C0 FF        CPY #$FF      ;STRING KOMPLETT
                                        ;VERSCHOBEN?
      A C03D D0 F7        BNE $C036     ;SPRUNG, WENN NEIN
String-Descriptoren und FRETOP behandeln
      A C03F 20 FD AE     JSR $AEFD     ;CHHKOM
      A C042 20 8B B0     JSR $B08B     ;ZEIGER AUF STRINGVARIABLE
                                        ;($47/$48)
      A C045 A0 00        LDY #$00      ;ZEIGER Y AUF
                                        ;LÄNGENDESCRIPTOR LOW
      A C047 A5 FB        LDA $FB       ;STRINGLÄNGE IN
      A C049 91 47        STA ($47),Y   ;LÄNGENDESCRIPTOR KOPIEREN
      A C04B C8           INY           ;ZEIGER AUF
                                        ;2.DESCRIPTORBYTE
      A C04C A5 FC        LDA $FC       ;STRINGADRESSE-LOW NACH
      A C04E 91 47        STA ($47),Y   ;ADRESSENDESCRIPTOR
                                        ;LOW UND
      A C050 85 33        STA $33       ;FRETOP-LOW KOPIEREN
      A C052 C8           INY           ;ZEIGER Y AUF
                                        ;LÄNGENDESCRIPTOR HIGH
```

```
A C053 A5 FD        LDA $FD       ;STRINGADRESSE-HIGH NACH
A C055 91 47        STA ($47),Y   ;ADRESSENDESCRIPTOR
                                  ;HIGH UND
A C057 85 34        STA $34       ;FRETOP-HIGH KOPIEREN
A C059 60           RTS           ;ZURÜCK NACH BASIC
```

Ein BASIC-Demoprogramm

Ich stelle Ihnen nun ein BASIC-Programm vor, das die Anwendung der Routine demonstriert und einen 255 Zeichen langen String auf Diskette schreibt und mit der Assembler-Routine wieder einliest.

Der String enthält mehrere Kommata, ein Sonderzeichen, das beim Einlesen mit dem BASIC-Befehl INPUT# auf keinen Fall enthalten sein darf. Unsere Routine verkraftet jedoch beliebige Zeichen!

```
100 X$="123456789,"
110 FOR I=1 TO 25:Y$=Y$+X$:NEXT
120 Y$=Y$+"12345"
130 :
140 REM *** STRING SCHREIBEN ***
150 OPEN 1,8,2,"TEST,S,W"
160 PRINT#1,Y$
170 CLOSE 1
180 :
190 REM *** STRING LESEN ***
200 OPEN 1,8,2,"TEST,S,R"
210 SYS 49152,1,A$
220 CLOSE 1
230 PRINT A$
```

Wenn Sie anstelle einer Floppy eine Datasette verwenden, ändern Sie bitte die OPEN-Befehle in:

```
150 OPEN 1,1,1,"TEST"
200 OPEN 1,1,0,"TEST"
```

Denken Sie bei der Anwendung der Routine bitte daran, daß diese ausschließlich Strings einlesen kann! Numerische Variablen können mit dem normalen INPUT#-Befehl problemlos gelesen werden.

Zahlenausgabe mit dem BASIC-Interpreter

Bevor ich Sie mit Ihren frisch erworbenen Kenntnissen allein lasse, stelle ich Ihnen noch eine sehr nützliche Interpreter-Routine vor, mit der Zwei-Byte-Werte auf dem Bildschirm ausgegeben werden.

Die Ausgabe von Zahlen stellt ein ganz besonderes Problem in Assembler dar. Zahlenausgaben sind in einem Video-Spiel nötig (Punktestand, Spieleranzahl etc.), in Textverarbeitungen (Spalten- und Zeilenposition, freier Platz im Textspeicher etc.) und vielen weiteren Programmen.

Um eine Integerzahl auszugeben, müssen wir jedoch ein "riesiges" Programm schreiben, das diese Zahl "aufspaltet" und die einzelnen Werte in den Bildschirm- oder ASCII-Code umwandelt.

Bevor Sie sich diese Mühe machen, empfehle ich Ihnen die Verwendung der Interpreter-Routine INTOUT, die beliebige Integerwerte auf dem Bildschirm ausgibt.

INTOUT ($BDCD)

INTOUT gibt eine Integerzahl ab der aktuellen Cursorposition aus. Die Zahl wird im X-Register und im Akkumulator übergeben (X=Low-Byte; Akku=High-Byte). Ist die auszugebende Zahl kleiner als 256 (Ein-Byte-Wert), wird als High-Byte der Wert $00 im Akkumulator übergeben.

```
Zahlenausgabe
      A C000 A2 06          LDX #$06     ;LOW-BYTE $06
      A C002 A9 04          LDA #$04     ;UND HIGH-BYTE $04
```

```
A C004 20 CD BD          JSR $BDCD     ;AN 'INTOUT' ÜBERGEBEN
A C007 60                RTS           ;ZURÜCK NACH BASIC
```

Dieses Demoprogramm gibt die Zahl 1300 ($0406) auf dem Bildschirm aus. Das X-Register wird mit dem Low-Byte $06, der Akkumulator mit dem High-Byte $04 geladen und anschließend INTOUT aufgerufen.

Assembler-Programme nachladen

Sie sind nun in der Lage, Assembler-Routinen zu schreiben, die mit BASIC-Programmen zusammenarbeiten und von diesen aufgerufen werden.

Zur praktischen Nutzung müssen Sie jedoch noch wissen, wie ein BASIC-Programm eine Assembler-Routine "nachladen" kann.

Eine Möglichkeit, beide Programmteile zu laden, besteht darin, zuerst das Assembler-Programm mit dem Monitor oder von BASIC aus mit LOAD"(NAME)",8,1 zu laden.

Der Zusatz ",1" hinter der Geräteadresse (im Beispiel wurde die Floppy verwendet) ist nötig, um die Assembler-Routine "absolut" zu laden, das heißt nicht wie ein BASIC-Programm an den Anfang des BASIC-RAM's, sondern an jene Adresse, an der sich die Routine beim Speichern befand.

Nach dem Laden des Assembler-Programms müssen Sie unbedingt den Befehl NEW eingeben (der das Assembler-Programm nicht betrifft; es bleibt unverändert erhalten). Anschließend können Sie das BASIC-Programm wie üblich mit LOAD"(NAME)",8.

Paktischer als dieses "Laden per Hand" ist das erwähnte "Nachladen". Die erste Zeile Ihres BASIC-Programms sollte lauten:

```
100 IF A=0 THEN A=1:LOAD"(NAME)",8,1
```

Nach dem Starten des BASIC-Programms wird durch diese Zeile die Assembler-Routine (NAME) automatisch von Diskette "nachgeladen" (bei Verwendung der Datasette benutzen Sie die Geräteadresse 1).

Leider ist diese Form des Nachladens nicht möglich, wenn ein BASIC-Programm compiliert wird.

Ergänzungen

In diesem Buch wurden fast alle Assembler-Befehle behandelt, die Ihr Computer kennt. Einige wenige Befehle, die nur sehr selten benötigt und bisher nicht behandelt wurden, stelle ich Ihnen im folgenden Abschnitt vor.

Dieser Abschnitt dient ausschließlich der Vervollständigung. Die noch ausstehenden Befehle werden nur sehr selten benötigt und daher in diesem praxisorientierten Buch nur kurz "angerissen". Nähere Informationen entnehmen Sie bitte mehr theoretisch orientierten Lehrbüchern.

Der BIT-Befehl

Der BIT-Befehl kann manchmal in Programmen sehr vorteilhaft zum Testen bestimmter Bits verwendet werden. Der Inhalt des Akkumulators wird mit dem Inhalt der angegebenen Speicherzelle UND-verknüpft.

Sind alle Bits des Ergebnisses gelöscht, ist das Zero-Flag gelöscht, ansonsten gesetzt. Zusätzlich werden die Bits 6 und 7 des angegebenen Wertes in das - noch zu erläuternde - V-Flag und das N-Flag übertragen und können mit BVC/BVS beziehungsweise BPL/BMI getestet werden.

Der Inhalt des Akkumulators wird durch die Verknüpfung nicht verändert, sondern nur die erwähnten Flags beeinflußt! Beispiel:

```
LDA #$FF  ;$FF NACH
STA $033C ;$033C
LDA #$00  ;$00 MIT INHALT VON $033C 'UND'-VERKNÜPFEN,
BIT $033C ;BIT 6 UND 7 VON $033C INS V-FLAG UND N-FLAG.

Resultat: Zero-Flag gesetzt, Negativ-Flag gesetzt, V-Flag gesetzt.
```

BVC, BVS und CLV

Mit den Befehlen BVC ("branch on overflow clear", "verzweige, wenn kein Überlauf") und BCS ("branch on overflow set", "verzweige, wenn Überlauf") wird ein nicht behandeltes Flag des Status-Registers getestet, das V-Flag.

Außer durch den BIT-Befehl wird das V-Flag gesetzt, wenn ein Übertrag von Bit 6 nach Bit 7 stattfand (Beispiel: Addition zweier Werte mit gesetztem Bit 6).

BCS verzweigt, wenn das V-Flag gesetzt und BVC, wenn es gelöscht ist.

Mit CLV wird das V-Flag gelöscht (analog dem Löschen des Carry-Flags mit CLC).

SED und CLD

SED ("set decimal flag", "setze das Dezimal-Flag") setzt das Dezimal-Flag. Von nun an rechnet der Computer bei folgenden ADC- und SBC-Befehlen nicht mehr "binär", sondern dezimal, solange, bis mit CLD ("clear decimal flag", "lösche das Dezimal-Flag") das Dezimal-Flag wieder gelöscht wird.

Um zu verstehen, was mit "dezimalem" Rechnen gemeint ist, müssen Sie die sogenannten "BCD-Zahlen" kennen.

Da diese in der Programmierpraxis jedoch sehr bedeutungslos sind, werden Sie in diesem praxisorientierten Buch nicht behandelt.

TSX und TXS

Mit TSX ("transfer S into X", "übertrage den Stapelzeiger S in das X-Register") und TXS ("transfer X into S", "übertrage das X-Register in den Stapelzeiger S") kann die LIFO-Struktur des Stacks umgangen werden.

Der Stapelzeiger S ist ein Register, das immer die Adresse des obersten Stapelelements enthält. Der Inhalt dieses Registers kann mit TSX in das X-Register kopiert und anschließend weiterverarbeitet werden, um gezielt auf bestimmte Elemente des Stapels zuzugreifen (TSX : DEX : LDA $FF,X).

Mit TXS kann der Stapelzeiger umgekehrt gezielt beeinflußt und auf ein bestimmtes Element gesetzt werden (TSX : DEX : DEX : TXS).

2.22 Der Maschinensprachemonitor

Der hier erklärte und im Anhang auch zum Abtippen aufgeführte Maschinensprachemonitor verfügt über Möglichkeiten, die über den durchschnittlichen Standard hinausgehen. Aus diesem Grund könnte er auch für diejenigen unter Ihnen interessant sein, die bereits einen Maschinensprachemonitor besitzen.

Der abgedruckte Monitor liegt ab $9000 (36834). Er besitzt keinen Assemblebefehl "A". Stattdessen ändern Sie einfach die disassemblierten Assemblerbefehle oder setzen ein Komma, gefolgt von der gewünschten Startadresse und dem Befehl. Um also ein Programm ab $C000 einzugeben das mit LDA $0400 beginnt geben Sie ein:

```
C000 LDA $0400
```

Mit diesem Monitor ist es möglich, jede beliebige Speicherkonfiguration einzuschalten. Somit ist es möglich, auch die Speicherbereiche 'unter' dem BASIC-ROM, dem KERNAL-ROM und 'unter' dem I/O-Bereich auszulesen und zu verändern. Selbstverständlich können in diesen Bereichen auch Programme gestartet werden. Eine Besonderheit dieses Monitors ist, außer

einigen verbesserten Standardbefehlen, die Möglichkeit, Daten aus den eben erwähnten Bereichen mit Hilfe veränderter LOAD- und SAVE-Routinen zu laden und auch von dort zu speichern.

Zu beachten ist noch, daß alle Eingaben mit RETURN abgeschlossen werden müssen.

Jetzt werden wir näher auf die einzelnen Befehle eingehen:

C
COMPARE: Vergleichen von zwei Speicherbereichen. Nicht übereinstimmende Speicherzellen werden angezeigt.

```
Format:  C 1000 1800 2000
```

Der Bereich von $1000 bis $1800 wird mit dem Bereich ab $2000 verglichen.

D
Disassemblieren: Disassembliert den angegebenen Speicherbereich. Beim Schreiben eigener Programme brauchen Sie nur den bestehenden Befehl zu überschreiben.

```
Format:  D 1000 1100
```

Disassembliert den Speicherbereich von $1000 bis $1100. Wenn keine Bereichsbegrenzung angeben wird, wird nur die erste Speicherzelle angegeben.

F
Fill: Füllt den angegebenen Speicherbereich mit den nach der Speicherbereichangabe stehenden Bytes.

```
Format:  F 1000 2000 46 49 4C 4C
```

Der Speicherbereich von $1000 bis $2000 wird mit den Bytes $46, $49, $4C, $4C gefüllt, was dem Wort 'FILL' im ASCII-Code entspricht.

G
Goto start a Program: Startet ein Programm ab der angegebenen Adresse.

```
Format:  G C000
```

Das Programm wird ab der Adresse $C000 gestartet.

H
Hunt: Durchsucht den angegebenen Speicherbereich nach der angegebenen Bytefolge. Es ist auch möglich, nach teilweise unbekannten Bytefolgen zu suchen.

```
Format:  H E000 FFFF 4C ?? FF
```

Durchsucht den Speicherbereich von E000 bis FFFF. Die Fragezeichen stehen für ein unbekanntes Byte.

L
Load: Lädt ein Programm in den Speicher.

```
Format:  L "FILENAME"
```

Lädt ein Progamm absolut von Disk in den Speicher.

```
Format:  L "FILENAME",08,1000
```

Lädt ein Programm von Disk ab der Adresse $1000 in den Speicher. Die auf Disk angegebene Adresse wird ignoriert. Die 08 ist die Geräteadresse. Soll ein Programm von Kassette geladen werden, so muß auf das zweite Anführungszeichen eine 01 folgen.

```
Format:  L"FILENAME",07,D000
```

Lädt ein Programm von Disk immer in den RAM-Bereich des Rechners. Bei dieser Anweisung wird demnach nicht wie normalerweise der I/O-Bereich, sondern der darunterliegende RAM-Bereich beschrieben. Wenn keine Angabe der Startadresse

erfolgt, wird das Programm absolut geladen. Diese Anweisung ist nur von Disk möglich.

M
Memory-Dump: Zeigt die Speicherzellen des angegebenen Speicherbereichs und zusätzlich deren Umsetzung in Bildschirmzeichen an.

```
Format:   M 1000 2000
```

Zeigt den Speicherbereich von $1000 bis $2000 an.

P
Printer: Nach der Eingabe von P werden alle Ausgaben anstatt auf dem Bildschirm auf dem Drucker mit der Geräteadresse 4 ausgegeben. Nach erneuter Eingabe von P erfolgt die Ausgabe wieder auf den Bildschirm.

```
Format:   P
```

R
Register: Zeigt die letzten Registerwerte an.

PC	Programmzeiger
IRQ	Interruptvektor
SR	Prozessorstatusregister
AC	Akkumulator
XR	X-Register
YR	Y-Register
SP	Stapelzeiger

```
Format:   R
```

Die Register werden nach einem BREAK automatisch angezeigt.

S
SAVE: Speichert den angegebenen Bereich.

```
Format:   S"FILENAME",08,1000,2000
```

Speichert den Bereich von $1000 bis $2000 auf das Gerät mit der Geräteadresse 08 (Disk). Für das Speichern auf Kassette muß Geräteadresse 01 gewählt werden.

```
Format:   S"FILENAME",07,E000,FFFF
```

Speichert den angegebenen Bereich je nach Speicherkonfiguration auf Diskette ab. Somit ist es möglich, den RAM-Bereich unter Kernal-ROM, I/O-Bereich und BASIC-ROM abzuspeichern.

Diese Funktion existiert nur im Zusammenhang mit dem Gebrauch einer Diskettenstation.

T
Transfer: Kopiert den angegebenen Speicherbereich in einen anderen angegebenen Speicherbereich.

```
Format: T 1000 2000 1800
```

Kopiert den Bereich von $1000 bis $2000 nach $1800. Bei Bereichsüberschreitungen treten beim Kopieren, sowohl nach oben als auch nach unten, keine Fehler auf.

X
EXIT: Rücksprung ins BASIC

```
Format:   X
```

@
Disk Command: Gibt den Status der Floppy auf den Bildschirm aus.

```
Format:   @
```

Erfolgen auf das & weitere Zeichen, so entspricht dieses einem OPEN 1,8,15," wobei die nachfolgenden Zeichen an die Floppy übergeben werden.

```
Format:   @I
```

@$

```
Format:  @$
```

Gibt die Directory auf den Bildschirm aus.

+

Addition: Addiert zwei hexadezimale Zahlen. Diese Zahlen müssen immer vierstellig angegeben werden.

```
Format:  + 2000 0152
```

Addiert $2000 mit $0152 und gibt das Ergebnis aus.

-

Subtraktion: Subtrahiert zwei Hexadezimale Zahlen.

```
Format:  - 2000 0152
```

Subtrahiert $0152 von $2000 und gibt das Ergebnis aus.

$

Umrechnung vom Hexadezimalsystem ins Dezimalsystem

```
Format:  $1000
```

#

Umrechnung vom Dezimalsystem ins Hexadezimalsystem

```
Format:  #4096
```

Anmerkung: Fast alle Befehle lassen sich durch das Betätigen der RUN/STOP-Taste unterbrechen.

Wenn Sie das Betriebssystem vom Monitor aus ausgeschaltet haben und ein Programm starten, das mit BRK endet, so wird sich der Rechner unweigerlich 'aufhängen'. Das gleiche geschieht, wenn Sie mit ausgeschaltetem BASIC oder Betriebssystem den Monitor mit X verlassen.

Durch die Möglichkeit, eine beliebige Speicherkonfiguration einzustellen, läßt sich auch das Zeichensatz-ROM auslesen, kopieren und saven.

Der folgende BASIC-Loader erzeugt den beschriebenen Maschinensprachemonitor, der im Speicherbereich $9000 (36864) bis $9D10 (40208) liegt. Speichern Sie Ihn bitte nach dem Abtippen unbedingt ab. Wenn der Monitor fehlerfrei läuft, können Sie Ihn auch von sich selbst aus als Maschinensprache-Programm abspeichern. Er ist dann wesentlich kürzer und somit schneller zu laden. Nach dem Laden des so abgespeicherten Monitors setzen Sie die BASIC-Zeiger mit NEW zurück und starten ihn mit

```
SYS 9*4096
```

3. Die Grafik und ihre Programmierung

3.1 Der Video Interface Chip (VIC)

Die Grafikmöglichkeiten des C64, hochauflösende Grafik, Sprites, Farbe und der Textbildschirm, werden vom VIC gesteuert. In den folgenden Seiten werden wir auf die einzelnen Eigenschaften des VIC eingehen.

Doch zunächst ein allgemeiner Hinweis auf die Struktur des VIC, der unbedingt berücksichtigt werden muß. Der VIC beansprucht 16 kByte des von der CPU adressierbaren Bereichs, der insgesamt 64 kByte beträgt. Nach dem Einschalten des Computers liegt dieser Bereich von $0000 bis $3FFF.

Der VIC verwaltet:

- Video-RAM (Bildschirmspeicher)
- Farbspeicher
- Zeichengenerator
- Grafikspeicher
- Sprites

Es ist möglich, den Adreßbereich des VIC zu verschieben, jedoch nur in 16-KByte Schritten. Daraus ergibt sich, daß sich der VIC-Adreßbereich in vier verschiedenen Bereichen befinden kann.

Diese vier Möglichkeiten sind in der Tabelle auf der nächsten Seite aufgeführt. Der Bereich wird in der NMI-CIA 2, in der Adresse $DD00 (56576), festgelegt. In dieser Speicherzelle sind die beiden Bits 0 und 1 ausschlaggebend. Die Bits sind low-aktiv, das heißt, der Computer erkennt sie als gesetzt, wenn sie gelöscht sind und umgekehrt.

Hier sind die Speicherbereiche tabellarisch aufgeführt:

Adressbereich (Hex)			Adressbereich (Dez)			Bitmuster	Wert
$0000	-	$3FFF	0	-	16383	11	03
$4000	-	$7FFF	16384	-	32767	10	02
$8000	-	$BFFF	32768	-	49151	01	01
$C000	-	$FFFF	49152	-	65535	00	00

Abb. 3.1: Speicherbereiche

Möchten Sie zum Beispiel den Adreßbereich nach $8000 (32768) verlegen, dann geben Sie folgende Zeile ein:

```
10 POKE 56576, PEEK(56576) AND 252 OR 1: REM WERT= 01
```

Sie werden feststellen, daß der Bildschirm voller ungewöhnlicher Zeichen ist und die Eingaben, die Sie über die Tastatur machen, nicht erscheinen. Der Grund dafür ist, daß das Betriebssystem die eingegebenen Daten immer noch in den Bereich ab $0400 (1024) hineinschreibt. Um den verschobenen Bereich vollständig funktionsfähig zu machen, müssen noch einige andere Register berücksichtigt werden, die wir auf den folgenden Seiten erläutern.

3.2 Das Video-RAM

Das Video-RAM, auch Bildschirmspeicher genannt, hat zwei Aufgaben. Im normalen Modus dient es als Zeichenspeicher, in dem die momentan auf dem Bildschirm sichtbaren Zeichen stehen. Die Zeichen werden dort in Bildschirmcodes abgelegt. Dieser Code unterscheidet sich vom ASCII-Code. Das Video-RAM umfaßt 1024 Zeichen, von denen aber nur 40 mal 25, also 1000 Zeichen auf dem Bildschirm erscheinen. Die nächsten 16 Bytes, von $07E8 bis $07F7, sind unbenutzt. Die letzten acht Bytes, von $07F8 bis $07FF, dienen als Spritezeiger. Im zweiten

Modus, dem Grafikmodus, wird das Video-RAM als Farbspeicher für hochauflösende Grafik benutzt.

Das Verschieben des Video-RAM

Das Video-RAM kann innerhalb des Adreßbereichs des VIC, im Normalfall von $0000 (0) bis $3FFF (16383), ohne großen Aufwand verschoben werden.

Die möglichen Werte können Sie auch hier einer Tabelle entnehmen:

Adressbereich (Hex)	Adressbereich (Dez)	Bitmuster	Wert
$0000 - $03E7	0 - 999	0000	00
$0400 - $07E7	1024 - 2023	0001	01
$0800 - $0BE7	2048 - 3047	0010	02
$0C00 - $0FE7	3072 - 4071	0011	03
$1000 - $13E7	4096 - 5095	0100	04
$1400 - $17E7	5120 - 6119	0101	05
$1800 - $1BE7	6144 - 7143	0110	06
$1C00 - $1FE7	7168 - 8167	0111	07
$2000 - $23E7	8192 - 9191	1000	08
$2400 - $27E7	9216 - 10215	1001	09
$2800 - $2BE7	10240 - 11239	1010	10
$2C00 - $2FE7	11264 - 12263	1011	11
$3000 - $33F7	12288 - 13287	1100	12
$3400 - $37E7	13312 - 14311	1101	13
$3800 - $3BE7	14336 - 15335	1110	14
$3C00 - $3FE7	15360 - 16359	1111	15

Abb. 3.2: VIC-Adressbereiche

Dazu ein Beispiel:

Das Video-RAM wird in unserem Programmbeispiel nach $0C00 (3072) verschoben.

```
10 POKE 56576, PEEK (56576) AND 252 OR 3
20 POKE 53272, PEEK (53272) AND 15 OR 16*3 (Wert)
30 POKE 648,3072/256: REM BILDSCHIRMANFANG AUF $0C00 LEGEN
40 PRINT CHR$(147) : REM BILDSCHIRM LÖSCHEN
```

Beachten Sie bitte, daß Sie die Werte nicht direkt in die Speicherzellen schreiben können, sondern die Bits im alten Wert durch eine logische Verknüpfung ändern müssen.
In der Adresse $0288 (648) ist der Zeiger immer noch auf den alten Anfang des Video-RAM ausgerichtet. Deshalb muß dem Computer mitgeteilt werden, daß das Video-RAM verschoben wurde. Das wird erreicht, indem man in der oben genannten Speicherzelle das High-Byte des derzeitigen Video-RAM angibt. In unserem Beispiel wäre das der Wert $0C (12). Wenn Sie aber das Video-RAM, über die 16 kByte hinaus verschieben wollen, so muß der ganze Adreßbereich mitverschoben werden.

```
10 POKE 56576, PEEK (56576) AND 252 OR 1
```

Damit wird der Adreßbereich des VIC nach $8000 (32768) verschoben. Danach müssen Sie aber noch den Standort des Video-RAMs an den Adreßbereich anpassen. Sie brauchen nur zu der Startadresse der VIC-Bank die Startadresse des Video-RAM, die Sie der Tabelle entnehmen können, hinzuzuaddieren. In unserem Beispiel wäre das $8000 (32768) plus $0C00 (3072) = $8C00 (35840). Das High-Byte des Standorts, hier ist es $8C (140), müssen Sie wieder in die Speicherzelle $0288 (648) schreiben.

Das Programm sieht dann folgendermaßen aus:

```
10 POKE 56576, PEEK(56576) AND 252 OR 1
20 POKE 53272, PEEK(53272) AND 15 OR 16 * 3
```

```
30 POKE 648, 35840/256
40 PRINT CHR$(147) : REM BILDSCHIRM LÖSCHEN
```

Wenn Sie das Video-RAM zur Übung in einige Bereiche verschoben haben, werden Sie vielleicht festgestellt haben, daß Sie es in den Bereich von $1000 (4096) bis $1FFF (8191) und von $9000 (36864) bis $9FFF (40959) nicht verschieben können. Das liegt daran, daß in diesem Bereich das für den VIC zur Verfügung gestellte Zeichensatz-ROM liegt. Das RAM, das in den oben aufgefürten Bereichen liegt, wird jedoch davon nicht berührt. Lediglich für den VIC-Chip ist dieser RAM-Bereich nicht lesbar. In die Bereiche von $5000 (20480) bis $5FFF (24575) und $D000 (53248) bis $DFFF (57343) kann das Video-RAM frei verschoben werden, da sich dort kein Zeichensatz-ROM befindet. Das Fehlen des Zeichensatz-ROM setzt allerdings voraus, daß Sie beim Benutzen dieser Bereiche den Zeichensatz hineinkopiert haben.

3.3 Der Zeichengenerator

Das Aussehen der Zeichen, die wir auf dem Bildschirm sehen, ist, wie aus der Speicheraufteilung ersichtlich, in einem Zeichensatz-ROM in der Adresse $D000 (53248) unter dem I/O-Bereich abgelegt. Das Zeichensatz-ROM beansprucht einen Speicherbereich von vier kByte. Jedes Zeichen besteht aus acht Bytes, dessen Bits eine Acht mal Acht-Matrix bilden. Das heißt, daß jedes Zeichen eine Auflösung von 64 Punkten hat. Die gesetzten Bits entsprechen der Farbe, die für das Zeichen im Farb-RAM spezifiziert wurde. Die gelöschten Bits nehmen die momentane Hintergrundfarbe an.

```
        01234567
Byte 0.**..**. = 01100110 = $66 (102)
Byte 1.**..**. = 01100110 = $66 (102)
Byte 2.**..**. = 01100110 = $66 (102)
Byte 3.******. = 01111110 = $7E (126)
Byte 4.**..**. = 01100110 = $66 (102)
Byte 5.**..**. = 01100110 = $66 (102)
Byte 6.**..**. = 01100110 = $66 (102)
Byte 7........ = 00000000 = $00   (0)
```

Da das Zeichensatz-ROM nicht beschreibbar ist, muß man es in das RAM kopieren, um es dort verändern zu können.

Da der VIC nur auf den 16 kByte großen Adreßbereich, in dem er sich momentan befindet, zugreifen kann, muß der Zeichensatz in diesen Bereich kopiert werden. Hier sind die möglichen Bereiche des Zeichensatzes tabellarisch aufgeführt. Die Adressen sind zu der Speicherbankstartadresse hinzuzuaddieren.

Adressbereich (Hex)			Adressbereich (Dez)			Bitmuster	Wert
$0000	-	$0FFF	0	-	4095	0001	01
$1000	-	$1FFF	4096	-	8191	0101	05
$2000	-	$2FFF	8192	-	12287	1001	09
$3000	-	$3FFF	12288	-	16383	1101	13

Abb. 3.3: Zeichensatzbereiche

Am günstigsten ist es, sofern mit BASIC gearbeitet wird, den Zeichensatz in den Bereich von $3000 (12288) zu kopieren. Folgende Beispielroutinen sollen Ihnen die Arbeitsweise mit dem Zeichensatz verdeutlichen.

Möchten Sie den Zeichensatz von BASIC aus in das RAM kopieren, verfahren Sie wie folgt:

```
5 WERT = 13
10 POKE 56334, PEEK (56334) AND 254
20 POKE 1, PEEK (1) AND 251
30 FOR I=53248 TO 57343
40 POKE I- 40960, PEEK (I)
50 NEXT I
60 POKE 1, PEEK(1) OR 4
70 POKE 56334, PEEK (56334) OR 1
80 POKE 53272, PEEK (53272) AND 240 OR WERT
```

Hier das Programm für Assemblerprogrammierer:

```
C000 SEI          ; Interrupt verhindern
C001 LDA $01
C003 AND #$FB     ; Bit 3 Löschen (I/O ausschalten)
C005 STA $01
C007 LDX #$00
C009 LDY #$10
C00B LDA $D000,X  ; ROM ins RAM kopieren
C00E STA $3000,X
C011 INX
C012 BNE $C00B
C014 INC $C00D    ; Erstes  High Byte erhöhen
C017 INC $C010    ; Zweites High Byte erhöhen
C01A DEY
C01B BNE $C00B
C01D LDX #$D0
C01F LDY #$30
C021 STX $C00D    ; Erstes  High Byte zurücksetzen
C024 STY $C010    ; Zweites High Byte zurücksetzen
C027 LDA $01
C029 ORA #$04     ; Bit 3 setzten (I/O einschalten)
C02B STA $01
C02D LDA $D018
C030 AND #$F1     ; Bits 1 bis 3 löschen
C032 ORA #$0C     ; Bits 2 und 3 setzen
C034 STA $D018    ; Zeiger auf $3000 setzen
C037 CLI          ; Interrupt ermöglichen
C038 RTS          ; Programmende
```

In der Adresse $D018 (53272) dienen die unteren vier Bits als Zeiger auf den derzeitigen Standort des Zeichengenerators, wobei das erste Bit von der CPU immer gesetzt wird.

Es sind also eigentlich nur die Bits 1 bis 3 ausschlaggebend, wie auch aus der Tabelle ersichtlich.

Nun wollen wir ein eigenes Zeichen in den Zeichensatz einbinden. Das Zeichen könnte folgendermaßen aussehen:

```
      01234567
Byte 0.******. = $7E (126)
Byte 1*......* = $81 (129)
Byte 2*.*..*.* = $A5 (165)
Byte 3*......* = $81 (129)
Byte 4**....** = $C3 (195)
Byte 5*.****.* = $BD (189)
Byte 6*......* = $81 (129)
Byte 7.******. = $7E (126)
```

Um nun das Zeichen in den Zeichensatzgenerator, den wir bereits nach $3000 (12288) verlegt haben, einzubinden, geben wir die dezimalen Werte in eine DATA-Zeile ein. Das sieht dann folgendermaßen aus:

```
10 FOR I=12288 TO 12288+7 : READ A : POKE I,A : NEXT I
20 DATA 126,129,165,129,195,189,129,126
```

Nachdem Sie das Programm gestartet haben, erscheint statt des Klammeraffen unser Mondgesicht. Zu beachten ist aber, daß das zugehörige reverse Zeichen immer noch ein Klammeraffe ist.

3.4 Das Farb-RAM

Durch das Farb-RAM läßt sich jedes Zeichen in 16 verschiedenen Farben darstellen. Das Farb-RAM ist nicht verschiebbar und befindet sich daher immer im Adreßbereich von $D800 (55296) bis $DBFF (56319). Es umfaßt 1024 Bytes. Jedes Byte ist für die Zeichenfarbe eines Zeichens zuständig, wie jedes Byte des Video-RAM ein Zeichen auf dem Bildschirm bestimmt. Die einzelnen Bytes des Farb-RAM können die Werte von 0 - 15 annehmen, es lassen sich also 16 verschiedene Farben darstellen.

Die Farbe wird durch die ersten vier Bits festgelegt, also im LOW-Nibble. Die restlichen vier Bits, das HIGH-Nibble, sind unbenutzt. Das Farb-RAM hat auch noch andere Aufgaben, die wir im entsprechenden Zusammenhang noch erläutern werden. Bei einem Schreibzugriff auf das Farb-RAM, zum Beispiel mittels POKE 55296, FARBE, nimmt das entsprechende Zeichen (in diesem Fall das erste Zeichen in der ersten Zeile) die durch FARBE definierte Farbe an. Wollen Sie einem beliebigen Zeichen auf dem Bildschirm eine andere Farbe zuordnen, dann machen Sie das am besten mit

POKE 55296+ZEILE*40+SPALTE, FARBE.

Die Farb-Codes können Sie der Tabelle entnehmen:

Code		Farbe	Code		Farbe
Dez	Hex		Dez	Hex	
0	$00	schwarz	8	$08	orange
1	$01	weiss	9	$09	braun
2	$02	rot	10	$0A	hellrot
3	$03	türkis	11	$0B	grau1
4	$04	violett	12	$0C	grau2
5	$05	grün	13	$0D	hellgrün
6	$06	blau	14	$0E	hellblau
7	$07	gelb	15	$0F	grau3

Abb. 3.4: Farb-Codes

Diese Tabelle ist beim Programmieren mit Farben fast unentbehrlich.

3.5 Extended Background Color Mode

Durch das Einschalten des Extended Background Color Mode ist es möglich, mehr als eine Hintergrundfarbe für ein Zeichen zu benutzen. Beim Setzen von Bit 6 in der Adresse $D011 (53265) wird dieser Modus eingeschaltet. Die vier Hintergrundfarben werden in den Hintergrundfarbregistern $D021 (53281) bis $D024 (53284) festgelegt.

Dazu ein kleines Beipielprogramm:

```
5 A=1:B=5:C=9
10 POKE 53265, PEEK (53265) OR 64
20 FOR I=1024 TO 1024+254 :POKE I, I-1024 :NEXT I
30 POKE 53282, A :POKE 53283, B: POKE 53284, C
40 A=A+1 :B=B+1 :C=C+1 :GETA$ :IF A$="" THEN 30
50 POKE 53265, PEEK (53265) AND 191
```

Zur Erläuterung:

In Zeile 5 werden die Farben für die Hintergrundfarbregister festgelegt. In Zeile 10 wird der Extended Background Color Modus eingeschaltet, indem das sechste Bit der Adresse $D011 (53265) gesetzt wird. Zeile 20 schreibt 255 Zeichen in das Video-RAM. In Zeile 30 werden die Farben in den Hintergrundfarbregistern erhöht und anschließend wartet das Programm, bis eine Taste gedrückt wird. Dann verzweigt es in Zeile 40, wo wieder der Normalmodus eingeschaltet wird.

Wenn Sie das Programm gestartet haben, werden Sie erkennen, daß Sie im Extended Background Color Modus nur 64 Zeichen darstellen können. Das liegt daran, daß die Bits 6 und 7 auf den Zeichencodes im Video-RAM als Zeiger auf ein Hintergrundfarbregister dienen. Es bleiben also nur 6 Bits für den Zeichencode. Daher die Einschränkung auf 64 Zeichen.

Für die Bits 6 und 7 gibt es folgende Kombinationen:

Bitmuster	Bereich	Farbregister
0 0	0 – 63	$D021 (53281)
0 1	64 – 127	$D022 (53282)
1 0	128 – 191	$D023 (53283)
1 1	192 – 255	$D024 (53284)

Abb. 3.5: Bit-Kombinationstabelle I

Nachdem der Extended Background Color Modus durch das Löschen von Bit 6 in Adresse $D011 (53265) wieder ausgeschaltet wird, nehmen die Zeichen von 64 bis 255 wieder ihre ursprüngliche Form an, wie aus unserem Programmbeispiel ersichtlich.

3.6 Der Multicolormodus

Im Multicolormodus kann man innerhalb eines Zeichens vier verschiedene Farben darstellen. Die Farbinformation wird jeweils durch zwei Bits in der Matrix festgelegt. Die Bits können folgende Zustände annehmen:

Bitmuster	Farbregister
0 0	$D021 (53281)
0 1	$D022 (53282)
1 0	$D023 (53283)
1 1	Farb – RAM, Bits 0 bis 2

Abb. 3.6: Bit-Kombinationstabelle II

Da in diesem Modus zwei Bits einen Punkt auf dem Bildschirm ergeben, halbiert sich die Matrix dementsprechend. Die Farbe des Zeichens hängt von den Hintergrundfarbregistern ab. Die Bitkombinationen geben, wie auch aus der Tabelle zu entnehmen, nur das Farbregister an. In den Registern kann man dementsprechend die Farben des Zeichens bestimmen. Die dritte Zeichenfarbe wird im Farb-RAM, in den Bits 0 bis 2, angegeben. Das dritte Bit dient als Flag, ob das Zeichen im Multicolor- oder im normalen Modus dargestellt wird. Daher ist es möglich, beide Modi problemlos zu kombinieren.

Weil die Auflösung nur noch halb so groß ist (nämlich 4*8 Punkte pro Zeichen beziehungsweise 160*200 auf dem ganzen Bildschirm), ist es vorteilhaft, zwei oder mehr Zeichen für ein Objekt zu definieren. Unser Gesicht sieht zum Beispiel folgendermaßen aus:

```
        01234567
Byte 0 00000011 = $03 (003)
Byte 1 00111100 = $3C (060)
Byte 2 11000010 = $C2 (194)
Byte 3 11000000 = $C0 (192)
Byte 4 11000100 = $C4 (196)
Byte 5 11000001 = $C1 (193)
Byte 6 00111100 = $3C (060)
Byte 7 00000011 = $03 (003)


Byte 0 11111111 = $FF (255)
Byte 1 00000000 = $00 (000)
Byte 2 10000010 = $82 (130)
Byte 3 00000000 = $00 (000)
Byte 4 00000000 = $00 (000)
Byte 5 01010101 = $55 (085)
Byte 6 00000000 = $00 (000)
Byte 7 11111111 = $03 (003)

Byte 0 11000000 = $C0 (192)
Byte 1 00111100 = $3C (060)
```

```
Byte 2 10000011 = $83 (131)
Byte 3 00000011 = $03 (003)
Byte 4 00010011 = $13 (019)
Byte 5 01000011 = $43 (067)
Byte 6 00111100 = $3C (060)
Byte 7 11000000 = $C0 (192)
```

Um nun die drei Zeichen in den Zeichensatz einzubinden, muß vorher der Zeichensatz nach $3000 (12288) kopiert werden, wie es im Kapitel 3.3 beschrieben wurde.

Zur Verdeutlichung haben wir noch ein kleines Progamm erstellt:

```
5 A=0:B=7:C=5:D=10
10 POKE 53281, A
20 POKE 53282, B
30 POKE 53283, C
40 POKE 646, D : REM aktuelle Schriftfarbe
50 POKE 53270, PEEK (53270) OR 16
60 FOR I=12288 TO 12288+23: READ A: POKE I, A: NEXT I
70 PRINT CHR$ (147)
80 POKE 646, 5: PRINT "MULTICOLOR = ";: POKE 646, 10: PRINT "@AB":
PRINT
90 POKE 646, 5: PRINT "NORM.MODUS = ";: PRINT "@AB"
100 POKE 53283, A
101 IF A>15 THEN A=0
102 A=A+1
104 GET A$: IF A$="" THEN 95
110 POKE 53270, PEEK (53272) AND 239
200 DATA 003, 060, 194, 192, 196, 193, 060, 003, 255, 000,
130, 000, 000, 085, 000, 255, 192
210 DATA 060, 131, 003, 019, 067, 060, 192
```

In den Zeilen 10 bis 30 werden die Farben in den Farbregistern festgelegt. In Zeile 40 wird die Schriftfarbe für das Farb-RAM festgelegt. In der nächsten Zeile wird der Mulicolormodus ein-

geschaltet. In Zeile 60 werden die drei Zeichen in den Zeichensatz eingebunden. In den Zeilen 80 bis 90 werden die Zeichen in den verschiedenen Modi auf den Bildschirm gebracht. In den darauffolgenden Zeilen wird das erste Farbregister erhöht, bis ein Tastendruck erfolgt.

Wie Sie anhand unseres Beispiels sehen, kann man die Farben innerhalb der Matrix unabhängig voneinander verändern.

3.7 Sprites und ihre Programmierung

Sprites sind kleine Grafiken, die unabhängig vom Hintergrund über den Bildschirm bewegt werden. So kann man zum Beispiel Spielfiguren vor einem Hintergrund darstellen. Dieses Kapitel zeigt Ihnen, wie Sie vorgehen müssen, um ein Sprite zu programmieren.

Ein Sprite besteht aus 24 mal 21 Einzelpunkten. Um einen Sprite zu entwerfen, benutzen Sie daher am besten ein Raster aus 24*21 Kästchen. Jedes Kästchen entspricht einem Bildpunkt, der an- oder ausgeschaltet werden kann. Füllen Sie die Kästchen aus, die den Bildpunkten entsprechen, die gesetzt werden sollen.

Der nächste Schritt besteht darin, den entworfenen Sprite in Daten umzuwandeln.

Dazu werden jeweils acht Punkte zu einem Byte zusammengefaßt. Also besteht eine Zeile bei einem Sprite aus drei Bytes, der ganze Sprite hat 21 Zeilen mit je 3 Bytes, also 63 Bytes.

Die Acht-Bit-Zahlen müssen nun in Dezimalzahlen umgerechnet werden. Blocks, in denen gar kein Punkt gesetzt ist, haben den Wert 0.

Legen Sie die so gewonnen Daten in DATA-Zeilen ab. Wichtig ist, daß auch Bytes mit dem Wert 0 abgelegt werden, sonst verschieben sich alle anderen Daten. Wenn Sie alle Zahlen dreistellig schreiben, bleibt das Listing übersichtlicher:

```
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

Jede Zahl von 1 bis 255 repräsentiert ein bestimmtes Punktemuster innerhalb eines Sprites. So ist es also möglich, jede beliebige Figur, von einem Punkt bis hin zum 24 mal 21 Punkte großen Block zu programmieren. Der Unterschied zur hochauflösenden Grafik besteht darin, daß man ein Sprite frei bewegen, vergrößern und verändern kann. Doch bevor man das Sprite einschalten kann müssen die Spritezeiger festgelegt werden. Diese befinden sich in den Adressen von $07F8 (2040) bis $07FF (2047), in der Reihenfolge der Spritenummern. Der Spritezeiger für Sprite 1 steht also in Adresse $07F8 (2040), der für Sprite 2 in der Adresse $07F9 (2041) und so weiter. Wenn Sie zum Beispiel einen Sprite in den Speicherbereich ab $2000 (8192) ablegen wollen, dann verfahren sie wie folgt:

```
10 POKE 2040, 8192/64
```

Somit haben Sie den Spritezeiger auf die Adresse $2000 (8192) zeigen lassen. Weil ein Sprite 63 Bytes beansprucht, kann man den gewünschten Speicherbereich durch 64 teilen, dann hat man den Wert für den Spritezeiger. Mit 64 kann der Computer leichter rechnen, das eine überzählige Byte pro Sprite wird nicht benutzt. Anschließend werden dann die Daten für den Sprite mit Hilfe einer FOR- NEXT-Schleife in den Speicher gelesen:

```
20 FOR I=0 TO 62: READ A: POKE 8192+I, A: NEXT I
```

Jetzt muß der Sprite eingeschaltet werden. Dafür ist die Speicherzelle $D015 (53269) zuständig. Die Bits 0 bis 7 dienen als Schalter für das jeweilige Sprite. Wird das erste Bit gesetzt, schaltet sich Sprite 1 ein, wird das zweite Bit gesetzt, schaltet sich Sprite zwei ein und so weiter. Wenn alle Bits gesetzt werden, schalten sich alle acht Sprites ein. Die nächste Zeile lautet also:

```
30 POKE 53269, 1
```

Jetzt ist der Sprite zwar eingeschaltet, aber trotzdem noch nicht auf dem Bildschirm sichtbar, weil er sich in der oberen Ecke des Bildschirms (unter dem Bildschirmrahmen) befindet. Da die Position des Sprites auf Null steht, muß auch diese verändert werden. Die X-Position wird in der Adresse $D000 (53248), die Y-Position in der Adresse $D001 (53249) festgelgt. Für Sprite 2 sind es die Adressen $D003 (53250) und $D004 (53251) und so weiter. Man kann den Sprite auch mit einer FOR- NEXT-Schleife über den Bildschirm laufen lassen. Um sich das zu verdeutlichen, geben Sie folgendes ein:

```
40 POKE 53249, 100: FOR X=0 TO 255: POKE 53248, X: NEXT X
```

Ein Problem werden Sie vielleicht schon erkannt haben: Es gibt 320 horizontale Positionen, aber der maximale Wert bei einem POKE in die X-Position kann nur 255 betragen. Sie werden sich fragen, wie man den Sprite dann über den Wert 255 hinaus verschieben kann. Für diese Aufgabe gibt es ein weiteres Register: Im Register $D010 (53264) befindet sich für jedes Sprite ein weiteres Bit für die X-Position. In diesem Bit wird markiert, ob

eine X-Koordinate über 255 liegt. Zu diesem Zweck wird vor der Adressierung das entsprechende Bit auf 1 gesetzt. Damit unser Sprite auch bis Position 320 läuft, geben Sie die folgende Zeile ein:

```
50 POKE 53264, 1:FOR X=0 TO 64: POKE 53248, X:NEXT X:POKE 53264,0
```

Mit Hilfe dieser Einrichtung ist es nun leicht möglich, jeden Sprite beliebig und unabhängig von den anderern über den ganzen Bildschirm laufen zu lassen.

Als weitere Möglichkeit können wir noch die Farbe des Sprites ändern. Hierzu verfügt jeder Sprite über ein Farbregister. Diese Register befinden sich von $D027 (53287) bis $D08E (53294). Die Adresse $D027 (53287) ist also für Sprite 1 zuständig, die Adresse D028 (53288) für Sprite 2 und so weiter.

```
60 POKE 53287, 14
```

Dadurch hat unser Sprite nun eine hellblaue Farbe bekommen.

Die nächste Besonderheit ist die Möglichkeit, die Sprites in horizontaler und/oder vertikaler Richtung zu vergrößern. Auch hierfür gibt es zwei Register. Eines für die Vergrößerung in X- und das andere für die Vergrößerung in Y-Richtung. Die beiden Register sind vom Aufbau her identisch mit der Speicherzelle $D015. Die Bits 0 bis 7 sind für die jeweiligen Sprites zuständig. Das heißt, wenn Bit 0 gesetzt wird, vergrößert sich der Sprite 1 in X-beziehungsweise Y-Richtung, bei Bit 1 das zweite Sprite und so weiter. In der Adresse $D017 (53271) wird das entsprechende Sprite in X und in der Adresse $D01D (53277) in Y Richtung vergrößert. Jetzt können wir unser Programm erweitern:

```
70 POKE 53271, 1
80 FOR I=1 TO 700: NEXT I: REM WARTESCHLEIFE
90 POKE 53277, 1
```

Eine weitere Besonderheit ist, daß Sie wählen können, ob der Sprite vor dem Hintergrund oder dahinter plaziert werden soll.

Damit kann man natürlich schöne Effekte hervorrufen. Diese Priorität kann man in der Adresse $D01A (53275) festlegen. Auch hier ist das Prinzip zur Festlegung der Spritenummer mit der Adresse $D015 indentisch. Um eine Reaktion erkennen zu können, müssen an der Stelle, wo sich das Sprite befindet, Zeichen auf dem Bildschirm stehen. Mit dem Register können Sie die Priorität jedes Sprites beliebig verändern. Wenn das entsprechende Bit nicht gesetzt ist, also 0 ist, bedeutet das, daß das Sprite vor dem Hintergrund steht. Ein gesetztes Bit bewirkt demnach den umgekehrten Fall. Die nächste Zeile lautet:

```
100 POKE 53275, 1
```

Es gibt ein Register, in dem eine Kollision zwischen verschiedenen Sprites verzeichnet wird. Dieses Register bleibt solange auf 0, bis entweder zwei oder mehrere Sprites zusammengestoßen sind. Diese Register setzen sich bei einer Kollision nicht selbständig zurück. Das muß also vom Programm besorgt werden. Wenn diese nicht zurückgesetzt werden, bleiben die Informationen über die Kollision bestehen, so daß eine darauffolgende nicht erkannt wird. Wenn man den Wert der Speicherzelle $D01E (53278) abfragt, kann man feststellen, welche Sprites miteinander kollidiert sind. Wenn bei PEEK(53278) der Wert 3 erscheint, dann haben die Sprites 1 und 2 eine Kollision gehabt. Die Bitstruktur des Registers braucht nicht weiter erläutert zu werden, da sie mit den anderen Registern identisch ist. Nach einer Abfrage muß das Register wieder durch POKE 53278, 0 zurückgestellt werden.

Genauso, wie eine Kollision von verschiedenen Sprites registriert wird, besteht die Möglichkeit, auf eine Kollision von Sprites mit dem Hintergrund abzufragen. Dazu dient die Speicherzelle $D01F (53279). Dieses Register wird wie Register $D01E (53278) behandelt. Die einzige Ausnahme ist das Ergebnis der PEEK - Abfrage. Das Resultat gibt nur Auskunft darüber, welcher Sprite mit einem Zeichen kollidiert ist. Nicht aber welches Zeichen es war und an welcher Position es sich befand. Diese Adresse muß auch wie das Register $D01E (53278) nach einer Kollision zurückgesetzt werden.

Wie beim Multicolorzeichensatz kann man einen Sprite auch in vier verschiedenen Farben darstellen. Durch das Setzen der einzelnen Bits in der Adresse $D01C (53276) kann man für jeden einzelnen Sprite den Multicolormodus einschalten. Wenn Sie in Zeile 5 POKE 53276, 3 eingeben, dann nehmen unsere Hubschrauber langsam Gestalt an. Dadurch haben die Hubschrauber aber (wie auch der Zeichensatz im Vierfarbmodus) eine geringere Auflösung, da jeweils zwei Bits einen Punkt auf dem Bildschirm ergeben.

Wir wissen ja, daß zwei Bits vier Informationen übermitteln können: 00, 01, 10, 11. Bei der Verwendung des Multicolormodus haben diese Bits folgende Wirkung:

Bitmuster	Farbregister
0 0	Der Punkt hat die Farbe des Hintergrundes (Man sieht keinen Punkt)
0 1	Die Farbe wird aus Adresse $D025 (53285) geholt
1 0	Die Farbe wird aus den Adressen D026 (53286) bis D02E (53294) geholt
1 1	Die Farbe wird aus Adresse $D026 (53286) geholt

Abb. 3.7: Bit-Kombinationstabelle III

Vieleicht haben Sie sich anfangs gefragt, warum der Hubschrauber so seltsam aussieht. Diese Frage kann man leicht beantworten. Er ist als Multicolorsprite entwickelt worden. Da ein Multicolorsprite aus weniger Punkten besteht, als ein einfarbiger Sprite, sieht er natürlich im normalen Sprite- Modus etwas seltsam aus. Als Abschluß dieses Kapitels wollen wir Ihnen nun ein Demo- Programm zur Verdeutlichung der Spritefunktionen präsentieren, in dem alle erwähnten Funktionen vorkommen.

Es ist nicht ungeschickt, anhand dieses Programmes, mit der Programmierung der Sprites zu experimentieren. Dadurch lernen Sie die Handhabung der Sprites am schnellsten. Die REM-Zeilen müssen nicht mit eingegeben werden.

```
0 POKE 53280, 0: POKE 53281, 0: REM HINTERGRUND FARBE SETZEN
1 POKE 53286, 4: REM MULTICOLOR FARBE SETZEN
2 FOR T=0 TO 62: READ X: POKE 8192+T, X: NEXTT: REM SPRITE-
DATEN EINLESEN
4 PRINT CHR$(147): REM BILDSCHIRM LÖSCHEN
5 POKE 53276, 3: REM MULTICOLORMODUS EINSCHALTEN
6 POKE 53287, 1: POKE 53288, 1: REM FARBE SETZEN
7 FOR I=0 TO 24: POKE 1024+I*40+20, 161: NEXTI: REM MAUER AUFBAUEN
10 POKE 2040, 8192/64: POKE 2041, 8192/64: REM SPRITEZEIGER SETZEN
20 POKE 53269,3: REM SPRITE 1 UND 3 EINSCHALTEN
30 POKE 53249, 100: POKE 53251, 150: FOR I=1 TO 255
40 POKE 53248, I: REM SPRITE 1 IN X-RICHTUNG BEWEGEN
50 POKE 53250, I: POKE 53251, I: REM SPRITE 2 IN X UND Y RICHTUNG
BEWEGEN
60 IF PEEK(53278)=3 THEN GOSUB 100: REM ABFRAGE AUF SPRITE-
SPRITE KOLLISION
61 IF PEEK(53279)=3 THEN GOSUB 300: REM
ABFRAGE AUF SPRITE-HINTERGRUND-KOLLISION
65 IF PEEK(53248)=255 THEN GOSUB 200: REM WENN 255, DANN 200
70 NEXT I
80 GOTO 30
100 REM SPRITES IN X UND Y RICHTUNG VERGRÖSSERN
101 REM UND REGISTER FÜR SPRITE KOLLISION ZURÜCKSETZEN
102 POKE 53271, 3: POKE 53277, 3: POKE 53278, 0
110 RETURN
200 POKE 53264, 3: REM SPRITES AUF POSITION 256 SETZEN
210 FOR I=1 TO 64: POKE 53248, I: POKE 53250, I: POKE 53251, I: NE
XT I: REM WEITERBEWEGEN
220 POKE 53264, 0
230 RETURN
300 REM SPRITES VERKLEINERN
301 REM UND REGISTER FÜR SPRITE-HINTERGRUND-KOLLISION ZURÜCKSETZEN
302 POKE 53271, 0: POKE 53277, 0: POKE 53279, 0
310 RETURN
999 REM SPRITE DATEN
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
```

```
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

3.8 Der Grafikbildschirm

Eine weitere Besonderheit des C64 ist der Grafikbildschirm, das heißt die hochauflösende Grafik. Im einfarbigen Modus hat der Grafikbildschirm eine Auflösung von 64000 Punkten, also 8*40 = 320 horizontal und 8*25 = 200 vertikal, was die gerade genannte Anzahl an Einzelpunkten ergibt. Das Einschalten des Modus erfolgt durch Setzen von Bit 5 in der Adresse $D011 (53265). Es ist ratsam, die Bits mittels logischer Verknüpfungen wie AND und OR zu setzen beziehungsweise zu löschen, da die anderen Bits in der Speicherzelle noch über weitere Aufgaben verfügen. Da der VIC nur auf einen adressierbaren Bereich von 16 kByte zugreifen kann, liegt der 8 kByte große Grafikbildschirm immer in einer Hälfte des Bereichs. Durch das Setzen oder Löschen von Bit 3 in der Adresse $D018 (53272) können folgende Bereiche für den Standort des Grafikbildschirms gewählt werden:

Bitmuster	Adressbereich (Hex)	Adressbereich (Dez)
0	$0000 - $1F3F	0 - 7999
1	$2000 - $3F3F	8192 - 16191

Abb. 3.8: Bereichstabelle

Auch diese Bereiche sind zu der jeweiligen Bankstartadresse, also der Anfangsadresse des VIC, hinzuzuaddieren.

Der Grafikbildschirm ist genauso wie der Zeichensatz aufgebaut. Der VIC interpretiert den Zeichensatz genauso wie den Grafikbildschirm. Beim Zeichensatz ergeben 64 Punkte ein Zeichen auf dem Bildschirm. Beim Grafikbildschirm ist das genauso, nur man muß sich vorstellen, der ganze Bildschirm wäre voller Leerzeichen, dessen Punkte man nach Belieben setzen und wieder löschen kann. Man kann sich den Grafikbildschirm als ein riesiges Zeichen vorstellen, das eine Auflösung von 64000 Punkten hat.

Das Einschalten des Grafikbildschirms erfolgt durch:

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

Jetzt muß noch dem VIC mitgeteilt werden, in welchem Adressbereich sich der Grafikbildschirm befindet. Im Normalfall ist das dritte Bit in der Adresse $D018 (53272) auf Null gesetzt, das heißt, der Grafikbildschirm liegt im Adressbereich von $0000 (0) bis $1F3F (7999), wie auch der Tabelle zu entnehmen ist. Dieser Bereich ist aber äußerst ungünstig, weil wichtige Speicherzellen in der Zeropage überschrieben werden können. Das Umsetzen des Zeigers erfolgt, wie schon erwähnt, durch das Setzen von Bit 3 in der Adresse $D018 (53272):

```
20 POKE 53272, PEEK(53272) OR 8
```

Dadurch ist der Zeiger für den Grafikbildschirm auf den Bereich von $2000 (8192) bis $3FEF (16191) gestellt.

Wenn der Grafikbildschirm eingeschaltet ist, dient das Video-RAM als Farbspeicher für die hochauflösende Grafik. Sie können innerhalb eines Feldes von 64 Punkten, also in Größe eines Zeichens, die Farbe beliebig angeben. Um Vorder- und Hintergrundfarbe festzulegen, muß man ein Byte in zwei Hälften aufspalten. Das geht folgendermaßen vor sich: Die Farbe der Punkte, zum Beispiel 5 (grün), wird mit 16 multipliziert und mit der Hintergrundfarbe, zum Beispiel 1 (weiß), addiert. Daraus ergibt sich dann der Wert, der beide Funktionen übernimmt:

```
30 FARBE=5: HINTERGRUND =1
40 FOR I=0 TO 254: POKE 1024+I, FARBE*16+HINTERGRUND:NEXT I
50 GOTO 50: REM SCROLLEN DES BILDSCHIRMS VERHINDERN
```

Wenn Sie das Programm starten, werden Sie feststellen, daß der Hintergrund eine weiße und die Punkte eine grüne Farbe annehmen. In Zeile 50 muß in eine Endlosschleife verzweigt werden, da die anschließende READY- Meldung den Bildschirm verschieben und neue Farben bewirken würde.

Wenn Sie den Grafikbildschirm eingeschaltet haben, werden Sie feststellen, daß dieser nicht gelöscht ist, sondern undefinierbare Punkte gesetzt sind. Um den Grafikbildschirm zu löschen, muß der Bereich von $2000 (8192) bis $3F3F (16191) mit Nullen gefüllt werden, weil in einer Null keine Bits gesetzt sind und daher auch keine Punkte zu sehen sind.

```
FOR I=8192 TO 16192: POKE I,0: NEXT I
```

Anschließend wollen wir Ihnen noch eine kleine Anwendung zeigen, die eine Sinuskurve auf den Bildschirm bringt. Anhand des Programmbeispiels können Sie sich mit der Programmierung des Grafikbildschirms vertraut machen. Die REM- Zeilen müssen nicht mit eingegeben werden.

```
10 REM SINUSPLOT
20 POKE 53265, PEEK(53265) OR 32: REM GRAFIKBILDSCHIRM EINSCHALTEN
30 POKE 53272, PEEK(53272) OR 8 : REM ZEIGER AUF 8192 LEGEN
40 FOR I=1024 TO 2027: POKE I, 80: NEXT I: REM FARBE SETZEN
50 FOR I=8192 TO 16191: POKE I,0: NEXT I: REM GRAFIKBILDSCHIRM LÖS
CHEN
60 FOR X=0 TO 318: Y=100: GOSUB 1000: NEXT X: REM X ACHSE ZEICHNEN
70 FOR Y=0 TO 199: X=160: GOSUB 1000: NEXT Y: REM Y ACHSE ZEICHNEN
80 FOR X=0 TO 319: Y=100-100*SIN(X*PI/160):GOSUB 1000: NEXT X: REM
SINUSKURVE ZEICHNEN
90 GOTO 90: REM ENDLOSSCHLEIFE
1000 OY=320*INT(Y/8)+(YAND7):REM BERECHNEN EINES PUNKTES
1010 OX=8*INT(X/8)
1020 MA=2^(7-(XAND7))
1030 AV=8192+OY+OX
1040 POKE AV, PEEK (AV) OR MA: REM PUNKT DARSTELLEN
1050 RETURN: REM RÜCKSPRUNG VON DER UNTERROUTINE
```

Wenn Sie das Programm vom Aufbau her verstanden haben, dann werden Sie auch bald Ihre eigenen mathematischen Funktionen auf diese Art und Weise darstellen können.

Die Programmierung von Kreisen oder Linien zwischen zwei Punkten ist beim C64 leider nicht so komfortabel wie bei einigen anderen Computern. So gibt es keinen Befehl, um eine Gerade von Punkt X nach Punkt Y zu ziehen, oder einen Befehl, einen Kreis mit einem bestimmten Radius an einem bestimmten Ort zu zeichnen. Deshalb haben wir ein kleines Graphikhilfsprogramm für Sie vorbereitet, das einige Grafikmöglichkeiten bietet. Mit diesem Programm können Sie schnell und komfortabel einfache Grafiken erzeugen. Das Grafikhilfsprogramm finden Sie am Ende des Kapitels.

3.9 Der Multicolorgrafikbildschirm

Der Multicolorgrafikbildschirm ist mit dem Multicolorzeichensatz zu vergleichen. Im Gegensatz zum normalen Modus hat man im Multicolormodus nicht nur zwei, sondern vier Farben zur

Verfügung. Jedoch muß man, wie beim Multicolorzeichensatz, die Hälfte der Auflösung opfern. Das heißt, man hat nur noch 160 horizontale und 200 vertikale Punkte zur Verfügung, weil zwei Bits einen Punkt auf dem Bildschirm ergeben. Durch Setzen von Bit 4 in der Adresse $D016 (53270) wird der Multicolormodus für den Grafikbildschirm aktiviert. Natürlich muß auch das Bit 5 in der Adresse $D011 (53265) gesetzt sein, da sonst der Grafikbildschirm nicht aktiviert ist. Genauso wie beim Multicolorzeichensatz kann man innerhalb eines Acht- mal Acht-Feldes 3 Farben plus der Hintergrundfarbe darstellen. Da zwei Bits, die nun für einen Punkt zuständig sind, nur die Farbquelle angeben und nicht die Farbe selbst, ergeben sich folgende Möglichkeiten für die Farbquellen:

Bitmuster	Farbquelle
0 0	Hintergrundfarbregister $D021 (53281)
0 1	Highnibble der Videomatrix
1 0	Lownibble der Videomatrix
1 1	Farbram

Abb. 3.9: Mögliche Farbquellen

Das Farb-RAM, das beim normalen Grafikbildschirm unbenutzt ist, dient hier als Farbquelle für das Bitpaar 11. Es ist natürlich problemlos möglich, Sprites und Grafik zu kombinieren. Das Mischen von Text und Grafik ist nicht ohne größeren Programmieraufwand möglich, aber dies werden wir im Kapitel Interruptprogrammierung noch näher erläutern.

Hier nun das versprochene Grafikhilfsprogramm:

Sprungtabelle für Funktionen

```
C000 JMP $C01E  ; Grafikmodus einschalten
C003 JMP $C03C  ; Grafik löschen
C006 JMP $C051  ; Farbe setzen
C009 JMP $C06D  ; Grafik invertieren
C00C JMP $C089  ; Grafikpunkt setzen
C00F JMP $C086  ; Grafikpunkt löschen
C012 JMP $C152  ; Grafik laden
C015 JMP $C139  ; Grafik abspeichen
C018 JMP $C175  ; Grafik drucken
C01B JMP $C161  ; Grafik ausschalten
```

Routine zum einschalten der Grafik

```
C01E JSR $C03C  ; Sprung zu Grafik löschen
C021 LDA $D011  ; Normalwert
C024 STA $C170  ; sichern
C027 LDA $D018  ; Normalwert
C02A STA $C171  ; sichern
C02D LDA #$3B   ; Grafikmodus
C02F STA $D011  ; einschalten
C032 LDA #$18   ; Zeiger auf
C034 STA $D018  ; $2000 stellen
C037 LDX #$10   ; Farbwert laden
C039 JMP $C057  ; Sprung zu Farbe setzen
C03C LDY #$00   ; Adresse
C03E LDX #$20   ; festlegen
C040 STY $FD    ; und
C042 STX $FE    ; speichen
C044 TYA        ; Akku auf Null setzen
C045 NOP        ; keine Operation
C046 STA ($FD),Y; Bereich
C048 INY        ; von
C049 BNE $C046  ; $2000 bis
C04B INC $FE    ; $3FFF mit
```

```
C04D DEX        ; Nullen
C04E BNE $C046  ; füllen
C050 RTS        ; Rücksprung
```

Grafikfarbe setzen

```
C051 JSR $AEFD  ; auf Komma prüfen
C054 JSR $B79E  ; Wert ins X  Register holen
C057 LDY #$00   ; Adresse
C059 LDA #$04   ; festlegen
C05B STY $FD    ; Werte
C05D STA $FE    ; speichern
C05F TXA        ; Farbe in Akku schieben
C060 LDX #$04   ; Video-
C062 STA ($FD),Y; RAM
C064 INY        ; mit
C065 BNE $C062  ; Farb-
C067 INC $FE    ; wert
C069 DEX        ; auf-
C06A BNE $C062  ; füllen
C06C RTS        ; Rücksprung
```

Grafik invertieren

```
C06D LDY #$00   ; Bereich
C06F LDA #$20   ; festlegen
C071 STY $FD    ; und
C073 STA $FE    ; speichn
C075 LDX #$20   ; Zähler stellen
C077 LDA ($FD),Y; Wert laden
C079 EOR #$FF   ; alle Bits undrehen
C07B STA ($FD),Y; Wert speichern
C07D INY        ; ersten Zähler erhöhen
C07E BNE $C077  ; wenn nicht Null dann weiter
C080 INC $FE    ; ansonsten Adressee erhöhen
C082 DEX        ; zweiten Zähler erniedrigen
C083 BNE $C077  ; noch nicht Null?, dann weiter
C085 RTS        ; ansonsten Rücksprung
```

Punkt setzen beziehungsweise löschen

```
C086 LDA #$80    ; Zeiger für Grafikpunkt setzen
C088 BIT $00A9   ; Zeiger für Grafikpunkt löschen
C08B STA $97     ; Zeiger speichern
C08D JSR $AEFD   ; auf Komma prüfen
C090 JSR $B7EB   ; Werte laden
C093 CPX #$C8    ; Y-Koordinate größer als 199?
C095 BCS $C085   ; wenn ja, dann Rücksprung
C097 LDA $15     ; X-Koordinate +1 kleiner als 320?
C099 CMP #$01    ; dann
C09B BCC $C0A5   ; weiter
C09D BNE $C085   ; ansonsten Rücksprung
C09F LDA $14     ; X-Koordinate größer als 319
C0A1 CMP #$40    ; dann
C0A3 BCS $C085   ; Rücksprung
C0A5 TXA         ; Y-Koordinate in den Akku schieben
C0A6 LSR         ; durch
C0A7 LSR         ; 8
C0A8 LSR         ; mal
C0A9 ASL         ; 2
C0AA TAY         ; Y-Koordinate wieder ins Y-Register
C0AB LDA $C0FF,Y; Wert aus Tabelle holen
C0AE STA $C173   ; und speichern
C0B1 LDA $C100,Y; mal 320
C0B4 STA $C174   ; und speichern
C0B7 TXA         ; Y-Koordinate holen
C0B8 AND #$07    ; Bits 3 bis 7 löschen
C0BA CLC         ; Carry löschen und
C0BB ADC $C173   ; mit Tabellenwert addieren
C0BE STA $C173   ; und wieder speichern
C0C1 LDA $14     ; X-Koordinate laden
C0C3 AND #$F8    ; Bits 0 bis 2 löschen
C0C5 STA $C172   ; und speichern
C0C8 CLC         ; Carry löschen
C0C9 LDA #$00    ; mit Null
C0CB ADC $C173   ; addieren
C0CE STA $FD     ; und speichern
C0D0 LDA #$20    ; mit 32
```

```
C0D2 ADC $C174  ; addieren
C0D5 STA $FE    ; und speichern
C0D7 CLC        ; Carry löschen
C0D8 LDA $FD    ; Anfangsadresse mit
C0DA ADC $C172  ; Y-wert addieren
C0DD STA $FD    ; und speichern
C0DF LDA $FE    ; Anfangsadresse mit
C0E1 ADC $15    ; X-Wert addieren
C0E3 STA $FE    ; und speichern
C0E5 LDA $14    ; X-Wert laden
C0E7 AND #$07   ; Bits 3 bis 7 löschen
C0E9 EOR #$07   ; und umdrehen
C0EB TAX        ; X-Wert ins X-Register
C0EC LDA $C131,X; Wert aus Tabelle holen
C0EF LDY #$00   ; Zähler auf Null setzen
C0F1 BIT $97    ; Überprüfen ob der Punkt
C0F3 BPL $C0FA  ; gesetzt oder gelöscht werden soll
C0F5 EOR #$FF   ; alle Bits umdrehen
C0F7 AND ($FD),Y; Punkt löschen
C0F9 BIT $FD11  ; Punkt setzen
C0FC STA ($FD),Y; Punkt auf Bildschirm bringen
C0FE RTS        ; Rücksprung
```

Multiplikationstabelle

```
C0FF 00 00 40 01 80 02 C0 03
C107 00 05 40 06 80 07 C0 08
C10F 00 0A 40 0B 80 0C C0 0D
C117 00 0F 40 10 80 11 C0 12
C11F 00 14 40 15 80 16 C0 17
C127 00 19 40 1A 80 1B C0 1C
C12F 00 1E 01 02 04 08 10 20
C137 40 80 20 FD AE 20 D4 E1
```

Grafik speichern

```
C139 JSR $AEFD  ; auf Komma prüfen
C13C JSR $E1D4  ; Filenamen und Geräteadresse holen
C13F LDX #$00   ; End-
C141 LDY #$40   ; adresse
C143 LDA #$00   ; festlegen
C145 STA $FD    ; Anfangs-
C147 LDA #$20   ; Adresse
C149 STA $FE    ; speichern
C14B LDA #$FD   ; Sekundäradrese
C14D STA $B9    ; festlegen
C14F JMP $FFD8  ; Sprung zu SAVE
```

Grafik laden

```
C152 JSR $AEFD  ; auf Komma prüfen
C155 JSR $E1D4  ; Filenamen und Geräteadresse holen
C158 LDA #$01   ; Sekundäradresse
C15A STA $B9    ; festlegen
C15C LDA #$00   ; Flag für LOAD setzen
C15E JMP $FFD5  ; Sprung zu LOAD
```

Grafik Ausschalten

```
C161 LDA $C170  ; Normalwert laden
C164 STA $D011  ; und speichern
C167 LDA $C171  ; Normalwert laden
C16A STA $D018  ; und speichern
C16D JMP $E544  ; Bildschim löschen und Rücksprung
C170 BRK
C171 BRK
C172 BRK
C173 BRK
C174 BRK
```

Grafik drucken (für FX80 mit VC Interface)

```
C175 JSR $AEFD  ; Auf Komma prüfen
C178 JSR $B79E  ; logische Filenummer holen
C17B JSR $FFC9  ; Ausgabegerät setzen
C17E LDA #$00   ; Low und
C180 LDY #$20   ; High Byte laden
C182 STA $FD    ; Zeiger auf
C184 STY $FE    ; Grafik setzen
C186 LDX #$19   ; Anzahl
C188 LDY #$07   ; der
C18A BIT $05A0  ; Zeilen festlegen
C18D LDA $C1D6,Y; Drucker
C190 JSR $FFD2  ; auf
C193 DEY        ; Grafikmodus
C194 BPL $C18D  ; stellen
C196 LDA #$28   ; Spalte
C198 STA $15    ; festlegen
C19A LDA #$80   ; Maske
C19C STA $97    ; festlegen
C19E LDA #$00   ; Code
C1A0 STA $14    ; festlegen
C1A2 LDY #$07   ; Zähler festlegen
C1A4 LDA ($FD),Y; Bitmuster zusammen setzen
C1A6 AND $97    ; Vergleich ob Bit gesetzt
C1A8 BEQ $C1B1  ; wenn nicht, dann Zähler erniedrigen
C1AA LDA $14    ; wenn ja
C1AC ORA $C1DE,Y; dann Bit
C1AF STA $14    ; setzen
C1B1 DEY        ; Zähler ernidrigen
C1B2 BPL $C1A4  ; schon Null, wenn ja dann weiter
C1B4 LDA $14    ; Code an
C1B6 JSR $FFD2  ; Drucker senden
C1B9 LSR $97    ; Maske durch 2
C1BB BCC $C19E  ; wenn Null, dann Code festlegen
C1BD LDA $FD    ; Adresse
C1BF ADC #$07   ; um
C1C1 STA $FD    ; 8
C1C3 BCC $C1C7  ; erhöhen
C1C5 INC $FE    ; Highbyte erhöhen
```

```
C1C7 DEC $15    ; nächste
C1C9 BNE $C19A  ; Spalte
C1CB DEX        ; nächste
C1CC BNE $C18B  ; Zeile
C1CE LDA #$0D   ; Zeilenvorschub
C1D0 JSR $FFD2  ; ausgeben
C1D3 JMP $FFCC  ; Ausgabe wieder auf Bildschirm
```

Druckertabelle

```
C1D6 01 40 06 2A 1B 0D 31 1B
C1DE 80 40 20 10 08 04 02 01
```

Und hier noch ein Ladeprogramm in BASIC:

```
100 FOR I=49152 TO 49637
110 READ X: POKE I, X: S=S+X: NEXT I
120 DATA  76, 30,192, 76, 60,192, 76, 81,192, 76,109,192
130 DATA  76,137,192, 76,134,192, 76, 82,193, 76, 57,193
140 DATA  76,117,193, 76, 97,193, 32, 60,192,173, 17,208
150 DATA 141,112,193,173, 24,208,141,113,193,169, 59,141
160 DATA  17,208,169, 24,141, 24,208,162, 16, 76, 87,192
170 DATA 160,  0,162, 32,132,253,134,254,152,234,145,253
180 DATA 200,208,251,230,254,202,208,246, 96, 32,253,174
190 DATA  32,158,183,160,  0,169,  4,132,253,133,254,138
200 DATA 162,  4,145,253,200,208,251,230,254,202,208,246
210 DATA  96,160,  0,169, 32,132,253,133,254,162, 32,177
220 DATA 253, 73,255,145,253,200,208,247,230,254,202,208
230 DATA 242, 96,169,128, 44,169,  0,133,151, 32,253,174
240 DATA  32,235,183,224,200,176,238,165, 21,201,  1,144
250 DATA   8,208,230,165, 20,201, 64,176,224,138, 74, 74
260 DATA  74, 10,168,185,255,192,141,115,193,185,  0,193
270 DATA 141,116,193,138, 41,  7, 24,109,115,193,141,115
280 DATA 193,165, 20, 41,248,141,114,193, 24,169,  0,109
290 DATA 115,193,133,253,169, 32,109,116,193,133,254, 24
300 DATA 165,253,109,114,193,133,253,165,254,101, 21,133
310 DATA 254,165, 20, 41,  7, 73,  7,170,189, 49,193,160
320 DATA   0, 36,151, 16,  5, 73,255, 49,253, 44, 17,253
```

```
330 DATA 145,253, 96,  0,  0, 64,  1,128,  2,192,  3,  0
340 DATA   5, 64,  6,128,  7,192,  8,  0, 10, 64, 11,128
350 DATA  12,192, 13,  0, 15, 64, 16,128, 17,192, 18,  0
360 DATA  20, 64, 21,128, 22,192, 23,  0, 25, 64, 26,128
370 DATA  27,192, 28,  0, 30,  1,  2,  4,  8, 16, 32, 64
380 DATA 128, 32,253,174, 32,212,225,162,  0,160, 64,169
390 DATA   0,133,253,169, 32,133,254,169,253,133,185, 76
400 DATA 216,255, 32,253,174, 32,212,225,169,  1,133,185
410 DATA 169,  0, 76,213,255,173,112,193,141, 17,208,173
420 DATA 113,193,141, 24,208, 76, 68,229,  0,  0,  0,  0
430 DATA   0, 32,253,174, 32,158,183, 32,201,255,169,  0
440 DATA 160, 32,133,253,132,254,162, 25,160,  7, 44,160
450 DATA   5,185,214,193, 32,210,255,136, 16,247,169, 40
460 DATA 133, 21,169,128,133,151,169,  0,133, 20,160,  7
470 DATA 177,253, 37,151,240,  7,165, 20, 25,222,193,133
480 DATA  20,136,208,240,165, 20, 32,210,255, 70,151,144
490 DATA 225,165,253,105,  7,133,253,144,  2,230,254,198
500 DATA  21,208,207,202,208,189,169, 13, 32,210,255, 76
510 DATA 204,255,  1, 64,  6, 42, 27, 13, 49, 27,128, 64
520 DATA  32, 16,  8,  4,  2,  1
530 IF S<>60459 THEN PRINT "FEHLER IN DATAS !!" : end
540 PRINT "OK !"
```

3.10 Wie wende ich das Grafikhilfsprogramm an?

Das Grafikhilfsprogramm bietet einige Möglichkeiten, die Programmierung des Grafikbildschirms komfortabler und schneller zu gestalten. Der Aufruf der einzelnen Funktionen geschieht über SYS- Aufrufe, wobei bei einigen Befehlen noch weitere Parameter übergeben werden müssen.

Der Befehlssatz:

SYS 49152
Schaltet den Grafikbildschirm ein, wobei gleichzeitig der Grafikbildschirm gelöscht und die Farbe gesetzt wird.

SYS 49155
Löscht den Grafikbildschirm

*SYS 49158,FARBE*16+HINTERGRUND*
Für den angegebenen Wert wird die Punktfarbe mit 16 multipliziert und mit der Hintergrundfarbe addiert.

SYS 49161
Der Grafikbildschirm wird invertiert. Das heißt, die gesetzten Punkte werden gelöscht und die gelöschten gesetzt. So bekommen Sie eine negative Abbildung Ihres Bildes.

SYS 49164,X,Y
Ein Punkt wird gesetzt. Die X- und Y-Koordinaten werden durch Kommata getrennt. Die X-Koordinate kann von 0 bis 319 angegeben werden, die Y-Koordinate von 0 bis 199.

SYS 49167,X,Y
Ein Punkt wird gelöscht. Die Koordinaten werden wie beim obigen Befehl übergeben.

SYS 49170,"NAME",GA
Der Grafikbildschirm wird je nach Geräteadresse von Diskette oder Kassette geladen.

SYS 49173,"NAME",GA
Sichert den Grafikbildschirm je nach Geräteadresse auf Diskette oder Kassette.

SYS 49176,LF
Druckt den Grafikbildschirm aus. Die logische Filenummer wird nach dem Komma eingegeben. Die Hardcopyroutine ist für den Epson FX80 mit VC-Interface vorgesehen.

SYS 49179
Schaltet den Grafikbildschirm aus.

Mit dieser Grafikhilfe kann unser Sinusplotprogramm mit weniger Aufwand und schneller programmiert werden:

```
10 SYS 49152: REM GRAFIK EINSCHALTEN UND LÖSCHEN
20 SYS 49158,5*16+0: REM FARBE SETZEN
30 FOR X=0 TO 319: SYS 49164,X,100: NEXT X: REM X ACHSE ZEICHNEN
40 FOR Y=0 TO 199: SYS 49164,160,Y: NEXT Y: REM Y ACHSE ZEICHNEN
50 FOR X=0 TO 319: Y=100-100*SIN(X*PI/160): SYS 49164,X,Y: NEXT X:
REM SINUSKURVE
60 GET A$: IF A$="" THEN 60: REM AUF TASTE WARTEN
70 SYS 49179: REM GRAFIK AUSSCHALTEN
```

Sie werden festgestellt haben, daß das Zeichnen der Sinuskurve und der X-und Y-Achse durch die Assemblerroutinen erheblich beschleunigt wird. Sie können das Programm nach Bedarf noch erweitern, indem Sie zwischen der Zeile 60 und 70 die Grafik abspeichern oder ausdrucken.

3.11 Interruptprogrammierung

Das Wort "Interrupt" kommt aus dem Englischen und heißt Unterbrechung. Ein Interrupt in Bezug auf den 6510-Prozessor bedeutet eine Unterbrechung des normalen Programmablaufs durch andere Chips, zum Beispiel durch den VIC oder eine der CIAs.

Normalerweise wird jede 1/60 Sekunde vom Timer A ein Interrupt ausgelöst, indem ein Zähler heruntergezählt wird. Wenn der Zähler den Nullpunkt erreicht, erfolgt der Interrupt. Genauere Angaben finden Sie unter dem Kapitel Timer. Anschließend springt der Prozessor über den Vektor $FFFE (65534) und $FFFF (65535) in eine Betriebssystemroutine nach $FF48 (65352), wo er dann einige Register zwischenspeichert und anschließend über den Interruptvektor $0314 (788) und $0315 (789) nach $EA31 (59953) zur eigentlichen Interruptroutine verzweigt, um dort dann unter anderem das Blinken des Cursors und die Abfrage der Tastatur zu verwalten. Der Clou an der ganzen Sache ist der, daß man den Interrupt in seine eigene Maschinenroutine verzweigen lassen kann, und zwar über die Register $0314 (788) und $0315 (789). Die Adresse für die

Interruptroutine ist in diesen Registern in Low- und Highbyte abgelegt, und zwar steht in der Adresse $0314 das Low- und und in der Adresse $0315 (789) das Highbyte. Zu beachten ist, daß vor dem Ändern dieses Vektors das Auslösen einer Unterbrechung durch den Assembler-Befehl SEI verhindert werden muß, damit nicht die Gefahr besteht, daß der Prozessor schon nach Ändern von nur einem Byte dieses Zeigers versucht, in eine Interruptroutine zu verzweigen. Nach Ändern der beiden Adressen sollte man es dem Prozessor durch den Assembler-Befehl CLI wieder ermöglichen, Interrupts abzuarbeiten. Zu beachten ist außerdem noch, daß man seine eigene Routine mit einem Sprung in die Interruptroutine des Betriebssystems beenden sollte, also mit JMP $EA31.

Zu erwähnen ist noch, daß die hier aufgeführten Beispielprogramme mit dem abgedruckten Maschinensprachemonitor einzugeben sind.

Hierzu ein Beispiel:

```
C000 SEI       ; Interrupt verhindern
C001 LDA #$0D  ; Lowbyte
C003 STA $0314 ; setzen
C006 LDA #$C0  ; Highbyte
C008 STA $0315 ; setzen
C00B CLI       ; Interrupt wieder ermöglichen
C00C RTS       ; Programmende
```

Interruptroutine:

```
C00D INC $D020 ; Rahmenfarbe erhöhen
C010 JMP $EA31 ; Sprung zur System-Interruptroutine
```

Das Programm wird mit SYS 49152 gestartet. Nach dem Start blinkt die Rahmenfarbe, während Sie sich noch im normalen Basic befinden. Sie können den Zähler des Timers A, der den Interrupt auslöst, nach Belieben beschleunigen oder verlang-

samen; dieses geschieht, indem man den Startwert des Zählers erhöht oder erniedrigt. Das Highbyte des Zählers befindet sich in der Adresse $DC05 (56325). Wenn man durch POKE 56325, WERT den Zähler erhöht, wird der Interrupt entsprechend seltener ausgeführt. Wenn Sie den Zähler erniedrigen, wird der Interrupt öfter ausgeführt; nun muß man darauf achten, daß man den Zähler nicht zu sehr erniedrigt, da sonst der Interrupt zu häufig ausgeführt wird und dadurch der Ablauf anderer Programme erheblich verlangsamt und der Cursor zu sehr beschleunigt wird. Auch die LIST- Funktion würde sehr lange Zeit beanspruchen, um ein Basicprogramm aufzulisten.

Die Möglichkeiten des Interrupts sind noch lange nicht erschöpft. Es ist zum Beispiel möglich, die Kollision zwischen zwei oder mehr Sprites oder die Kollision von Sprites mit dem Hintergrund über den Interrupt abzufragen. Dies hat den Vorteil, daß die Abfrage um einiges genauer ist als die vom Basic aus. Wenn zwei Sprites miteinander kollidieren, wird, falls erlaubt, ein Interrupt ausgelöst, den man dann zu einer eigenen Routine verzweigen lassen kann.

Ein Interrupt vom VIC kann von folgenden Ereignissen ausgelöst werden:

- vom Rasterstrahl
- Kollision von Sprites
- Kollision von Sprites und Hintergrunddaten
- Lightpen

Doch zunächst wollen wir uns mit der Spritekollision beschäftigen.

Es gibt ein sogenanntes Interrupt Enable Register, wo festgelegt wird, von welchem Ereignis ein Interrupt ausgelöst werden soll. Dies geschieht, indem man das entsprechende Bit im Interrupt Enable Register setzt. Dadurch hat man angegeben, von welchem Ereignis der Interrupt ausgelöst werden soll. Folgende Bits im Enable Register entsprechen folgenden Ereignissen:

Bitmuster	Interruptquelle
0001	Rasterstrahl
0010	Kollision zwischen Sprites und Hintergrunddaten
0100	Kollision zwischen Sprites
1000	Lightpen

Abb. 3.11: Interrupt Modi

Freigegeben wird ein Interrupt, indem das entsprechende Bit im Enable Register, Adresse $D01A (53274), auf 1 gesetzt wird. Die restlichen Bits, also die Bits 4 bis 7, sind unbenutzt. Wenn nun durch ein Ereignis ein Interrupt ausgelöst wird, dann wird automatisch das entsprechende Bit im sogenannten Interrupt Latch Register, Adresse $D019 (53273), gesetzt. Gleichzeitig wird bei jedem der oben aufgezählten Ereignisse Bit 7 von $D019 ebenfalls gesetzt. Hierdurch hat man die Möglichkeit zu unterscheiden, ob der Interrupt vom VIC oder von der CIA beziehungsweise von welchem Ereignis speziell die Unterbrechung ausgelöst wurde.

Hier ein Beispielprogramm, das, wenn das Sprite mit Hintergrunddaten kollidiert, die Rahmenfarbe erhöht:

```
C000 SEI        ; Interrupt verhindern
C001 LDA #$01   ; Sprite 1
C003 STA $D015  ; einschalten
C006 LDA #$80   ; Spritepointer
C008 STA $07F8  ; auf $2000 setzen
C00B LDA #$64   ; Sprite x und y
C00D STA $D000  ; Koordinaten
C010 STA $D001  ; festlegen
C013 LDA #$32   ; Interrupt
C015 STA $0314  ; auf
C018 LDA #$C0   ; $C032
```

```
C01A STA $0315  ; umlenken
C01D LDA $D01A  ; Interrupt
C020 ORA #$02   ; Enable Register auf
C022 STA $D01A  ; Spritehintergrundkollision stellen
C025 LDX #$00   ; Sprite-
C027 LDA $C046,X; daten
C02A STA $2000,X; nach
C02D INX        ; $2000
C02E BNE $C027  ; kopieren
C030 CLI        ; Interrupt wieder erlauben
C031 RTS        ; Programmende
```

Interruptroutine

```
C032 LDA $D01F  ; Kollisionsregister löschen
C035 LDA $D019  ; Latchregister laden
C038 STA $D019  ; und löschen
C03B BMI $C040  ; wenn Bit 7 gesetzt, dann Farbe erhöhen
C03D JMP $EA31  ; ansonsten zur normalen Interruptroutine
C040 INC $D020  ; Rahmenfarbe erhöhen
C043 JMP $EA31  ; Sprung zur Interruptroutine
```

Spritedaten

```
C046 00 00 00 0F FF C0 10 00
C04E 20 20 00 10 4C 01 88 4C
C056 01 88 40 00 08 40 00 08
C05E 40 00 08 46 06 08 43 0C
C066 08 41 98 08 40 F0 08 40
C06E 60 08 20 00 10 1F FF E0
C076 00 00 00 00 00 00 00 00
C07E 00 00 00 00 00 00 00 00
```

Nach dem Programmstart wird die Rahmenfarbe erhöht, falls ein Hintergrundzeichen mit dem Sprite kollidiert. Anstatt die Rahmenfarbe zu erhöhen, könnte man zum Beispiel das Sprite explodieren lassen und so weiter. Dies eignet sich gut für die

Programmierung von eigenen Spielen. Die Möglichkeiten sind praktisch unbegrenzt.

Diese Routine kann man leicht in eine Sprite-Sprite Kollisionsabfrage umändern, indem man im sogenannten Enable Register den Interrupt bei einer Sprite-Sprite Kollision freigibt. Dazu brauchen Sie nur in der Adresse $C020 in unserer Routine Bit 2 zu setzen, dann wird nur ein Interrupt ausgelöst, wenn zwei Sprites miteinander kollidieren.

Noch eine weitere Besonderheit des VIC ist der Rasterzeileninterrupt. Wenn ein Schreibzugriff auf das Register $D012 (53266) und Bit 7 der Adresse $D011 (53265) erfolgt, wird der Wert in ein internes Latchregister übertragen. Beim Lesezugriff wird in diesen Registern die momentane Zeile, in der sich gerade der Rasterstrahl befindet, angegeben. Kommt der Rasterstrahl in die angegebene Zeile, so wird im sogenannten Latchregister Bit 0 gesetzt. Ist zusätzlich noch im sogenannten Enableregister Bit 0 gesetzt, wird ein Interrupt ausgelöst.

Durch diese Möglichkeit des Interrupts kann man zum Beispiel den Textbildschirm und den Grafikbildschirm kombinieren, indem man bei der angegebenen Bildschirmzeile den Grafikbildschirm mittels Interrupt aus- beziehungsweise einschaltet, so daß in der oberen Hälfte des Bildschirms der Grafik- und in der unteren Hälfte der Textbildschirm eingeschaltet ist.

Um Ihnen diese Möglichkeit zu demonstrieren, haben wir ein Programm vorbereitet, das zwei verschiedene Bildschirmfarben gleichzeitig darstellen kann:

```
C000 SEI       ; Interrupt verhindern
C001 LDA #$1F  ; System-
C003 STA $0314 ; interrupt
C006 LDA #$C0  ; auf C01F
C008 STA $0315 ; stellen
C00B LDA #$60  ; Rasterzeile oben
C00D STA $D012 ; festlegen
C010 LDA $D011 ; High-
```

```
C013 AND #$7F  ; bit
C015 STA $D011 ; löschen
C018 LDA #$F1  ; Interrupt Enableregister
C01A STA $D01A ; auf Rasterzeileninterrupt stellen
C01D CLI       ; Interrupt wieder ermöglichen
C01E RTS       ; Programmende
```

Interruptroutine

```
C01F LDA $D019 ; Interruptregister laden
C022 STA $D019 ; und löschen
C025 BMI $C02E ; Bit 7 gesetzt?, wenn ja, dann zu Rasterz.
C027 LDA $DC0D ; ansonsten CIA Interrupt verhindern
C02A CLI       ; Interrupt ermöglichen
C02B JMP $EA31 ; zur normalen Interruptroutine
C02E LDA $D012 ; Rasterstrahl
C031 CMP #$70  ; schon unten?
C033 BCS $C045 ; wenn ja, dann Farbe auf schwarz schalten
C035 LDA #$04  ; ansonsten
C037 STA $D020 ; Farbe auf Purpur
C03A STA $D021 ; schalten
C03D LDA #$70  ; Rasterstrahl auf Zeile
C03F STA $D012 ; $70 stellen
C042 JMP $EA81 ; und abschließen
C045 LDA #$00  ; Farbe
C047 STA $D020 ; auf schwarz
C04A STA $D021 ; schalten
C04D LDA #$60  ; Rasterstrahl auf
C04F STA $D012 ; Zeile $60 legen
C052 JMP $EA81 ; und abschließen
```

Die Farben können je nach Bedarf in den Adressen $C035 und $C045 geändert werden. Die Breite des Farbstreifens kann in den Adressen $C031, $C03D und $C04D geändert werden.

Es ist vorteilhaft, wenn Sie mit dem Programm experimentieren, um so die Handhabung mit dem Rasterzeileninterrupt schneller zu erlernen.

3.12 Feinscrolling (Laufschrift)

In den Bits 0 bis 2 in der Adresse $D016 (53270) ist es möglich, den Bildschirm punktweise nach rechts oder nach links zu verschieben. Durch diesen Vorteil ist es möglich, den Zeichensatz so fließend wie ein Sprite über den Bildschirm scrollen zu lassen. Da ein Zeichen horizontal aus 8 Bits besteht, und man den Bildschirm 7 Bits nach links oder nach rechts verschieben kann, besteht die Möglichkeit, ein oder mehr Zeichen so fein zu scrollen. Das gleiche gilt auch für die Bits 0 bis 2 in der Adresse $D011 (53265), nur mit dem Unterschied, daß der Bildschirm hoch und runter geschoben werden kann.

Dies geht folgendermaßen vor sich: Der Bildschirm wird um sieben Bits nach rechts verschoben. Anschließend wird ein Zeichen gelöscht und eine Position weiter nach vorne gebracht, wobei der Bildschirm vorher wieder um 7 Bits zurückgesetzt wird. Dadurch wird ein vollkommenes Fließen des Textes erreicht.

Um Ihnen das zu demonstrieren, haben wir ein Programm entwickelt, das einen Text, den man frei angeben kann, von rechts nach links über den Bildschirm laufen läßt. Auch dieses Programm kann mit dem abgedruckten Maschinensprachemonitor eingegeben werden:

```
C000 SEI          ; Interrupt verhindern
C001 JSR $E544    ; Bildschirm löschen
C004 LDA #$55     ; Startadresse
C006 STA $FB      ; für den
C008 LDA #$C0     ; Text
C00A STA $FC      ; angeben
C00C LDY #$00     ; Bedingung für
C00E BEQ $C039    ; unbedingten Sprung
C010 LDA $D012    ; auf Rasterstrahl
C013 BNE $C010    ; warten
C015 LDX $D016    ; Verschieberegister laden
C018 DEX          ; und erniedrigen
C019 CPX #$BF     ; mit unterstem Wert vergleichen
```

```
C01B BNE $C03B   ; wenn nicht, dann zur Warteschleife
C01D LDX #$00    ; Zähler auf Null stellen.
C01F LDA $05E1,X ; Zeile um ein
C022 STA $05E0,X ; Byte verschieben
C025 INX         ; Zähler erhöhen
C026 CPX #$27    ; schon ganze Zeile
C028 BNE $C01F   ; nein, dann weiter
C02A LDA ($FB),Y ; Zeichen holen
C02C CMP #$FF    ; überprüfe auf Endmarkierung
C02E BEQ $C004   ; wenn ja, dann von vorne beginnen
C030 STA $0607   ; geholtes Zeichen auf Bildschirm bringen
C033 INC $FB     ; Zeiger auf nächstes Zeichen stellen
C035 BNE $C039   ; verzweige wenn kein Überlauf
C037 INC $FC     ; High-Byte des Zeigers erhöhen
C039 LDX #$C7    ; Verschieberegister
C03B STX $D016   ; zurücksetzen
C03E LDX #$00    ; Ver-
C040 LDY #$08    ; zö-
C042 DEX         ; ger-
C043 BNE $C042   ; ungs
C045 DEY         ; Schl-
C046 BNE $C042   ; eife
C048 LDA #$00    ; Warten auf
C04A STA $DC00   ; Taste
C04D LDA $DC01   ; wenn nicht gedrückt
C050 CMP #$FF    ; dann wieder
C052 BEQ $C010   ; zum Anfang, ansonsten
C054 RTS         ; Programmende
```

Der Text für die Laufschrift.

```
C055 2A 2A 2A 20 04 01 14 01
C05D 20 02 05 03 0B 05 12 20
C065 2A 2A 2A 20 20 20 20 20
C06D 20 20 20 20 20 20 2A 20
C075 20 20 FF 00 00 00 00 00
```

Das Programm können Sie mit irgendeiner Taste unterbrechen. Den Text können Sie selbst ab der Adresse $C055 angeben, wobei zu beachten ist, daß der Text in Bildschirmcodes angegeben ist.

Um den Programmablauf zu beschreiben, fangen wir bei der ersten Zeile unseres Programms an.

Ein SEI ist deshalb nötig, weil der Interrupt, der von der CIA ausgelöst wird, ein Zucken der scrollenden Zeichen hervorrufen würde. Um das gleiche zu vermeiden, wird noch in den Zeilen $C010 und $C013 auf den Rasterstrahl gewartet, bis dieser sich wieder in der ersten Zeile befindet. Die Zeitschleife wird benötigt, weil die Verschiebung sonst zu schnell erfolgen würde und das dazu führen würde, daß die Zeichen wiederrum zucken würden.

Da die Tastaturabfrage beim maskierten Interrupt nicht über Get erfolgen kann, muß diese direkt über den Port geschehen.

Das Laufschriftprogramm kann in eigenen Programmen eingebaut werden, um diese optisch schöner zu gestalten.

3.13 Screen-Scrolling

Das Verschieben des Bildschirminhalts in eine bestimmte Richtung nennt man SCROLLING. Der C64 kann normalerweise den Bildschirminhalt nur in eine Richtung verschieben. Dies geschieht im BASIC-Modus, wenn der Cursor über den unteren Bildschirmrand bewegt wird oder der Cursor durch den PRINT-Befehl über den unteren Bildschirmrand gedruckt wird.

Es ist manchmal aber auch notwendig, den Bildschirminhalt in eine andere Richtung zu verschieben, zum Beispiel den Hintergrund bei einem Spiel nach links oder rechts zu verschieben.

Dazu sind aber eigene Scrollroutinen notwendig, die den Bildschirminhalt sauber- und ruckfrei über den Bildschirm laufen lassen. Um dieses feine und saubere Scrolling zu erreichen,

müssen die Scrollroutinen sehr ausgetüftelt sein, weil es beim Scrollen sehr stark auf Zeit ankommt. Schon ein Taktzyklus in der Routine kann zuviel sein, und der Rasterstrahl kommt beim Verschieben ins "Gehege". Das hat zur Folge, daß es zu einem "häßlichen" Zucken des Bildschirms kommt.

Der VIC kann die Scrollroutine unterstützen, da man die Möglichkeit hat, den Bildschirminhalt innerhalb der 8 * 8-Matrix zu verschieben. Das geschieht, indem man das Register in der Adresse $D016 (53270) verändert.

Dabei sind für das Scrollen nur die Bits 0 - 2 wichtig. Wenn man hier den Wert hoch- bzw. herunterzählt, wird der Bildschirminhalt nach rechts oder links verschoben. Da ein Zeichen aber nur 8 Pixel horizontal und vertikal hat, kann man den Bildschirminhalt maximal 7 Pixel in die jeweilige Richtung verschieben. Anschließend muß man den Bildschirminhalt mittels einer eigenen Scrollroutine um 8 Pixel, also zeichenweise verschieben. Da dieses Verschieben sehr schnell geschehen muß, kann die notwendige Geschwindigkeit nur durch ein Maschinenprogramm erreicht werden.

Die Scrollroutinen funktionieren folgendermaßen:

Der Bildschirminhalt wird erst mit Hilfe der Verschieberegister in $D016 (53270) und $D011 (53265) pixelweise in die jeweilige Richtung verschoben. Dabei muß wieder auf den Rasterstrahl geachtet werden, der das Bild fünfzigmal in der Sekunde neu aufbaut. Das stellt auch kein Problem dar. Man wartet, bis der Rasterstrahl in einem unsichtbaren Bereich des Bildschirms ist und verschiebt dann ein Pixel in die erwünschte Richtung.

Nachdem man den Bildschirminhalt auf diese Weise siebenmal verschoben hat, kommt das eigentliche Problem: Die gesamten Zeichen eine Zeile müssen verschoben werden, aber die Verschiebung darf nicht länger als 1/50 Sekunde dauern.

Diese Geschwindigkeit erreicht nur eine ausgetüftelte Kopierroutine. Aber auch hier muß vorher auf den Rasterstrahl gewartet werden.

Die fertigen Scrollroutinen sind sowohl in Datazeilen als auch im Source-Code abgedruckt. Die Routinen liegen alle im Speicher ab $C000 (49152), von wo sie auch gestartet werden.

Man lädt einfach das fertige Maschinenprogramm in den Speicher oder startet den BASIC-Loader mit RUN.

Anschließend braucht man einfach nur das Maschinenprogramm mit SYS 49152 aufzurufen, und schon scrollt der Bildschiminhalt in die vorher festgelegte Richtung.

Hier sind also vier Scrollroutinen für die vier verschiedenen Scroll-Richtungen abgedruckt:

```
C000 SEI          Interrupt sperren
C001 LDX §$07     Zähler setzen
C003 LDA $D011    Verschieberegister
C006 ORA §$07     setzen
C008 STA $D011    und speichern
C00B LDY §$06     Pixel
C00D JSR $C06C    verschieben
C010 DEX          Zähler vermindern
C011 BNE $C00D    weiter, wenn
                  verschoben
C013 LDA $D011    auf
C016 BLP $C013    Rasterstrahl
C018 LDA $D011    warten
C01B BMI $C018    und
C01D LDA §$6B     die
C01F CMP $D012    Position
C022 BNE $C01D    ermitteln
C024 LDY §$28     Zähler laden
C026 LDA $03FF,Y  Zeichen holen
C029 STA $C076,Y  und speichern
C02C DEY          nächstes Zeichen
C02D BNE $C026    wenn fertig, dann
                  weiter
C02F LDA $0428,Y  Zeichen holen
C032 STA $0400,Y  und speichern
C035 INY          nächstes Zeichen
```

```
C036 BNE $C02F   fertig, dann weiter
C038 LDA $0528,Y Zeichen holen
C03B STA $0500,Y und speichern
C03E INY         nächstes Zeichen
C03F BNE $C038   fertig, dann weiter
C041 LDA $0628,Y Zeichen holen
C044 STA $0600,Y und speichern
C047 INY         nächstes Zeichen
C048 BNE $C041   fertig, dann weiter
C04A LDY §$40    Zeiger setzen
C04C LDA $06E8,Y Zeichen holen
C04F STA $06C0,Y und speichern
C052 INY         nächstes Zeichen
C053 BNE $C04C   fertig, dann weiter
C055 LDY §$27    Zeiger setzen
C057 LDA $C077,Y Zeichen holen
C05A STA $07C0,Y und speichern
C05D DEY         nächstes Zeichen
C05E BPL $C057   fertig, dann weiter
C060 LDA $D011   Verschieberegister
C063 ORA §$07    zurücksetzen
C065 STA $D011   und speichern
C068 NOP
C069 JMP $C00B   zum Start
```

Unterroutine für Pixelverschiebung

```
C06C LDA $D012   auf
C06F CMP §$FA    Rasterstrahl
C071 BNE $C06C   warten
C073 DEC $D011   verschieben
C076 RTS         Rücksprung

C000 SEI         Interrupt sperren
C001 LDA §$10    Verschieberegister
C003 STA $D011   setzen
C006 LDX §$06    Zähler setzen
C008 JSR $C061   Pixel verschieben
C00B DEX         Zähler erniedrigen
C00C BNE $C008   fertig, dann weiter
```

```
C00E LDA $D011   Bildschirm
C011 AND §$EF    aus-
C013 STA $D011   schalten
C016 LDY §$27    Zeiger setzen
C018 LDA $07C0,Y Zeichen holen
C01B STA $0100,Y und speichern
C01E DEY         nächstes Zeichen
C01F BPL $C018   fertig, dann weiter
C021 LDA $06C0,Y Zeichen holen
C024 STA $06E8,Y und speichern
C027 DEY         nächstes Zeichen
C028 BNE $C021   fertig, dann weiter
C02A DEY         Zeiger setzen
C02B LDA $05C1,Y Zeichen holen
C02E STA $05E9,Y und speichen
C031 DEY         nächstes Zeichen
C032 BNE $C02B   fertig, dann weiter
C034 DEY         Zeiger setzen
C035 LDA $04C2,Y Zeichen holen
C038 STA $04EA,Y und speichern
C03B DEY         nächstes Zeichen
C03C BNE $C035   fertig, dann weiter
C03E LDY §$C3    Zeiger setzen
C040 LDA $03FF,Y Zeichen holen
C043 STA $0427,Y und speichern
C046 DEY         nächstes Zeichen
C047 BNE $C040   fertig, dann weiter
C049 LDY §$27    Zeiger setzen
C04B LDA $0100,Y Zeichen holen
C04E STA $0400,Y und speichern
C051 DEY         nächstes Zeichen
C052 BPL $C04B   fertig, dann weiter
C054 LDA $D011   Bild-
C057 AND §$F8    schirm
C059 ORA §$10    wieder
C05B STA $D011   einschalten
C05E JMP $C000   zum Start
```

Unterroutine zur Pixelverschiebung

```
C061 LDA $D012   auf
C064 CMP §$FB    Rasterstrahl
C066 BNE $C061   warten
C068 INC $D011   verschieben
C06B RTS         Rücksprung
```

```
C000 LDA $D016    Verschieberegister
C003 AND §$F8     setzen
C005 STA $D016    und speichern
C008 SEI          Interrupt sperren
C009 LDX §$06     Anzahlverschiebung
C00B JSR $C060    Punktverschiebung
C00E DEX          Zähler erniedrigen
C00F BNE $C00B    wenn 7mal verschoben,
                  dann weiter
C011 LDA $D011    Auf
C014 BPL $C011    Rasterstrahl
C016 LDA $D011    wrten
C019 BMI $C016    und
C01B LDA §$49     Position
C01D CMP $D012    ermitteln
C020 BNE $C01d    ok, dann weiter
C022 LDX §$FF     Adressen
C024 STX $FB      für
C026 INX          Kopierroutine
C027 STX $FD      festlegen
C029 LDA §$03     und
C02B STX $FC      wieder
C02D INX          speichern
C02E STX $FE      Carryflag
C030 CLC          löschen
C031 LDX §$19     Zähler
C033 LDY §$27     festlegen
C035 LDA ($FD),Y  Zeichen holen
C037 PHA          und speichern
C038 LDA ($FB),Y  Zeichen holen
C03A STA ($FD),Y  und speichern
```

```
C03C DEY          nächstes Zeichen
C03D BNE $C038    weiter, wenn fertig
C03F PLA          Zeichen holen
C040 STA ($FD),Y  und speichern
C042 LDA $FB      Zeiger
C044 ADC §$28     für
C046 STA $FB      neue
C048 LDA $FC      Bildschirm-
C04A ADC §$00     koordinaten
C04C STA $FC      berechnen
C04E LDA $FD      und
C050 ADC §$28     wieder
C052 STA $FD      neu
C054 LDA $FE      fest-
C056 ADC §$00     legen
C058 STA $FE      Zähler
C05A DEX          vermindern
C05B BNE $C033    Wenn alles verschoben
                  dann weiter
C05D JMP $C000    zum Start
```

Unterprogramm für Pixelverschiebung

```
C060 LDA §$FB     auf
C062 CMP $D012    Rasterstrahl
C065 BNE $C062    warten
C067 INC $D016    Pixel verschieben
C06A RTS          Rücksprung


C000 LDA $D016    Verschiebe Reg.
C003 ORA §$07     laden und setzen
C005 STA $D016    und Speichern
C008 SEI          Interrupt Sperren
C009 LDX §$06     7mal verschieben
C00B JSR $C068    verschieben
C00E DEX          Zähler vermindern
C00F BNE $C00B    Wenn 7 mal verschoben,
                  dann weiter
```

```
C011 LDA $D011     Auf
C014 BPL $C011     Rasterstrahl
C016 LDA $D011     warten
C019 BMI $C016     und
C01B LDA §$3C      Position
C01D CMP $D012     ermitteln
C020 BNE $C01D     falls gefunden, dann weiter
C022 LDX §$27      Werte
C024 STX $FB       für
C026 INX           die
C027 STX $FD       Kopieroutine
C029 LDX §$03      festlegen
C02B STX $FC       und
C02D STX $FE       speichern
C02F CLC           Carry löschen
C030 LDX §$19      Zeiger
C032 LDY §$D9      holen
C034 LDA ($FB),Y   Zeichen holen
C036 PHA           zwischenspeichern
Co37 LDA ($FD),Y   Zeichen von alter
C039 STA ($FB),Y   Stelle holen und in
                   neuer speichern
C03B INY           nächstes Zeichen
C03C BNE $C037     Wenn alles verschoben
                   , dann weiter
C03E DEY           Zähler erniedrigen
C03F PLA           Alten Zähler holen
C040 STA ($FD),Y   und speichern
C042 LDA $FB       neue
C044 ADC §$28      Bild-
C046 STA $FB       koordinaten
C048 LDA $FC       berech-
C04A ADC §$00      nen
C04C STA $FC       und
C04e LDA $FD       wieder
C050 ADC §$28      neu
C052 STA $FD       setzen
C054 LDA $FE       für
C056 ADC §$00      weitere
C058 STA $FE       Zeilenverschiebung
```

```
C05a DEX          Zähler erniedrigen
C05B BNE $C032    Wenn alles verschoben
                  dann weiter
C05D LDA $D016    Verschieberegister
                  laden
C060 ORA §$07     und neu
C062 STA $D016    setzen
C065 JMP $C000    zum Start
```

Unterroutine für Pixelverschiebung

```
C068 LDA §$FB     Auf
C06A CMP $D012    Rasterstrahl
C06D BNE $C068    warten
C06F DEC $D016    Pixel verschieben
C072 RTS          Rücksprung
```

```
100 fori=1to159step15:forj=0to14:reada$:b$=right$(a$,1)
105 a=asc(a$)-48:ifa>9thena=a-7
110 b=asc(b$)-48:ifb>9thenb=b-7
120 a=a*16+b:c=(c+a)and255:poke49151+i+j,a:next:reada:ifc=athenc=
0:next:end
130 print"fehler in zeile:";peek(63)+peek(64)*256:stop
300 data 78,a2,07,ad,11,d0,09,07,8d,11,d0,a2,06,20,6c, 97
301 data c0,ca,d0,fa,ad,11,d0,10,fb,ad,11,d0,30,fb,a9, 79
302 data 6b,cd,12,d0,d0,f9,a0,28,b9,ff,03,99,76,c0,88, 189
303 data d0,f7,b9,28,04,99,00,04,c8,d0,f7,b9,28,05,99, 87
304 data 00,05,c8,d0,f7,b9,28,06,99,00,06,c8,d0,f7,a0, 73
305 data 40,b9,e8,06,99,c0,06,c8,d0,f7,a0,27,b9,77,c0, 140
306 data 99,c0,07,88,10,f7,ad,11,d0,09,07,8d,11,d0,ea, 229
307 data 4c,0b,c0,ad,12,d0,c9,fa,d0,f9,ce,11,d0,60,00, 65
308 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00, 0
309 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00, 0
310 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00, 0
```

```
100 fori=1to108step15:forj=0to14:reada$:b$=right$(a$,1)
105 a=asc(a$)-48:ifa>9thena=a-7
110 b=asc(b$)-48:ifb>9thenb=b-7
120 a=a*16+b:c=(c+a)and255:poke49151+i+j,a:next:reada:ifc=athenc=
0:next:end
130 print"fehler in zeile:";peek(63)+peek(64)*256:stop
300 data 78,a9,10,8d,11,d0,a2,06,20,61,c0,ca,d0,fa,ad, 201
301 data 11,d0,29,ef,8d,11,d0,a0,27,b9,c0,07,99,00,01, 72
302 data 88,10,f7,b9,c0,06,99,e8,06,88,d0,f7,88,b9,c1, 230
303 data 05,99,e9,05,88,d0,f7,88,b9,c2,04,99,ea,04,88, 241
304 data d0,f7,a0,c3,b9,ff,03,99,27,04,88,d0,f7,a0,27, 191
305 data b9,00,01,99,00,04,88,10,f7,ad,11,d0,29,f8,09, 158
306 data 10,8d,11,d0,4c,00,c0,ad,12,d0,c9,fb,d0,f9,ee, 148
307 data 11,d0,60,00,00,00,00,00,00,00,00,00,00,00,00, 65


100 fori=1to107step15:forj=0to14:reada$:b$=right$(a$,1)
105 a=asc(a$)-48:ifa>9thena=a-7
110 b=asc(b$)-48:ifb>9thenb=b-7
120 a=a*16+b:c=(c+a)and255:poke49151+i+j,a:next:reada:ifc=athenc=
0:next:end
130 print"fehler in zeile:";peek(63)+peek(64)*256:stop
300 data ad,16,d0,29,f8,8d,16,d0,78,a2,06,20,60,c0,ca, 81
301 data d0,fa,ad,11,d0,10,fb,ad,11,d0,30,fb,a9,49,cd, 219
302 data 12,d0,d0,fb,a2,ff,86,fb,e8,86,fd,a2,03,86,fc, 97
303 data e8,86,fe,18,a2,19,a0,27,b1,fd,48,b1,fb,91,fd, 54
304 data 88,d0,f9,68,91,fd,a5,fb,69,28,85,fb,a5,fc,69, 2
305 data 00,85,fc,a5,fd,69,28,85,fd,a5,fe,69,00,85,fe, 197
306 data ca,d0,d6,4c,00,c0,a9,fb,cd,12,d0,d0,fb,ee,16, 158
307 data d0,60,12,d0,d0,f9,ce,16,d0,60,00,00,00,00,00, 239


100 fori=1to115step15:forj=0to14:reada$:b$=right$(a$,1)
105 a=asc(a$)-48:ifa>9thena=a-7
110 b=asc(b$)-48:ifb>9thenb=b-7
120 a=a*16+b:c=(c+a)and255:poke49151+i+j,a:next:reada:ifc=athenc=
0:next:end
130 print"fehler in zeile:";peek(63)+peek(64)*256:stop
300 data ad,16,d0,09,07,8d,16,d0,78,a2,06,20,68,c0,ca, 72
301 data d0,fa,ad,11,d0,10,fb,ad,11,d0,30,fb,a9,3c,cd, 206
```

```
302 data 12,d0,d0,fb,a2,27,86,fb,e8,86,fd,a2,03,86,fc, 137
303 data 86,fe,18,a2,19,a0,d9,b1,fb,48,b1,fd,91,fb,c8, 198
304 data d0,f9,88,68,91,fd,a5,fb,69,28,85,fb,a5,fc,69, 2
305 data 00,85,fc,a5,fd,69,28,85,fd,a5,fe,69,00,85,fe, 197
306 data ca,d0,d5,ad,16,d0,09,07,8d,16,d0,4c,00,c0,a9, 58
307 data fb,cd,12,d0,d0,f9,ce,16,d0,60,00,00,00,00,00, 135
```

Noch ein kleiner Hinweis zu der Scrollroutine, die nach unten scrollt. Hier kam es zu zeitlichen Problemen, da hier der Bildschirminhalt von unten nach oben geändert werden muß. Es ist also nicht mehr möglich, die Bildänderung so zu steuern, daß diese nicht vom Rasterstrahl überholt wird. Deshalb wird hier der Bildschirm beim Kopieren kurzzeitig ausgeschaltet. Dadurch ist zwar kein Rucken mehr vorhanden, aber dafür blinkt der Hintergrung bei jeder Verschiebung kurz auf. Das Blinken stört am wenigsten, wenn die Rahmenfarbe identisch mit der Hintergrundfarbe ist.

3.14 Registerbeschreibung des VIC

Der VIC verfügt über 47 Register, die im folgenden beschrieben werden:

REG 0 — Sprite-Register 0 X-Koordinate
Hier sind 8 Bits der Bildschirmkoordinate X enthalten, auf der das Sprite dargestellt wird.
Bit 9 befindet sich in REG 16.

REG 1 — Sprite-Register 0 Y-Koordinate
Wie oben, jedoch für die Y-Richtung. Dieses Register hat im Gegensatz zu REG 0 keinen Übertrag.

Die Register 2 bis 15 folgen alle dem oben beschriebenen Aufbau. Registerpaar 2/3 ist für Sprite 1, Registerpaar 4/5 für Sprite 2 zuständig und so weiter.

REG 16 MSB der X-Koordinaten
Hier befinden sich die Überläufe aus dem Sprite-X-Register, und zwar Bit 0 für Sprite 0, Bit 1 für Sprite 1 und so weiter.

REG 17 Steuerregister 1
Bit 0 bis 2 Offset der Darstellung vom oberen Bildrand in Rasterzeilen.
Bit 3 0=24 Zeilen, 1=25 Zeilen
Bit 4 0=Bildschirm aus
Bit 5 1=Standart Bitmap Mode
Bit 6 1=Extended Background Color Mode
Bit 7 Übertrag aus REG 18

REG 18 Hier wird die Nummer der Rasterzeile angegeben, bei deren Strahldurchlauf ein IRQ ausgelöst werden kann. Übertrag dieses Registers in REG 17.

REG 19 X- Anteil der Bildschirmposition, an der sich der Strahl gerade befand, als ein Strobe ausgelöst wurde.

REG 20 Wie oben, jedoch Y- Anteil.

REG 21 Sprite Enable
Jedem Sprite ist ein Bit zugeordnet. 1=Sprite an, 0=Sprite aus.

REG 22 Steuerregister 2
Bit 0-2 Offset der Darstellung vom linken Rand in Rasterpunkten.
Bit 3 0=38, 1=40 Zeichen.
Bit 4 1=Multicolor Mode.

REG 23 Sprite Expand X
Jedem Sprite ist ein Bit zugeordnet. 1=Sprite wird doppelt so Breit.

REG 24 Basisadresse von Zeichengenerator und Video-RAM.
Bit 1-3 Adressbits 11-13 für die Zeichenbasis.
Bit 4-7 Adressbits 10-13 für die Basis der Video-RAM

REG 25 IRR Interrupt Request Register
Bit 0 Auslöser ist REG 18
Bit 1 Auslöser ist REG 31
Bit 2 Auslöser ist REG 30
Bit 3 Auslöser ist LP
Bit 7 =1 wenn mindestens eins der anderen Bits =1 ist.

REG 26 IMR Interrupt Mask Register
Belegung wie oben. Bei Übereinstimmung mindestens eines Bits aus IRR und IMR wird der Pin IRQ=0

REG 27 Jedem Sprite ist ein Bit zugeordnet. 1= Hintergrundzeichen hat vor dem Sprite Priorität.

REG 28 Jedem Sprite ist ein Bit zugeordnet. 1= Spritemulticolor Mode.

REG 29 Sprite Expand Y
Jedem Sprite ist ein Bit zugeordnet. 1= Sprite wird doppelt so hoch.

REG 30 Sprite- Sprite Kollision
Jedem Sprite ist ein Bit zugeordnet. Berührt ein Sprite ein anderes, si werden die entsprechenden Bits=1. Gleichzeitig wird IRR Bit 2 =1. Nach dem Ereignis muß dieses Register gelöscht werden, da sich die Bits nicht selbstständig zurücksetzen.

REG 31 Sprite- Background Kollision.
Wie oben, jedoch tritt das Ereignis ein, wenn ein Sprite Berührung mit einem Hintergrundzeichen hat.

REG 32 Exterior Color (Rahmenfarbe)

REG 33-36 Backgroundcolor 0-3 (Hintergrundfarben)

REG 37-38 Sprite Multicolor 0-1

REG 39-46 Color Sprite 0- Sprite 7

3.15 Pinbeschreibung des VIC 6567

1-7	D6-D0	Prozessordatenbus
8	-IRQ	0 wenn ein Bit des IMR und des IRR übereinstimmen
9	-LP	Eingang, Lightpenstrobe
10	-CS	Prozessorbusaktionen finden nur bei CS=0 statt.
11	R/-W	;0=Übername der Daten vom Bus.
12	BA	0 wenn Daten bei einem Lesezugriff
13	VDD	+12V
14	COLOR	Farbinformationen Ausgang
15	SYNC	Zeilen und Bildsynchonisationsimpulse
16	AEC	0=VIC benutzt Systembus, 1=Bus frei
17	0OUT	Ausgang Systemtakt
18	-RAS	dyn. RAM Seuersignal
19	-CAS	wie oben
20	GND	
21	0color	Eingang Farbfrequenz
22	0IN	Eingang Dotfrequenz
23	ALL	Prozessoradressbus
24-29	A0/A8	
	A5/A13	gemultiplexter (Video-) RAM-Adressbus
30-31	A6-A7	(Video-) RAM-Adressbus
32-34	A8-A10	Prozessoradressbus
35-38	D11-8	Daten aus Farb-RAM
39	D7	Prozessordatenbus
40	VCC	+5V

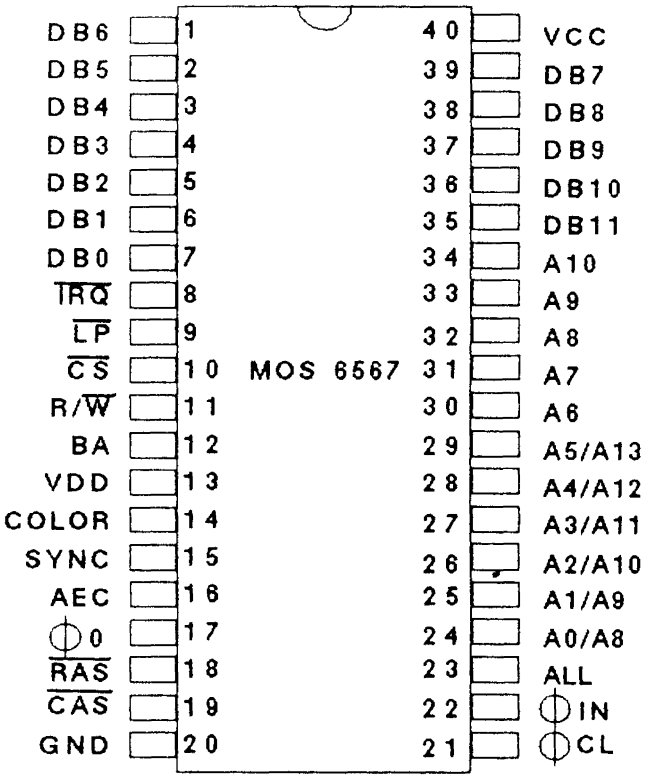


Abb. 3.14: Der 6567

4. Der Soundcontroller

Der C64 verfügt über einen hochwertigen Soundbaustein, der SID (Sound Interface Device) genannt wird. Mit ihm lassen sich nahezu jegliche Geräusche darstellen, mit denen Sie beispielsweise Ihre Programme untermalen können. Hier nun eine kurze Beschreibung seiner Fähigkeiten:

Die Basisadresse des SID ist bei $D400 (54272). Er verfügt über 3 getrennte Tongeneratoren mit einem Frequenzbereich von 0 bis ca. 4 kHz. Die Wellenform sowie auch die Hüllkurve jedes Tongenerators können getrennt gewählt werden. Bei der Wellenform hat man die Wahl zwischen Dreieck, Sägezahn, Rechteck und Rauschen. Für jeden Generator stehen sieben Register zur Verfügung, die bei jedem Tongenerator die gleiche Bedeutung haben. All diese Register, bis auf die des Tongenerators 3, der eine Sonderstellung einnimmt, sind "write only". Zusätzlich steht noch ein Filter zur Verfügung, mit dem es möglich ist, sowohl die Tongeneratoren als auch eine externe Signalquelle zu verändern. Mit dem dritten Tongenerator ist es unter anderem auch möglich, Zufallszahlen zu erzeugen.

Bevor Sie sich die folgende Beschreibung des SID durchlesen, und Sie zu denjenigen gehören, die sich mit dem Hexadezimalsystem noch nicht ganz auskennen, ist zu empfehlen, das erste Kapitel dieses Buches vorher durchzulesen.

4.1 Die Frequenz

Jeder der 3 Tongeneratoren verfügt über sieben Register. Die ersten beiden geben die Frequenz des Tons an. Sie wird als 16-Bit-Zahl gespeichert. Wenn man die Frequenz, die in Herz angegeben ist, ins Hexadezimalsystem umrechnet und in die beiden Register schreibt, wird man leider nicht den gewünschten Ton zu hören bekommen. Zuvor muß sie in eine an den Computer angepaßte Zahl umgerechnet werden. Diese Zahl hängt von der Taktfrequenz des Computers ab, die bei dem amerikanischen Model höher ist als bei dem hiesigen.

Die Taktfrequenz läßt sich errechnen, indem man die Anzahl der Schwingungen des Quarzes duch 18 bez. 14 teilt.

Europäische Version:	17734472 Hz / 18 = 985.2484 kHz
Amerikanische Version:	14318180 Hz / 14 = 1022.7271 kHz

Der vorerwähnte 16-Bit-Wert setzt sich wie folgt zusammen:

```
WERT = FREQUENZ * 2^24 / TAKTFREQUENZ
```

Beide Frequenzangaben müssen in Herz erfolgen. Der so errechnete Wert muß jetzt lediglich in LOW- und HIGH-Byte aufgespalten weden:

```
HI = INT (WERT/256)
LO = WERT - 256*HI
```

Durch Änderung des Wertes um eins, erreicht man eine Tonfrequenzänderung von ca. 0.06 Hz.

Doch wie kann man anhand der Tonfrequenz den entsprechenden Ton der Tonleiter erhalten? Dafür gibt es zwei Möglichkeiten: 1. Nachschlagen in der Tabelle am Ende des Kapitels oder 2. Errechnen mit einer gleich gezeigten Formel. Das Errechnen der entsprechenden Note hat besonderen Vorteil beim Spielen von Tonleitern, da nicht jeder Ton gespeichert werden muß, sondern errechnet werden kann.

Die Frequenz des gewünschten Tons läßt sich wie folgt berechnen:

```
FREQUENZ = 2^(NR/12)*16.35
```

NR ist die Nummer der entsprechenden Note, und 16.35 ist der Ton mit der niedrigsten Frequenz der in der Tabelle aufgeführten Noten. Dieser Wert kann durch einen beliebigen Frequenzwert aus der Tabelle ersetzt werden. Durch Veränderung von NR um eins, verändert sich die Frequenz in Halbtonschritten.

4.2 Wellenform

Man kann zwischen den vier verschiedenen Wellenformen Dreieck, Sägezahn, Rechteck und Rauschen wählen. Dafür sind die obersten 4 Bits des fünften Registers jedes Tongenerators zuständig. Es ist nicht möglich, mehrere Wellenformen zu mischen. Auf die anderen Funktionen dieses Registers werden wir später noch eingehen.

Durch Setzen eines der Bits wird die entsprechende Wellenform ausgewält.

Bit 4: Dreieckwelle
Bit 5: Sägezahnwelle
Bit 6: Rechteckwelle
Bit 7: Rauschen

Wird die Rechteckwelle gewählt, so ist es zusätzlich möglich, die Pulsbreite zu variieren. Zur Festlegung der Pulsbreite dienen die Register zwei und drei des jeweiligen Tongenerators. Von dem sich aus den zwei Registern ergebenden 16-Bit-Wert werden jedoch nur die untersten 12 Bits genutzt. Hieraus ergibt sich, daß der höchste Wert, der auf die Pulsbreite Einfluß hat, $0FFF (4095) ist. Extrem kleine oder extrem große Werte erzeugen keinen Ton. Wenn man in diese Register den Wert 2047, der der Hälfte des Maximums entspricht, schreibt, erhält man eine vollkommen regelmäßige Rechteckschwingung.

Die Dreieckwelle entspricht in etwa einer Sinusschwingung und eignet sich zum Imitieren sämtlicher Zupfinstrumente.

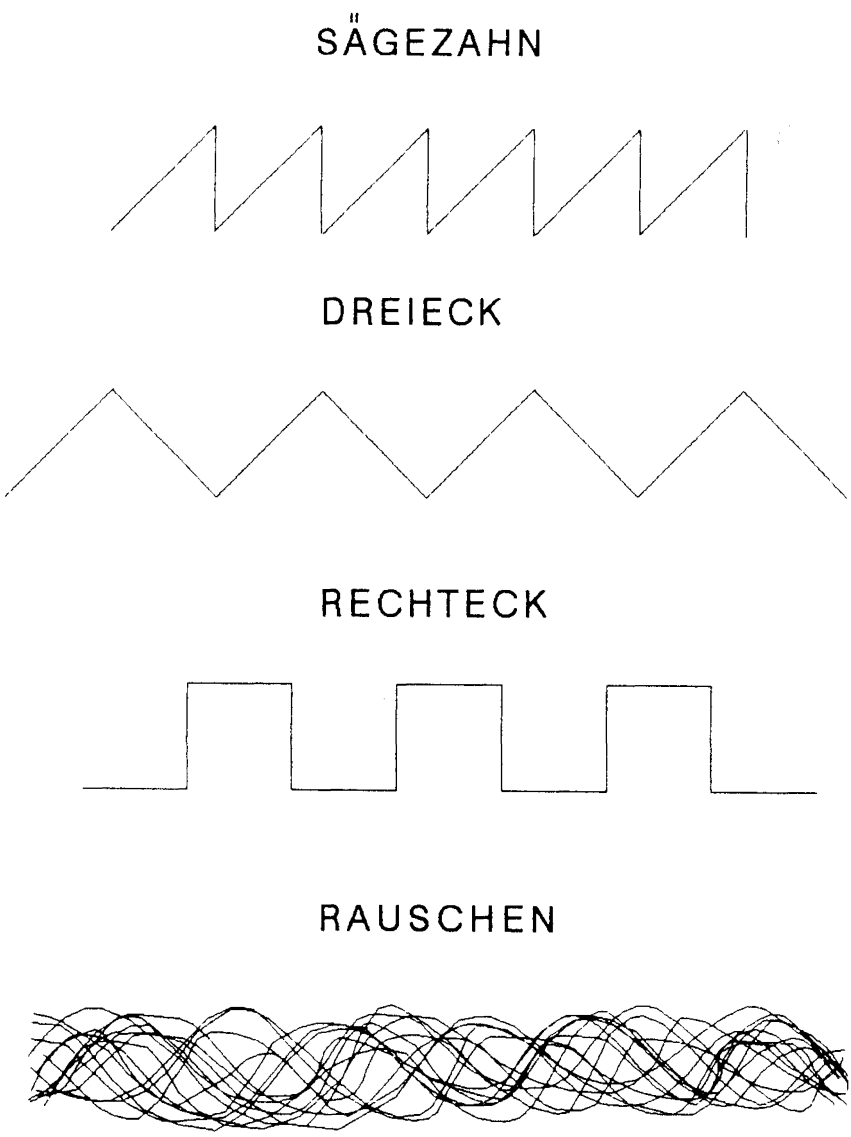


Abb. 4.2.1: Wellenformen

4.3 Hüllkurve

Mit Hüllkurve ist nichts anderes als der Lautstärkeverlauf des zu erklingenden Tones gemeint. Durch ihre Veränderung ist es möglich, den Ton noch weiter zu beeinflussen. Der Lautstärkeverlauf ist in vier verschiedene Abschnitte unterteilt. Wir unterscheiden zwischen:

ATTACK

Das Spielen eines Tones beginnt mit der Attack-Phase. Es ist jedoch nötig, dem SID mitzuteilen, wann er beginnen soll, den Ton zu spielen. Hierfür existiert Bit 0 des Registers 3 des jeweiligen Tongenerators. Sobald das sogenannte KEY-Bit gesetzt wird, fängt der Ton an zu spielen und beginnt mit der Attack-Phase. In dieser Phase steigt die Lautstärke von Null auf die im LOW-Nibbel des Registers 24 angegebene Gesamtlautstärke. Die Zeit, in der dies geschieht, wird im HIGH-Nibbel des Registers 5 der jeweiligen Stimme angegeben. Die Zeit kann zwischen 0.002 sec und 8 sec liegen. Hohe Werte ergeben eine lange Attack-Phase. Die Attack-Phase wird immer eingeleitet, auch wenn der Ton noch nicht am Ende der weiter unten besprochenden Release-Phase war.

DECAY

Nach Abschluß der Attack-Phase kommt die Decay-Phase. In ihr ändert sich die Lautstärke von der eben erreichten bis auf die im HIGH-Nibbel der Registers 6 eingestellte. Die dafür benötigte Zeit wird im LOW-Nibbel des Registers 5 angegeben. Sie ist einstellbar zwischen 0.006 sec und 24 sec.

SUSTAIN-Spanne

Nachdem die neue Lautstärke erreicht wurde, bleibt sie konstant. Ohne unser Zutun würde der Ton ewig mit der gleichen Lautstärke spielen. Es wird erst die nächste Phase erreicht, sobald das KEY-Bit, das wir zu Beginn der Attack-Phase gesetzt hatten, wieder gelöscht wird. Die Länge der Sustain-Spanne ist

nämlich in keinem Register einstellbar. Sie muß durch Ablaufen einer Zeitschleife festgelegt werden. Sobald das KEY-Bit gelöscht wird, wird umgehend die Release-Phase erreicht, selbst dann, wenn der Ton sich erst in der Attack-Phase befand.

RELEASE

In dieser Phase fällt die Lautstärke von der bei Decay angegebenen auf Null zurück. Die dafür benötigte Zeit wird im LOW-Nibbel des Registers 6 angegeben. Sie ist wie bei Decay zwischen 0.006 sec und 24 sec wählbar.

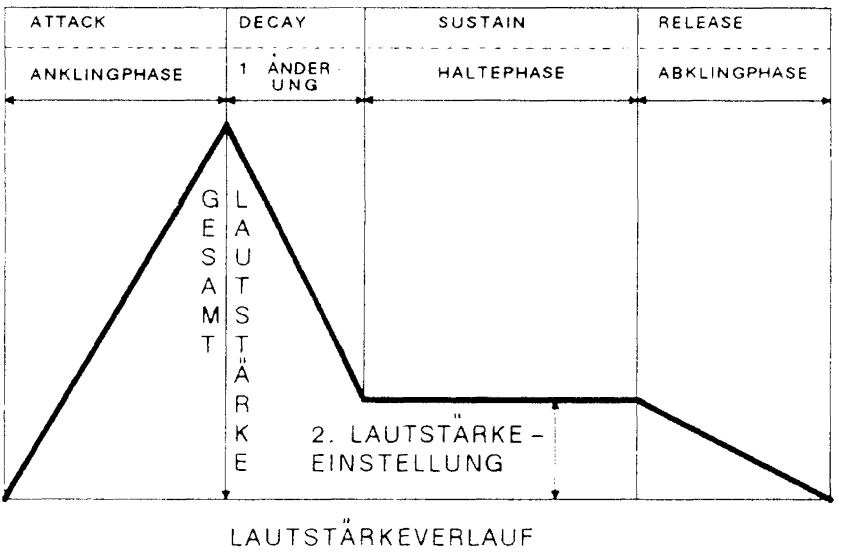


Abb. 4.3.1: **Lautstärkediagramm**

Nachfolgend eine Tabelle, die Ihnen das Wählen der gewünschten Zeiten ermöglicht:

WERT		ATTACK	RELEASE DECAY
DEZ	HEX	DEZ	DEZ
0	00	2 ms	8 ms
1	01	8 ms	24 ms
2	02	16 ms	48 ms
3	03	24 ms	72 ms
4	04	38 ms	114 ms
5	05	56 ms	168 ms
6	06	68 ms	204 ms
7	07	80 ms	240 ms
8	08	100 ms	300 ms
9	09	250 ms	750 ms
10	0A	500 ms	1.5 s
11	0B	800 ms	2.4 s
12	0C	1 s	3 s
13	0D	3 s	9 s
14	0E	5 s	15 s
15	0F	8 s	24 s

Abb. 4.3.2: Zeittabelle

Bevor wir uns anhand einiger Beispiele mit den Grundkenntnissen der Tonprogrammierug vertraut machen, ist noch einiges, was unbedingt beachtet werden sollte, zu erklären.

Als erstes müssen Sie eine Grundlautstärke, die für alle drei Stimmen gilt, wählen. Sie liegt im Bereich von 00 (nicht hörbar)

bis 15 (recht laut). Dieser Wert muß in den LOW-Nibbel des Registers 24 geschrieben werden. Daraufhin legen Sie alle weiteren Parameter wie Frequenz, Hüllkurve und Wellenform fest. Nachdem dies geschehen ist, müssen Sie dem SID mitteilen, daß er den eben gewählten Ton zu spielen beginnen soll. Dies erreichen Sie, indem sie Bit 0 des Registers 4 setzen (KEY-Bit). Dadurch schaltet der SID die Attack-Phase (Anklingphase) ein, und es erklingt ein Ton, der nach der in Register 5 angegebenen Zeit die in Register 24 angegebene Gesamtlautstärke erreicht. Da dieses Bit im selben Register zu finden ist, das auch für die Wellenform zuständig ist, wird beides - am Ende der Parametereinstellung - zugleich erledigt.

Doch wollen wir uns jetzt von der unerträglich geräuschlosen Theorie entfernen und in die Praxis übergehen.

```
10 SID=54272
20 POKE SID+24,15
30 POKE SID+5,194
40 POKE SID+6,90
50 POKE SID,180
60 POKE SID+1,8
70 POKE SID+4,33
80 FOR N=1 TO 1400:NEXT:POKE SID+4,32
```

Dieses kleine BASIC-Programm wird Sie bestimmt nicht vom Hocker reißen, aber es ist schon ein Anfang, der für die weiteren Erprobungen vollkommen ausreicht. Als erstes wird der Variablen SID die Basisadresse des SID zugewiesen. Daraufhin wird die gesamte Lautstärke auf den Wert 15, das heißt auf das Maximum, eingestellt. Als nächstes werden die Zeiten für die Attack- und für die darauffolgende Decay-Phase des Tons festgelegt. Der Wert 194 (%11000010) spaltet sich demnach wie folgt auf: Das LOW-Nibbel erhält den Wert 2 und das HIGH-Nibbel den Wert 12, was bedeutet, daß das Anklingen des Tons langsam und die darauffolgende Änderung sehr abrupt geschieht. Als nächstes kommt Zeile 40, in der die neu zu erreichende Lautstärke auf 5 und die Release-Zeit auf 10 gestellt wird. In

Zeile 50 und 60 wird die Frequenz des Tones festgelegt. Diese Werte entsprechen, wie aus der Tabelle ersichtlich, dem 36. Ton und somit dem C-3.

Danach wird die Sägezahnwelle eingestellt und KEY-Bit (Bit 0 des Registers 4) gesetzt, damit der SID anfängt, den Ton zu spielen. Die Warteschleife dient zur Überbrückung der Zeit, die der Ton aufgrund der angegebenen Parameter braucht, um die Gesamtlautstärke zu erreichen und daraufhin auf die Lautstärke, die in Register 6 angegeben ist, abzufallen.

Zum guten Schluß wird das KEY-Bit wieder gelöscht um in die Release-Phase einzutreten. Es muß darauf geachtet werden, daß die Angabe der Wellenform bei diesem Vorgang nicht gelöscht wird, da dies sonst einen abrupten Abbruch des Tons zu Folge hätte.

Um diese Vorgänge nachvollziehen zu können, sollten Sie immer mit der Registertabelle arbeiten.

Wer einen Ton mit einem der beiden anderen Tongeneratoren erzeugen will, muß lediglich die Basisadresse um sieben erhöhen.

Das folgende Programm spielt eine Tonleiter in Halbtonschritten. Die entsprechenden Frequenzen werden, wie bereits erläutert, errechnet. Zur Demonstration wird hierzu der Tongenerator 3 verwendet.

```
100 SID=54272:B=SID+2*7
115 Q=2^24/(17734472/18)
120 POKE B+5,8:POKE B+6,215:POKE SID+24,15
130 FOR I=12 TO 94:N=INT(2^(I/12)*16.35*Q+.5)
140 H=INT(N/256):POKE B,N-256*H:POKE B+1,H
150 POKE B+4,17:FOR W=1 TO 200:NEXT:POKE B+4,16:NEXT
```

Als erstes wird die Basisadresse auf die Anfangsadresse des Tongenerators 3 gestellt. Daraufhin wird der Faktor, mit dem die errechnete Frequenz multipliziert werden soll, ermittelt. In Zeile

120 werden Hüllkurve und Gesamtlautstärke festgelegt. Dann werden die für die jeweilige Tonnummer entsprechende Frequenz und gleichzeitig auch der Wert, der vom SID verarbeitet wird, errechnet. In Zeile 140 wird dieser Wert in LOW- und HIGH-Byte zerlegt und in die entsprechenden Register geschrieben. Als letztes wird die Wellenform festgelegt und gleichzeitig das KEY-Bit gesetzt, eine Warteschleife durchlaufen und das Abklingen des Tons veranlaßt. Alsdann wird der nächste Ton gespielt.

An dieser Stelle wollen wir noch auf zwei Möglichkeiten der Tonbeeinflussung hinweisen, die im Zusammenhang mit Register 4 des jeweiligen Tongenerators stehen. Es sind Bit 1 und Bit 2.

Bit 1:

Durch Setzen dieses Bits synchronisieren Sie die Grundfrequenzen zweier Tongeneratoren. Der Tongenerator, nach dem synchronisiert wird, muß, um ein Ergebnis zu erzielen, auf eine von 0 verschiedene Frequenz gesetzt werden. Alle anderen Register dieses Tongenerators haben keine Funktion mehr.

Bit 2:

Dieses Bit hat nur eine Bedeutung, falls die Dreieckwelle gewählt wurde. Ist dies der Fall, so wird das Tonsignals durch Ringmodulation (Summe und Differenz der beiden Grundstimmen), d.h. durch ein kombiniertes Signal der beiden entsprechenden Tongeneratoren ersetzt. Es sind wiederum nur die Register für die Frequenz beim zweiten Tongenerator maßgebend.

4.4 Filter

Der SID verfügt, wie bereits erwähnt, über einen Filter, mit dem es ermöglicht wird, die Tongeneratoren und eine externe Quelle zu verändern. Das externe Signal wird über die AUDIO IN Leitung an den Computer übergeben.

Die Filterfrequenz ist in den Registern 21 und 22 festgelegt. Von den 16 Bits der beiden Register werden lediglich 11 benutzt. Von Register 21 werden nur die Bits 0 bis 2 benutzt. Die restlichen Bits sind unbenutzt. Der Wert, der in die beiden Register geschrieben wird, muß erst aus der Filterfrequenz erechnet werden. Es wird wie folgt verfahren.

WERT = (F-30)/5.8182 Hz

F ist die Frequenz, die in Herz angegeben wird.

Nachdem die Frequenz angegeben wurde, muß festgelegt werden, welcher Teil der Tonfrequenz gefiltert werden soll. Dies kann man mit den Bits 4 bis 6 des Registers 24 einstellen:

Bit 4 LOWPASS:

Ist dieses Bit gesetzt, so werden alle Frequenzkompünenten über der eingestellten Filterfrequenz mit 12 dB je Oktave abgeschwächt.

Bit 5 BANDPASS:

Ist das Bit gesetzt, so werden alle Frequenzen ungleich der eingestellten Filterfrequenz mit 6 dB je Oktave vermindert.

Bit 6 HIGHPASS:

Dieses Bit entspricht Bit 4 des Registers, nur daß alle Frequenzen unterhalb der Filterfequenz mit 12 dB je Oktave verringert werden.

Es ist möglich, die verschiedenen Filterarten zu kombinieren. Durch eine Kombination von LOW- und HIGHPASS kann die Tonfrequenz zwischen den Werten eingeschachtelt werden (Bandsperre).

Als letztes noch die Veränderung der Filterresonanz. Sie wird im Register 23 festgelegt und kann einen Wert von 0 bis 15 annehmen, wobei ein hoher Wert eine hohe Resonanz hervorruft und

ein niedriger eine niedrige Resonanz. Somit erreicht man eine Betonung der Frequenzkomponenten. Mit der Veränderung der Filterresonanz wärend des Spielens des Tons lassen sich nahezu alle Musikinstrumente nachahmen.

4.5 Tongenerator 3

Der Tongenerator 3 nimmt eine Sonderstellung im Vergleich zu den übrigen Tongeneratoren ein. Seine Register sind als einzige lesbar und daher sind bei ihm der Verlauf der Hüllkurve und der Zustand des Oszillators feststellbar.

Hüllkurve:

Die Hüllkurve des Tongenerators 3 kann im Register 28 ($1C) gelesen werden. Ihr Wert gibt die momentane Lautstärke des Tons an. Der Wert steigt bis auf 255 ($FF) an, wenn die im Register 24 ($18) gesetzte Gesamtlautstärke erreicht wird. Danach fällt der Wert wieder und erreicht nach Beendigung der RELEASE-Phase den Wert 0. Diesen Umstand kann man sich zu Nutze machen, wenn man beispielsweise eine fest eingestellte SUSTAIN-Spanne unabhängig von der Länge der ATTACK-Phase haben will. Man liest das Hüllkurve-Register aus und wartet, bis er den Wert 255 enthält. Von diesem Punkt an läßt man eine Zeitschleife ablaufen, um nach ihrem Ablauf das KEY-Bit zu löschen. Mit Hilfe dieses Registers kann auch erkannt werden wann die RELEASE Zeit verstrichen ist. Weiterhinn besteht die Möglichkeit, in Abhängigkeit des Zustands der Hüllkurve die Frequenz oder die Pulsbreite während des Tonsignals zu verändern und somit eine Reihe von wirkungsvollen Efekten zu erzielen.

Oszillator:

Dies ist das Register 27 ($1B), mit welchem es möglich ist, die 8 höchstwertigen Bits des Oszillators 3 abzufragen. Je nach Wahl der Wellenform ändern sich die Werte dieses Registers im Verlauf des Tonsignals. Wurde die Dreieckwelle gewählt, so nimmt der Inhalt des Registers gleichmäßig von 0 bis 255 zu und

gleichmäßig wieder ab. Bei der Wahl der Sägezahnwelle nimmt der Wert ebenfalls gleichmäßig zu, fällt jedoch nach dem Erreichen von 255 sofort auf 0 zurück und beginnt alsdann wieder erneut zu steigen. Bei der Rechteckwelle pendelt der Wert ständig zwischen 0 und 255. Die Länge der beiden Phasen hängt von der Wahl der Pulsbreite ab. Wird Rauschen gewählt, dann nimmt das Register zufällige Werte an. Es kann somit verwendet werden, um Zufallszahlen zu erzeugen.

Alle Funktionen des Tongenerotors 3 können auch verwendet werden, wenn dieser mit Hilfe des Bits 7 des Registers 24 stumm geschaltet ist.

4.6 Der Analog/Digitalwandler

Ein A/D-Wandler ist eine Einrichtung zur Umwandlung eines analogen Signals (z.B. Spannung) in einen digitalen Wert. Die prinzipielle Schwierigkeit bei einer solchen Umwandlung besteht darin, einen analogen Wert mit unendlich feiner Abstufung in einen digitalen Wert mit endlicher Abstufung (feste Intervalle) umzuformen. Dabei entsteht zwangsläufig ein +/- Fehler, dessen Höchstwert dem kleinsten digitalen Schritt gleicht.

Der SID 6581 enthält zwei A/D-Wandler. Hierbei handelt es sich um eine Anordnung mit einer intern erzeugten Referenzspannung von ca. 2,5V. Der Meßvorgang besteht darin, daß eine externe Kapazität zunächst entladen und anschließend ein Wert in Register 25 bzw. 26 übernommen wird, der der benötigten Zeit für eine erneute Aufladung der Kapazität auf die Referenzspannung entspricht. Dieser Vorgang wiederholt sich zyklisch.

4.6.1 Die Handhabung des A/D-Wandlers

Aus dem oben Gesagten ergibt sich, daß nur eine potentiometrische Beschaltung des Wandlers in Frage kommt. Als Meßwertaufnehmer eignen sich demnach nur veränderliche Widerstände

in irgendeiner Form, z.B. Photowiderstände, Heißleiter, Kaltleiter usw.

Sollen Spannungen gemessen werden, so müssen diese zuvor in eine geeignete Form umgewandelt werden, z.B. mit Hilfe eines Unijunction-Transistors.

Die Meßanordnung sieht einfach so aus, daß an das eine Ende des Meßwiderstandes +5V angelegt werden (an den Controlports des C64 verfügbar) und das andere Ende mit dem Analogeingang des SID (ebenfalls an den Controlports verfügbar, Bezeichnung POTX und POTY) verbunden wird. Der aus den Registern 25 und 26 ausgelesene Wert ist ein Maß für den Widerstand.

Um die ganze Skala von 8 Bits ausnutzen zu können, muß sich der Widerstand im Bereich von 200 Ohm (nicht kleiner!!!) bis 200 Kiloohm bewegen.

Die programmtechnische Anwendung finden Sie bei der Verwendung der Paddles, wie sie im Kapitel 5.4.3 beschrieben ist.

4.7 Registerbeschreibung des SID Basisaddresse $D400 (54272)

REG $00 Oszillatorfrequenz niederwertiges Byte für Stimme 1

REG $01 Oszillatorfrequenz höherwertiges Byte für Stimme 1

REG $02 Pulsbreite niederwertiges Byte für Stimme 1

REG $03 Pulsbreite höherwertiges Byte für Stimme 1
Die Register 2 und 3 bestimmen das Puls-Pauseverhältnis des Rechteckausgangs von Stimme 1. Von Register 3 werden nur die Bits 0-3 benutzt.

REG $04 Steuerregister der Stimme 1

Bit 0 Key: Steuerbit für den Ablauf des Hüllkurvengenerators. Beim Übergang von 0 nach 1 steigt die Lautstärke von Stimme 1 innerhalb der in REG 5 programmierten ATTACK-Zeit von Null auf den Maximalpegel. Beim Übergang von 1 auf 0 geht die Lautstärke innerhalb der in REG 6 programmierten RELEASE-Zeit auf Null zurück.

Bit 1: 1=Oszillator 1 wird mit Oszillator 3 synchronisiert. Dieses Bit hat auch Wirkung, wenn die Stimme 3 stummgeschaltet ist.

Bit 2: 1=Dreieckschwingungsausgang von Oszillator 1 wird durch ein Frequenzgemisch (Summe und Differenz) der Frequenzen von Oszillator 1 und 3 ersetzt. Dieser Effekt tritt auch dann ein, wenn Stimme 3 stummgeschaltet ist.

Bit 3: Wenn zusammen mit dem Rauschgenerator noch eine weitere Schwingungsform desselben Oszillators ausgewählt wurde, kann es vorkommen, daß der Rauschgenerator blockiert. Die Blockade kann durch Setzen dieses Bits wieder aufgehoben werden.

Bit 4 tri: 1=Dreieckschwingung ausgewählt.

Bit 5 saw: 1=Sägezahnschwingung ausgewählt.

Bit 6 pul: 1=Rechteckschwingung ausgewählt. Das Puls-Pauseverhältnis dieser Schwingung wird in REG 2 und REG 3 eingestellt.

Bit 7 nse: 1=Rauschgenerator ausgewählt. Anmerkung zu den Bits 4-7: Es ist praktisch möglich, mehrere Schwingungsformen gleichzeitig auszuwählen. Zu beachten ist jedoch, außer dem zu Bit 3 Gesagten, daß das Ergebnis nicht etwa die Summe aller Formen darstellt, sondern vielmehr eine logische UND-Verknüpfung der Komponenten ist.

REG 5 ATTACK/DECAY

Bit 0-3 Diese Bits bestimmen die Zeit, in der die Lautstärke vom Maximum auf den Sustainpegel abfällt. Der einstellbare Bereich beträgt 6 msec bis 24 sec.

Bit 4-7 Hiermit wird die Zeit festgelegt, in der die Lautstärke nach Setzen des KEY-Bits von Null auf das Maximum ansteigt. Der einstellbare Bereich beträgt 2 msec bis 8 sec.

REG 6 SUSTAIN/RELEASE

Bit 0-3 Mit diesen Bits wird die Zeit eingestellt, innerhalb der die Lautstärke nach Rücksetzen des KEY-Bits vom Sustainpegel auf Null abfällt. Der einstellbare Bereich beträgt 6 msec bis 24 sec.

Bit 4-7 Diese Bits geben den Sustainpegel an, d.h. die Lautstärke, mit der der Ton nach Ablauf der Decayzeit andauert.

REG 7 Diese Register steuern die Stimme 2 und 3 analog zu den Registern 0-6 mit folgenden Ausnahmen:

REG 13 SYNC synchronisiert Oszillator 2 mit Oszillator 1. RING ersetzt den Dreieckausgang von Oszillator 2 mit den ringmodulierten Frequenzen der Oszillatoren 2 und 1.

REG 14 Diese Register steuern die Stimme 3 analog zu den Registern 0-6 mit folgenden Ausnahmen:

REG 20 SYNC synchronisiert Oszillator 3 mit Oszillator 2. RING ersetzt den Dreieckausgang von Oszillator 3 mit dem Frequenzgemisch aus den Oszillatoren 3 und 2.

REG 21 Filterfrequenz niederwertiges Byte. Es werden nur die Bits 0-2 benutzt.

REG 22 Filterfrequenz höherwertiges Byte
Die 11-Bit-Zahl der Register 21 und 22 bestimmt die Filtereckfrequenz, bzw. -mittenfrequenz.

REG 23 Filterresonanz und -schalter

Bit 0 1=Stimme 1 wird über den Filter geleitet.
Bit 1 1=Stimme 2 wird über den Filter geleitet.
Bit 2 1=Stimme 3 wird über den Filter geleitet.
Bit 3 1=Die externe Signalquelle wird gefiltert.
Bit 4-7 Diese Bits bestimmen die Resonanzfrequenz des Filters. Diese benutzt man dazu, bestimmte Ausschnitte des Frequenzspektrums hervorzuheben. Die Wirkung kann besonders gut bei der Sägezahnschwingung beobachtet werden.

REG 24 Dieses Register hat folgende Funktionen:

Bit 0-3 Gesamtlautstärke
Bit 4 Schaltet den Tiefpaßzweig des Filters ein.
Bit 5 Schaltet den Bandpaßzweig des Filters ein.
Bit 6 Schaltet den Hochpaßzweig des Filters ein.
Bit 7 1=Stimme 3 unhörbar. Von dieser Möglichkeit kann man Gebrauch machen, wenn der Verlauf der Stimme 3 nur zur Parametergewinnung für die anderen Stimmen dienen soll (siehe hierzu Register 27 und 28).

Auf alle zuvor aufgeführten Register kann nur ein Schreibzugriff durchgeführt werden. Ein Lesezugriff bringt keine Aussage. Alle folgenden Register können nur gelesen werden.

REG 25 A/D-Wandler 1

REG 26 A/D-Wandler 2

REG 27 Oszillator 3
Dieses Register liefert unter anderem eine Zufallszahl, die dem augenblicklichen Stand des Rauschgenerators 3 entspricht. Der Generator muß hierzu eingeschaltet sein, jedoch kann die Stimme 3 unhörbar bleiben (Bit 7 in REG 24 =1).

REG 28 Hüllkurvengenerator 3
Aus diesem Register kann man den augenblicklichen Stand der relativen Lautstärke von Stimme 3 entnehmen. So können entsprechend dem Lautstärkeverlauf die Frequenz oder die Filterparameter geändert werden.

WERTE FUER MUSIKNOTEN :

NR.	NOTE - OKTAVE	FREQUENZ	HIGH - BYTE	LOW - BYTE
1	C - 0	16.4	1 (01)	22 (16)
2	C# - 0	17.3	1 (01)	39 (27)
3	D - 0	18.4	1 (01)	57 (39)
4	D# - 0	19.4	1 (01)	75 (4B)
5	E - 0	20.6	1 (01)	95 (5F)
6	F - 0	21.8	1 (01)	116 (74)
7	F# - 0	23.1	1 (01)	138 (8A)
8	G - 0	24.5	1 (01)	161 (A1)
9	G# - 0	26	1 (01)	186 (BA)
10	A - 0	27.5	1 (01)	212 (D4)
11	A# - 0	29.1	1 (01)	240 (F0)
12	H - 0	30.9	2 (02)	14 (0E)
13	C - 1	32.7	2 (02)	45 (2D)
14	C# - 1	34.6	2 (02)	78 (4E)
15	D - 1	36.7	2 (02)	113 (71)
16	D# - 1	38.9	2 (02)	150 (96)
17	E - 1	41.2	2 (02)	190 (BE)
18	F - 1	43.7	2 (02)	231 (E7)
19	F# - 1	46.2	3 (03)	20 (14)
20	G - 1	49.0	3 (03)	66 (42)
21	G# - 1	51.9	3 (03)	116 (74)
22	A - 1	55.0	3 (03)	169 (A9)
23	A# - 1	58.3	3 (03)	224 (E0)
24	H - 1	61.7	4 (04)	27 (1B)
25	C - 2	65.4	4 (04)	90 (5A)
26	C# - 2	69.3	4 (04)	156 (9C)
27	D - 2	73.4	4 (04)	226 (E2)
28	D# - 2	77.8	5 (05)	45 (2D)
29	E - 2	82.4	5 (05)	123 (7B)
30	F - 2	87.3	5 (05)	207 (CF)
31	F# - 2	92.5	6 (06)	39 (27)
32	G - 2	98.0	6 (06)	133 (85)
33	G# - 2	103.8	6 (06)	232 (E8)
34	A - 2	110.0	7 (07)	81 (51)
35	A# - 2	116.5	7 (07)	193 (C1)
36	H - 2	123.5	8 (08)	55 (37)
37	C - 3	130.8	8 (08)	180 (B4)
38	C# - 3	138.6	9 (09)	56 (38)
39	D - 3	146.8	9 (09)	196 (C4)
40	D# - 3	155.6	10 (0A)	89 (59)
41	E - 3	164.8	10 (0A)	247 (F4)
42	F - 3	174.6	11 (0B)	158 (9E)
43	F# - 3	185.0	12 (0C)	78 (4E)
44	G - 3	196.0	13 (0D)	10 (0A)
45	G# - 3	207.7	13 (0D)	208 (D0)
46	A - 3	220.0	14 (0E)	162 (A2)
47	A# - 3	233.1	15 (0F)	129 (81)

NR.	NOTE - OKTAVE	FREQUENZ	HIGH - BYTE	LOW - BYTE
48	H - 3	246.9	16 (10)	109 (6D)
49	C - 4	261.6	17 (11)	103 (67)
50	C# - 4	277.2	18 (12)	112 (70)
51	D - 4	293.7	19 (13)	137 (89)
52	D# - 4	311.1	20 (14)	178 (B2)
53	E - 4	329.6	21 (15)	237 (ED)
54	F - 4	349.2	23 (17)	59 (3B)
55	F# - 4	370.0	24 (19)	157 (9D)
56	G - 4	392.0	26 (1A)	20 (14)
57	G# - 4	415.3	27 (1B)	160 (A0)
58	A - 4	440.0	29 (1D)	69 (45)
59	A# - 4	466.2	31 (1F)	3 (03)
60	H - 4	493.9	32 (20)	219 (DB)
61	C - 5	523.3	34 (22)	207 (CF)
62	C# - 5	554.4	36 (24)	225 (E1)
63	D - 5	587.3	39 (27)	18 (12)
64	D# - 5	622.3	41 (29)	101 (65)
65	E - 5	659.3	43 (2B)	219 (DB)
66	F - 5	689.5	46 (2E)	118 (76)
67	F# - 5	740.0	49 (31)	58 (3A)
68	G - 5	784.0	52 (34)	39 (27)
69	G# - 5	830.6	55 (37)	65 (38)
70	A - 5	880.0	58 (3A)	138 (8A)
71	A# - 5	932.3	62 (3E)	5 (05)
72	H - 5	987.8	65 (41)	181 (B5)
73	C - 6	1046.5	69 (45)	157 (9D)
74	C# - 6	1108.7	73 (49)	193 (C1)
75	D - 6	1174.7	78 (4E)	36 (24)
76	D# - 6	1244.5	82 (52)	201 (C9)
77	E - 6	1318.5	87 (57)	182 (B6)
78	F - 6	1396.9	92 (5C)	237 (ED)
79	F# - 6	1480.0	98 (62)	115 (73)
80	G - 6	1568.0	104 (68)	78 (4E)
81	G# - 6	1661.2	110 (6E)	130 (82)
82	A - 6	1760.0	117 (75)	20 (14)
83	A# - 6	1864.7	124 (7C)	10 (0A)
84	H - 6	1975.5	131 (83)	106 (6A)
85	C - 7	2093.0	139 (8B)	59 (3B)
86	C# - 7	2217.5	147 (93)	130 (82)
87	D - 7	2349.3	156 (9C)	72 (48)
88	D# - 7	2489.0	165 (A5)	147 (93)
89	E - 7	2637.0	175 (AF)	107 (6B)
90	F - 7	2793.8	185 (B9)	218 (DA)
91	F# - 7	2960.0	196 (C4)	231 (E7)
92	G - 7	3136.0	208 (D0)	156 (9C)
93	G# - 7	3322.4	221 (DD)	4 (04)
94	A - 7	3520.0	234 (EA)	40 (28)
95	A# - 7	3729.3	248 (F8)	20 (14)

Abb. 4.7.1: Notentabelle

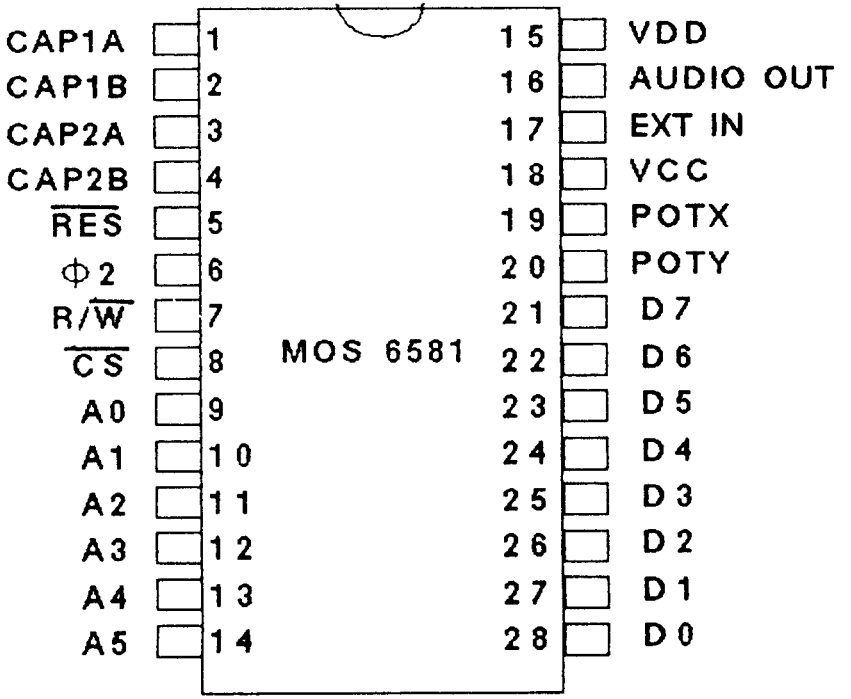


Abb. 4.7.2: Der MOS 6581

AUDIO / VIDEO :

Pin	Signal
1	Luminance
2	GND
3	Audio out
4	Video out
5	Audio in

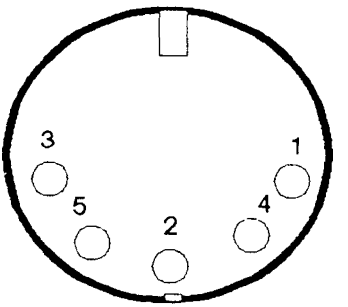


Abb. 4.7.3: Portbelegung

5. Die CIAs 6526

5.1 Datenein- und -ausgabe von Maschinenprogrammen

Will man eigene Maschinenprogramme schreiben, so kann man besonders für die Datenein- und ausgabe auf die Routinen des Betriebssystems zurückgreifen. Diese Routinen stehen in einer Sprungtabelle am Ende des ROMs (siehe dazu die letzte Seite des ROM-Listings).

5.1.1 Ein- und Ausgabe von einzelnen Bytes

Die grundlegenden Routinen sind

```
      BSOUT  $FFD2   Ausgabe eines Bytes
und   BASIN  $FFCF   Eingabe eines Bytes
```

Selbstverständlich gibt es noch weitere Routinen zur Ein-/Ausgabe. Diese sind jedoch nur sehr begrenzt anwendbar. Für die meisten Anwendungen reichen die oben aufgeführten allerdings aus.

Das auszugebende bzw. einzulesende Byte wird im Akku übergeben. Der Akku ist das Universalregister des Prozessors, in dem fast alle Operationen ablaufen.

Beispiel: Ausgabe eines Textes auf dem Bildschirm.

```
C000 LDX #$00     ; Zähler auf 00 setzen
C002 LDA $C00E,X  ; Text ab Adresse $C00E holen
C005 JSR $FFD2    ; Ausgabe auf Bildschirm
C008 INX          ; Zähler erhöhen
C009 CPX #$08     ; Schon alle Zeichen geholt ?
C00B BNE $C002    ; Nein, dann nächstes Zeichen
C00D RTS          ; Rücksprung von Subroutine

C00E 42 45 49 53 50 49 45 4C      Beispiel
```

Die Buchstaben sind als ASCII-Code ab Speicheradresse $C00E abgelegt.

Eingaben werden ähnlich gelöst. Soll ein Text über die Tastatur eingegeben und abgespeichert werden, so erscheint der Cursor, und die Zeichen werden bis zum Drücken der RETURN-Taste übernommen.

```
C000 LDX #$00     ; Zähler auf 00 setzen
C002 JSR $FFCF    ; Zeichen von der Tastatur holen
C005 STA $C00E,X  ; in diesem Fall ab $C00E speichern
C008 INX          ; Zähler erhöhen
C009 CMP #$0D     ; geholtes Zeichen auf RETURN testen
C00B BNE $C002    ; nein, dann nächtes Zeichen holen
C00D RTS          ; Rücksprung von Subroutine
```

Soll ein Zeichen von der Tastatur abgefragt werden, ohne auf dem Bildschirm zu erscheinen, so muß die folgende Routine verwendet werden.

```
C000 JSR $FFE4    ; Wirkung wie BASIC GET
C003 CMP #$20     ; vergleiche mit SPACE-Taste
C005 BNE $C000    ; nein, dann wieder ein Zeichen holen
```

Falls kein Zeichen gedrückt wurde, steht im Akku $00.
Bei der Ausgabe auf dem Bildschirm kann natürlich von der Bildschirmsteuerung, wie im BASIC bekannt, Gebrauch gemacht werden.

Dazu gehören zum Beispiel die Codes zur Cursorsteuerung oder zum Löschen des Bildschirms. Der entsprechende Code wird dazu in den Akku geladen und an die Ausgaberoutine übergeben.

Beipiel: Bildschirm löschen

```
LDA   #$93        ; Code zum Bildschirm löschen  (Dez. 147)
JSR   $FFD2       ; ausgeben
```

Speziell zur Bildschirmausgabe gibt es noch einige nützliche Routinen, die die Programmierung vereinfachen können.

Die Routine zum Löschen des Bildschirms kann auch direkt aufgerufen werden. (Anstelle des Umwegs über die Ausgabe eines Steuerzeichens)

```
JSR   $E544       ; Bildschirm löschen
```

Auch für Cursor Home existiert eine eigene Routine.

```
JSR   $E566       ; Cursor Home
```

Besonders interessant ist die Möglichkeit, den Cursor direkt auf eine bestimmte Bildschirmposition zu setzen. Das folgende Programm setzt den Cursor auf die Bildschirmkoordinaten 16,3.

```
C000 LDX #$10     ; Y-Koordinate festlegen. Hier $10
C002 LDY #$03     ; X-Koordinate festlegen. Hier $03
C004 CLC          ; Carryflag löschen, um Cursor zu setzen
C005 JSR $FFF0    ; Routine zum Setzen und Holen des Cursors
C008 LDA #$41     ; ASCII-Code für A
C00A JSR $FFD2    ; A auf Bildschirm ausgeben
```

Das Unterprogramm CURSOR $FFF0 hat zwei Funktionen. Bei Aufruf mit gelöschtem Carryflag setzt es den Cursor auf die Zeile und Spalte, die im X- und Y-Register stehen. Das Kuriose dabei ist, daß die X-Koordinate im Y-Register, und die Y-Koordinate im X-Register steht. Wird die CURSOR-Routine dagegen mit gesetztem Carryflag aufgerufen, wird die momentane Cursorposition geholt und im X- und Y-Register übergeben.

Zum Schluß noch zwei Routinen, die die Verarbeitung von Zeichen in Maschinensprache erleichtern. Sie waren vielleicht schon mit dem Problem konfrontriert, ein Byte (zum Beispiel $41) auch als HEX-Zahl "41", anstelle des Zeichens "A", auf dem Bildschirm erscheinen zu lassen. Nur leider gibt der Computer dieses gewünschte Ergebnis so ohne weiteres nicht aus.

Die folgenden zwei Routinen ermöglichen es, sowohl ein Byte - wie gerade beschrieben- als HEX-Zahl auszugeben, als auch zwei ASCII-Werte zu einem Byte zu verknüpfen.

Routine zur Ausgabe eines Bytes:

```
C002 PHA          ; auszugebenden Wert merken
C003 LSR          ; vier Bits nach rechts verschieben
C004 LSR          ; um HIGH-Nibble zu isolieren
C005 LSR
C006 LSR
C007 JSR $C00F    ; umrechnen und ausgeben
C00A PLA          ; gemerkten Wert holen
C00B JSR $C00F    ; LOW-Nibble ausgeben
C00E RTS          ; Rücksprung ins Hauptprogramm
C00F AND #$0F     ; LOW-Nibble isolieren
C011 CMP #$0A     ; Vergleiche mit Zahl
C013 CLC
C014 BMI $C018    ; ja, dann verzweigen
C016 ADC #$07     ; zu 7 addieren
C018 ADC #$30     ; zu 30 addieren
C01A JSR $FFD2    ; und ausgeben
C01D RTS          ; Rücksprung vom Unterprogramm
```

Wird diese Routine angesprungen, so muß sich das auszugebende Byte im Akku befinden

Das nächste Programm wartet zweimal auf eine Tasteneingabe und verbindet die Zeichen zu einem Byte.

```
C000 JSR $FFE4      ; Zeichen von Tastatur holen
C003 BEQ $C000      ; Kein Zeichen? Dann erneut holen
C005 PHA            ; Wert speichern
C006 JSR $F13E      ; zweites Zeichen holen
C009 BEQ $C006      ; Kein Zeichen? Dann erneut holen
C00B JSR $C01B      ; zur Umwandlungsroutine
C00E STA $FB        ; umgewandeltes Zeichen speichern
C010 PLA            ; zuerst geholtes Zeichen nehmen
C011 JSR $C01B      ; und umwandeln
C014 ASL            ; Bits an richtige Stelle schieben
C015 ASL
C016 ASL
C017 ASL
C018 ORA $FB        ; und mit gemerktem Wert verknüpfen
C01A RTS            ; Rücksprung ins Hauptprogramm
C01B CMP #$41       ; Buchstabe oder Zahl?
C01D SEC
C01E BMI $C022      ; Wenn Zahl, dann verzweigen
C020 SBC #$08       ; acht abziehen
C022 SBC #$30       ; 30 abziehen
C024 RTS            ; Rückkehr vom Unterprogramm
```

Nach Aufruf dieser Routine steht das geholte Byte im Akku.

5.1.2 Ein- und Ausgabe über Peripheriegeräte

Auch für die Ein- und Ausgabe auf Peripheriegeräte stellt das Betriebssystem die notwendigen Routinen bereit. Dazu soll kurz auf das Konzept der Ein-/Ausgabe eingegangen werden. Den Peripheriegeräten wird eine Nummer von 0 bis 15 zugewiesen, über die sie vom Betriebssystem angesprochen werden.

Nummer	Gerät
0	Tastatur
1	Kasettenrecorder
2	RS-232 Schnittstelle
3	Bildschirm
4 - 15	Geräte am seriellen IEC-Bus (Drucker, Floppy)

Zu dieser Geräte- oder "Primäradresse" kommt noch optional eine Sekundäradresse, die die Arbeitsweise des Peripheriegeräts bestimmt, sowie ein Datei- oder 'File'-name.

Um nun nicht jedesmal alle Parameter angeben zu müssen, wenn ein Peripheriegerät angesprochen wird, wird die logische Filenummer eingeführt. Zu jeder Filenummer werden einmal mit OPEN die Primär- und Sekundäradresse sowie ein Filename zugeordnet. Jeder weitere Bezug geschieht dann über die logische Filenummer.

Vor der ersten Ein- oder Ausgabe ist die Datei zu öffnen. Das kann von BASIC aus mit OPEN geschehen, aber auch in Maschinensprache. Dazu müssen vorher die Fileparameter gesetzt werden. Die logische Filenummer muß in der Adresse $B8 (184) stehen, die Gerätenummer in Adresse $BA (186), die Sekundäradresse in Adresse $B9 (185), die Länge des Filenamens in $B7 (183) (Null, wenn kein Filename gegeben ist) und die Adresse des Filenamens in $BB/$BC (187/188). Anschließend wird die OPEN-Routine($FFC0) aufgerufen.

Zum Übergeben der Parameter ist es nicht nötig, die Werte einzeln in die zugehörigen Adressen zu schreiben. Das Betriebssystem verfügt über zwei Routinen, die diese Arbeit übernehmen.

```
LDA   LF        ; logische Filenummer
LDX   GA        ; Geräteadresse
LDY   SA        ; Sekundäradresse
JSR   $FFBA     ; Fileparameter übergeben
LDA   LN        ; Länge des Filenamens
LDX   LOW       ; LOW-Byte der Adresse des Filenamens
LDY   HIGH      ; HIGH-Byte der Adresse
JSR   $FFBD     ; Filenameparameter übergeben
```

Soll jetzt die Ausgabe auf die geöffnete Datei erfolgen, so ist folgende Routine aufzurufen:

```
LDX   LF        ; logische Filenummer
JSR   $FFC9     ; CKUOT, Ausgabe auf Datei legen
```

Wird jetzt die Routine BSOUT ($FFD2) aufgerufen, so geht die Ausgabe anstatt auf den Bildschirm auf das Gerät, dem die logische Filenummer im X-Register zugeordnet ist.

Ist LF die logische Filenummer des Druckers, so würde jetzt durch Aufruf des obigen Beispielprogramms AUSGABE der Text auf den Drucker geschrieben. Die Ausgabe geht solange auf dieses Gerät, bis die Routine CLRCH aufgerufen wird

```
JSR   $FFCC     ; CLRCH , Ausgabe auf Bildschirm
```

Soll die Dateneingabe aus einer Datei geschehen, die sich auf Band oder auf die Floppy bezieht, kann man das folgendermaßen erreichen:

```
LDX   LF        ; Filenummer des Eingabegeräts
JSR   $FFC6     ; CHKIN
```

Jetzt werden durch Aufrufen von BASIN ($FFCF) Daten aus der geöffneten Datei geholt. Dies geschieht solange, bis mit CLRCH wieder auf die Standardeingabequelle (Tastatur) umgeschaltet wird. Die Datei wird dadurch nicht geschlossen. Dies geschieht erst durch Aufruf der Routine CLOSE.

```
LDA   LF        ; logische Filenummer
JSR   $FFC3     ; CLOSE, Datei schließen
```

Insbesondere bei Schreiboperationen auf den Kassettenrecorder oder die Floppy darf die CLOSE-Routine auf keinen Fall vergessen werden.

5.2 Die Technik der Datenspeicherung - LOAD und SAVE

Zur Daten- und Programmspeicherung stehen Ihnen beim Commodore 64 grundsätzlich zwei Möglichkeiten zur Verfügung - Speicherung auf Kassette oder Speicherung auf Diskette. Da sich die beiden Speichermedien in Möglichkeiten und Art der Datenspeicherung sehr unterscheiden, sollen sie getrennt beschrieben werden.

5.2.1 Datenspeicherung auf Kassette

Die Technik des Kassettenrekorders erlaubt es prinzipiell nur Daten sequentiell aufzuzeichnen und auch nur in der gleichen Reihenfolge wieder zu lesen. Einen wahlweisen Zugriff auf bestimmte Daten ist also nicht möglich. Man muß solange lesen, bis man die gewünschten Daten erreicht hat. Es lassen sich nur die Datei komplett lesen, die Änderung vornehmen und dann die Datei wieder komplett auf Band zurückschreiben.

Da zur Programmspeicherung nur ein sequentielles Schreiben und Lesen notwendig ist, bietet sich der Kassettenrekorder als preiswertes Gerät zur Programmspeicherung an. Als Nachteil bleibt jedoch die geringe Geschwindigkeit, mit der die Auszeichnung geschieht. Dies ist ein prinzipieller Nachteil, da die Daten seriell (bitweise) übertragen und gespeichert werden. Aus Gründen der Datensicherheit werden die gesamten Daten zweimal hintereinander übertragen, um Aussetzer (drop outs) durch fehlerhafte Bandstellen korrigieren zu können.

Sehen wir uns jetzt die Technik der Datenspeicherung auf Kassette etwas genauer an. Sollen Daten auf Band geschrieben werden, so muß erst einmal festgestellt werden, ob das Band läuft. Der Computer kann eine gedrückte Taste der Datasette registrieren, indem er Bit 4 des Prozessorports (Adresse $01) abfragt. Ist dieses Bit der Speicherzelle 1 gelöscht, so ist eine Bandtaste gedrückt. Nachdem der Computer eine gedrückte Taste registriert hat, beginnt er seine Arbeit.

Zuerst wird ein Ton zur Synchronisation auf das Band geschrieben. Anschließend werden die Daten geschrieben.

Bevor die eigentlichen Programmdaten aufgezeichnet werden, wird ein sogenannter Header geschrieben, der unter anderem den Datentyp angibt.

Wie bereits gesagt, werden die Daten zweimal auf Band geschrieben. Dies geschieht, um eine höhere Datensicherheit zu erzielen. Damit diese Prozedur nicht zu lange dauert, werden die zu schreibenden Daten zuerst in einem Puffer gesammelt, bevor sie auf Band geschrieben werden. Der Bandpuffer ist 192 Zeichen lang und liegt beim Commodore 64 von Adresse $033C bis $03FB (828 - 1019). Beim Einlesen wird immer ein ganzer Block gelesen und im Bandpuffer gespeichert von wo aus die Daten mit INTUT# oder GET# einzeln geholt werden können.

Durch das zweimalige Schreiben der Daten ist der Computer in der Lage, maximal 31 Fehler pro Block zu korrigieren, sofern diese Daten im zweiten Block lesbar sind.

Tritt ein Fehler auf, so wird dieser durch setzten des entsprechenden Bits im Statusregister angezeigt.

Hier eine Bescheibung der möglichen Fehlerarten:

Bit 2 gesetzt:

Dieser Fehler tritt auf, wenn ein Blockende gefunden wurde, jedoch noch nicht alle Daten gelesen wurden.

Bit 3 gesetzt:

Zu diesem Fehler kommt es, wenn alle Daten eines Blocks gelesen wurden, jedoch kein Blockende erkannt wurde.

Bit 4 gesetzt:

Bit 4 ist gesetzt, wenn ein Byte nicht korrekt gelesen werden konnte und dieses Byte im zweiten Block auch fehlerhaft war.

Bit 5 gesetzt:

Bit 5 ist gesetzt, falls die beim Speichern errechnete Prüfsumme des Blocks nicht mit der Prüfsumme beim Lesen übereinstimmt. Die Prüfsumme wird gebildet, indem die Daten des Blocks mit der Exklusive-Oder-Anweisung verknüpft werden.

Sehen wir uns den oben erwähnten Header einmal näher an. Das Interessanteste am Header ist das erste Byte. Es ist das Kennzeichen für den Datentyp.

Der Header ist folgendermaßen aufgebaut:

1.	Byte	Kennzeichen für Datentyp
2.	Byte	Startadresse, Low Byte
3.	Byte	Startadresse, High Byte
4.	Byte	Endadresse, Low Byte
5.	Byte	Endadresse, High Byte
6.- 21.	Byte	Filename, der bei FOUND ausgegeben wird
22.-192.	Byte	Filename, der nicht ausgegeben wird

Der Datentyp (erstes Byte) hat folgende Bedeutung:

Datentyp 1:

Bei Datentyp 1 handelt es sich meist um BASIC-Programme. Sie werden im allgemeinen ab BASIC-Startadresse $0801 (2049) geladen. Durch Laden einer Sekundäradresse, die ungleich eins ist, wird das Programm an die Adresse geladen, von der es gesaved wurde.

Datentyp 2:

Der Datentyp zwei ist die Kennzeichnung für den Anfang eines Dateiblocks. Nach ihm folgen weder Start- noch Endadresse und auch kein Programmname. Von Byte 2 bis Byte 192 folgen Daten, die mit PRINT# geschrieben und mit INPUT# oder GET# gelesen werden. Es können auch mehrere Blöcke von Typ 2 hintereinander stehen, je nach Länge der Datei. Damit der Computer die Daten dieser Blöcke verarbeiten kann, muß er zuvor einen Dateiheader mit Datentyp 4 finden. In diesem Header findet er auch den Dateinamen.

Datentyp 3:

Wird im Header der Datentyp 3 erkannt, so wird das folgende Programm, ohne Änderung des Betriebssystems, immer ab der im Header angegebenen Adresse geladen. Der Headertyp 3 wird dann benutzt, wenn Maschinenprogramme geladen werden sollen, deren Anfangsadresse nicht dem BASIC-Start entspricht. Dieser Datentyp wird bei Programmen mit AUOTSTART verwendet. Das Angeben einer eigenen Startadresse, was beispielsweise zur Unterdrückung eines AUTOSTART führt, ist im Kapitel "Nutzen des ROM-Listings" beschrieben.

Datentyp 5:

Wird im Header der Datentyp 5 erkannt, so sollte die Fehlermeldung "FILE NOT FOUND ERROR" ausgegeben werden. Stattdessen erfolgt die Ausgabe der sinnlosen Meldung "DEVICE NOT PRESENT ERROR". Die einfache Erklärung dafür ist, daß es sich dabei um einen Fehler im Betriebssystems handelt. Das Aufbringen einer End of Tape (EOT) Markierung dient dazu, das Ende aller auf dem Band befindlichen Programme und Dateien zu markieren, um ein unnötiges weiteres Suchen zu verhindern.

Wird beim Schreiben eines Files, was ein Programm oder eine Datei sein kann, ein Filename, der aus mehr als 16 Zeichen

besteht, angegeben, so werden diese "überschüssigen" Zeichen durchaus gespeichert. Der Block hat ja noch 187 Bytes übrig, die vom Filenamen belegt werden können. Diese "überschüssigen" Zeichen werden zwar nicht bei der FOUND-Meldung ausgegeben, jedoch überprüft der Computer beim Finden des Namens auf diese unsichtbaren Zeichen. Es ist möglich, diese Zeichen mit der PEEK-Anweisung auszulesen. Von dieser Möglichkeit wird aber normalerweise kein Gebrauch gemacht.

Die Endadresse des geladenen Programms steht nach dem Laden in den Adressen $AE und $AF (174 und 175). $AE enthält das LOW- und $AF das HIGH-Byte.

Welche Befehle werden benutzt, um all diese verschiedenen Dateitypen auf Band zu schreiben und von Band zu lesen? Im folgenden eine Übersicht darüber:

LADEN

LOAD"NAME",1	lädt Programm ab BASIC-Startadresse. Programme des Datentyps 3 werden absolut geladen, das heißt an die auf Band aufgezeichnete Adresse.
LOAD"NAME",1,1	Programm wird immer absolut geladen

SPEICHERN

SAVE "NAME",1	speichert als BASIC-Programm ab
SAVE "NAME",1,1	speichert als Maschinenprogramm ab
SAVE "NAME",1,2	speichert als BASIC-Programm mit zusätzlichem EOT-Block
SAVE "NAME",1,3	speichert als Maschinenprogramm mit zusätzlichem EOT-Block

Beim Öffnen einer Banddatei hat die Sekundäradresse folgende Bedeutung:

OPEN 1,1,0, "NAME" öffnet Datei zum Lesen
OPEN 1,1,1, "NAME" öffnet Datei zum Schreiben
OPEN 1,1,2, "NAME" öffnet Datei zum Schreiben mit zusätzlichem EOT-Block

Der CLOSE-Befehl schließt eine Datei wieder. War die Datei zum Schreiben geöffnet, so wird in den Bandpuffer ein Endkennzeichen (Nullbyte) und der Puffer auf Band geschrieben.

Daraus wird klar, daß es unbedingt erforderlich ist, nach dem Schreiben auf eine Banddatei diese mit CLOSE zu schließen, da sonst die letzten Daten nicht auf Band geschrieben werden und verlorengehen. Wurde die Sekundäradresse 2 angegeben, wird zusätzlich noch ein END-of-Tape-Block (EOT) auf Band geschrieben.

Es ist zu beachten, daß es zu einer Einschränkung bei der Datenübertragung auf Band kommen kann. Normalerweise werden die Daten im ASCII-Format auf Band geschrieben (Buchstaben, Ziffern und Sonderzeichen). Werden mit PRINT#1, CHR$(1) Daten geschrieben, so lassen sich nicht alle möglichen Zeichen schreiben. Insbesondere wird CHR$(0) ausgefiltert, da es vom Betriebssystem als Endkennzeichen verwendet wird.
Auf das Laden und Speichern von Programmen werden wir im nächsten Abschnitt eingehen.

KASSETTE :

Pin	Signal
A – 1	GND
B – 2	+ 5V
C – 3	KASSETTE MOTOR
D – 4	KASSETTE READ
E – 5	KASSETTE WRITE
F – 6	KASSETTE SENSE

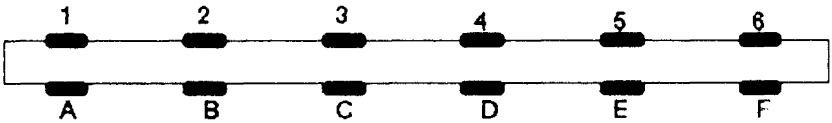


Abb. 5.2.1 Casetten-Port

5.2.2 Datenspeicherung auf Diskette

Alle Einschränkungen, die bei den Kassettenoperationen beschrieben wurden, bestehen bei der Datenübertragung auf Diskette nicht. Eine Diskette ist in einzelne Spuren und Sektoren unterteilt, auf die wahlweise zugegriffen werden kann. Dadurch ist es möglich, direkt auf die gewünschten Daten zuzugreifen, ohne erst eine Vielzahl anderer Daten überlesen zu müssen.

Die Commodore-Floppy 5141, das Diskettenlaufwerk des C64, hat folgende Daten:

35 Tracks (mit Tricks bis zu 40)
21 Blöcke auf Track 01-17
19 Blöcke auf Track 18-24

18 Blöcke auf Track 25-30
17 Blöcke auf Track 31-35
664 Blöcke Speicherkapazität (ca. 164 KB)

Programme werden folgendermaßen gespeichert: Zuerst werden das LOW-Byte und das HIGH-Byte der Programmadresse übertragen und unmittelbar danach das Programm selbst. Ein Unterschied zwischen BASIC-Programm und Maschinenprogramm wird beim Abspeichern nicht gemacht. Die Unterscheidung findet nur beim Laden statt.

LOAD "NAME",8 lädt BASIC-Programm. Die Programmstartadresse wird ignoriert, es wird ab BASIC-Start geladen (normalerweise $0801).

LOAD "NAME",8,1 lädt Maschinenprogramm, das Programm wird ab der beim Speichern angegebenen Startadresse geladen.

Mit dem folgendem Programm können sie die auf Diskette vermerkte Anfangsadresse feststellen.

```
10 OPEN 1,8,0,"NAME"
20 GET#1, A$, B$
30 IF A$ = "" THEN A$ = CHR$(0)
40 IF B$ = "" THEN B$ = CHR$(0)
50 PRINT ASC(A$)+256*ASC(B$)
60 CLOSE 1
```

Zuerst wird ein File zum Laden geöffnet (Sekundäradesse 0) und daraufhin die ersten zwei Bytes des Programms von Disk geholt. Sie bilden die Startadresse, die dezimal ausgegeben wird. Die Abfrage auf den Leerstring in Zeile 30 und 40 ist deshalb erforderlich, weil der GET-Befehl ein Nullbyte nicht akzeptiert.

Soll ein Maschinenprogramm auf Diskette geschrieben werden, so kann dies so geschehen:

```
10 OPEN 1,8,1,"NAME"
20 PRINT#1, CHR$(AD-INT(AD/256)*256);
30 PRINT#1, CHR$(AD/256);
40 FOR I=0 TO N -1
50 PRINT#1, CHR$(PEEK(AD+I));
60 CLOSE 1
```

In Zeile 10 wird ein Programmfile zum Schreiben geöffnet (Sekundäradesse 0). Daraufhin wird die Startadresse des Programms in der Reihenfolge LOW- und HIGH-Byte auf Disk geschrieben. Anschließend werden die Programmdaten geschrieben und im Anschluß daran die Datei wieder geschlossen. Das Schließen der Datei ist sehr wichtig, da ansonsten Daten verloren gehen können. Die Variable AD enthält dabei die Startadresse, N ist die Länge des Programms in Bytes.

Wie kann man nun Programme oder beliebige Speicherbereiche von Maschinensprache aus abspeichern?
Auch hierzu existieren wieder zwei Betriebssystem-Routinen, die diese Arbeit übernehmen.

```
LOAD  $FFD5
SAVE  $FFD8
```

Die LOAD-Routine wird folgendermaßen benutzt:

Der Akku wird mit Null geladen. Dies ist das Kennzeichen für LOAD. Wird die LOAD-Routine mit 1 im Akku aufgerufen, wird VERIFY durchgeführt. Die Sekundäradresse entscheidet, wohin geladen wird. Ist die Sekundäradresse ungleich Null, so wird an die Adresse geladen, die im Programm gespeichert ist. Ist die Sekundäradresse gleich null, wird an die Adresse geladen, die im X- (low) und Y-Register (high) übergeben wird. Ferner müssen Geräteadresse und Filename ausgegeben sein. Auch dazu gibt es ROM-Routinen.

Beispiel: Laden eines Maschinenprogramms von Diskette, mit dem Namen "NAME", an die Adresse $1000.

```
C000 LDX #$08     ; Geräteadresse für Floppy
C002 LDY #$00     ; Sekundäradresse 00
C004 JSR $FFBA    ; Fileparameter setzen
C007 LDA #$04     ; Anzahl der Zeichen des Namens
C009 LDX #$1E     ; LOW-Byte der Adresse des Namens
C00B LDY #$C0     ; HIGH-Byte der Adresse
C00D JSR $FFBD    ; Filenamenparameter setzen
C010 LDA #$00     ; $00 Flag für LOAD
C012 LDX #$00     ; LOW-Byte der Programmanfangsadresse
C014 LDY #$10     ; HIGH-Byte der Adresse
C016 JSR $FFD5    ; LOAD-Routine
C019 STX $AE      ; LOW-Byte der Endadresse speichern
C01B STY $AF      ; HIGH-Byte der Adresse speichern
C01D RTS          ; Rücksprung vom Hauptprogramm

C01E 4E 41 4D 45 00 00 00 00    NAME
```

Wie aus dem Beispiel ersichtlich, übergibt die LOAD-Routine im X- und Y-Register die Endadresse des geladenen Programms. Um das Programm absolut an die gespeicherten Adressen zu laden, muß als Sekundäradresse 01 gewählt werden. In diesem Fall könnten auch die Befehle in den Speicherzellen $C012 bis $C015 entfallen.

An dieser Stelle nun der angekündigte Nachtrag zum Laden eines Programms von Band. Im nächsten Beispiel soll ein Programm vom Band ab Adresse $6000 geladen werden, die mit abgespeicherte Adresse wird ignoriert, sofern der Datentyp nicht 3 ist. Ist dies der Fall, so ist es nicht ohne weiteres möglich, das zu ladende Programm auf eine beliebige Adresse zu legen. Wie dies jedoch dennoch geschehen kann, lesen Sie bitte im Kapitel 6.1 "Nutzung des ROM-Listings" nach.

```
C000 LDX #$01     ; Geräteadresse für Band
C002 LDY #$00     ; Sekundäradresse 00
C004 JSR $FFBA    ; Fileparameter setzen
C007 LDA #$04     ; Anzahl der Zeichen des Namens
```

```
C009 LDX #$1E      ; LOW-Byte der Adresse des Namens
C00B LDY #$C0      ; HIGH-Byte der Adresse
C00D JSR $FFBD     ; Filenamenparameter setzen
C010 LDA #$00      ; $00 Flag für LOAD
C012 LDX #$00      ; LOW-Byte der Programmanfangsadresse
C014 LDY #$60      ; HIGH-Byte der Adresse
C016 JSR $FFD5     ; LOAD-Routine
C019 STX $AE       ; LOW-Byte der Endadresse speichern
C01B STY $AF       ; HIGH-Byte der Adresse speichern
C01D RTS           ; Rücksprung vom Hauptprogramm

C01E 4E 41 4D 45 00 00 00 00    NAME
```

Wie Sie sehen, besteht der einzige Unterschied im Laden von Programmen von Band oder von Disk in der veränderten Geräteadresse.

Mit einem Maschinensprachemonitor wird das Programm ohne weitere Angaben immer absolut geladen und gespeichert.

Soll von einem BASIC-Programm aus ein Maschinenprogramm mit LOAD geladen werden, so ergeben sich einige Schwierigkeiten. Absolutes Laden läßt sich zwar leicht durch Angabe der Sekundäradresse 1 erreichen. Es besteht jedoch noch ein zweites Problem. Nach jedem LOAD wird die Endadresse des geladenen Programms immer gleich der Endadresse des BASIC-Programms gesetzt, und die Programmausführung beginnt wieder am Programmanfang. Dies ist für das Nachladen von BASIC-Programmen (Overlay) durchaus sinnvoll, macht jedoch beim Laden von Maschinenprogrammen oder sonstigen Speicherinhalten Probleme. In einem Overlay solchen Fall empfiehlt sich ein kleines Maschinenprogramm wie eines der obigen Beispiele, das z.B. vom BASIC-Programm aus mit SYS aufgerufen wird. Auch eine Parameterübergabe ist möglich, z.B. Filenamen und Geräteadresse.

Mit den nun gezeigten Beispielen lassen sich (allerdings mit den genannten Schwierigkeiten) Maschinenprogramme von BASIC aus laden.

```
10 IF A=1 THEN 40
20 A=1
30 LOAD "NAME",8,1
40 PRINT " PROGRAMM GELADEN"
50 END
```

Sollen mehrere Programme hintereinander geladen werden, so ist das mit dem folgenden BASIC-Programm möglich.

```
10 ON A GOTO 30,50,80
30 A=A+1
40 LOAD "NAME 1",8,1
50 PRINT " ERSTE PROGRAMM GELADEN"
60 A=A+1
70 LOAD"NAME 2",8,1
80 PRINT " ZWEITE PROGRAMM GELADEN"
90 END
```

Abspeichern von Programmen oder beliebigen Speicherbereichen mit der SAVE-Routine geschieht ähnlich. Hierzu muß der SAVE-Routine die Start- und Endadresse mitgeteilt werden. Außerdem werden Geräteadresse sowie Länge und Adresse des Filenamens (beim Abspeichern auf Diskette unbedingt erforderlich) benötigt. Die Startadresse muß in der Zeropage an zwei aufeinanderfolgenden Adressen stehen (LOW- und HIGH-Byte), im Akku wird ein Zeiger auf diese Adresse übergeben. Das LOW-Byte der Endadresse steht im X- und das HIGH-Byte im Y-Register. Geräteadresse und Filenamenparameter werden wie bei der LOAD-Routine gesetzt. Wir wollen als Beispiel jetzt den Speicherbereich von $C000 bis $C200 unter dem Namen "NAME" auf Diskette speichern und haben die Startadresse des Programms in $FB/$FC gespeichert.

```
C000 LDX #$08     ; Geräteadresse für Floppy
C002 LDY #$01     ; Sekundäradresse $01
C004 JSR $FFBA    ; Fileparameter übergeben
C007 LDA #$04     ; Anzahl der Zeichen des Namens
C009 LDX #$22     ; LOW-Byte der Adresse des Namens
```

```
C00B LDY #$C0     ; HIGH-Byte der Adresse
C00D JSR $FFBD    ; Filenamenparameter übergeben
C010 LDX #$00     ; LOW-Byte der Startadresse
C012 LDY #$C0     ; HIGH-Byte der Adresse
C014 STX $FB      ; LOW-Byte speichern
C016 STY $FC      ; HIGH-Byte speichern
C018 LDA #$FB     ; mitteilen, wo Adresse gespeichert wurde
C01A LDX #$01     ; LOW-Byte der Endadresse + 1
C01C LDY #$C2     ; HIGH-Byte der Endadresse
C01E JSR $FFD8    ; SAVE-Routine
C021 RTS          ; Rücksprung vom Unterprogramm

C022 4E 41 4D 45 00 00 00 00     NAME
```

Wie Sie aus dem Beispiel ersehen, wird der Inhalt der Endadresse nicht mehr abgespeichert, es muß deshalb immer die Endadresse plus eins angegeben werden. Zum Abschluß wollen wir noch ein BASIC-Programm ohne Filenamen mit EOT-Kennzeichen auf Kassette schreiben.

```
C000 LDX #$01     ; Geräteadresse für Band
C002 LDY #$02     ; Sekundäradresse 02 für End of Tape
C004 JSR $FFBA    ; Fileparameter übergeben
C007 LDA #$00     ; kein Name
C009 JSR $FFBD    ; Filenamenparameter übergeben
C00C LDA #$2B     ; Zeiger auf BASIC-Programmstart
C00E LDX $2D      ; LOW-Byte der BASIC-Endadresse
C010 LDY $2E      ; HIGH-Byte der BASIC-Endadresse
C012 JSR $FFD8    ; SAVE-Routine
C015 RTS          ; Rücksprung vom Unterprogramm
```

5.3 Die CIAs

Die CIAs (Complex Interface Adapter) sind zwei sehr leistungsfähige Interfacebausteine, deren Leistungsfähigkeit beim C64 jedoch nicht vollkommen ausgenutzt wird. Die CIAs sind die

Nachfolger der VIAs, die noch im VC20 und in der Floppy 1541 Verwendung gefunden haben.

Bevor wir auf die Arbeitsweise der CIAs eingehen, noch einige allgemeine Informationen:

Die beiden völlig identischen Bausteine unterscheiden sich lediglich in ihren Aufgaben und in der Weise, wie sie mit den übrigen Elementen des Computers verbunden sind. Der CIA 1 dient hauptsächlich der Tastaturabfrage. Zusätzlich werden von ihm der Joystick, Paddels, Lightpen und Maus verwaltet. Er ist mit der IRQ-Leitung (Interupt Request) des Computers verbunden und belegt den Adreßbereich von $DC00 bis $DCFF. Der CIA 2 hingegen dient ausschließlich der Steuerung von Peripheriegeräten. Seine Leitungen sind am Userport herausgeführt. Er ist mit der NMI-Leitung (Nicht Maskierbarer Interupt) des Computers verbunden und belegt den Speicherbereich von $DD00 bis $DDFF.

Der Interrupt der CIA 2 hat deshalb eine höhere Priorität im Vergleich zur CIA 1. Dies ist auch sinnvoll, da die CIA 2, wie bereits erwähnt, die zeitkritischen Ein- und Ausgabeoperationen ausführt.

Die CIAs verfügen jeweils über zwei freiprogrammierbare Timer, eine unbenutzte Echtzeituhr, und zwei freiprogrammierbare 8-Bit-Datenports.

5.4 Die E/A-Ports

5.4.1 Tastaturabfrage

Die CIA's verfügen über zwei 8-Bit-Datenregister (PRA und PRB) bei denen jede der Leitungen sowohl als Ein- als auch als Ausgang gewählt werden kann. Um dies festzulegen, verfügen die CIAs über jeweils zwei 8-Bit-Datenrichtungsregister (DDRA und DDRB). Wenn ein Bit im DDR gesetzt ist (=1), arbeitet das korrespondierende Bit des PR als Ausgang. Ist das Bit im DDR

gelöscht (=0), so ist das entsprechende Bit des PR als Eingang definiert. Durch das Auslesen der DDRs erfähr man die Eingangsspannungsbelegungen der PRs.

Die Portleitungen PA0 bis PA7 und PB0 bis PB7 des CIA 1 fragen unter anderem die Tastatur ab. Sie bilden eine 8-mal-8-Matrix, die es ermöglicht, 64 Tasten abzufragen.

Falls Sie die Tasten Ihres C64 einmal gezählt haben, haben Sie jedoch festgestellt, daß dieser 66 Tasten hat. Das ist dadurch zu erklären, daß die RESTORE-Taste über eine eigene Leitung verfügt, die bei einem Tastendruck die RESTORE-Leitung auf Masse legt und dadurch einen NMI (Nicht Maskierbarer Interrupt) auslöst.

Die zweite fehlende Taste ist SHIFT-LOCK. Sie ist parallel zur linken SHIFT-Taste geschaltet.

Eine Erklärung vorweg: Wenn ein Datenport im Datenrichtungsregister auf Eingang geschaltet ist, und dieser nicht belegt ist, so sind diese Bits im Datenregister auf HIGH geschaltet (gesetzt). Zum besseren Verständnis arbeiten Sie bitte mit der Tabelle am Ende des Kapitels.

Beim CIA 1 sind die Leitungen PA0 bis PA7 als Ausgang geschaltet, die Leitungen PB0 bis PB7 als Eingang. Fragt die Interruptroutine des Betriebssystems die Tastatur ab, so legt sie die Anschlüsse des Ports A kurzzeitig auf LOW, die Bits der Adresse $DC00 (56320) werden gelöscht.

Wird eine Taste gedrückt, so wird das Lowsignal des Port A über die Leiterbahn auf das entsprechende Bit des Port B übertragen.

Die anderen Bits des Port B sind in diesem Fall nicht angeschlossen und deshalb, wie bereits erklärt, gesetzt.

Erkennt das Betriebssystem einen vollständig gesetzten Port B, so wurde keine Taste gedrückt, und es wird zum Ende der Tastenabfrage gesprungen.

Wurde beispielsweise H gedrückt, so wird Bit 5 des Port B gelöscht. Daran kann der Rechner erkennen, daß eine der Tasten F3, S, F, H, K, :, = oder die Commodore-Taste gedrückt sein muß. Um festzustellen, welche dieser Tasten gedrückt ist, setzt der Rechner alle Bits des Port A bis auf das erste auf high und schiebt die Bits des Port B nacheinander ins Carry-Flag, um zu überprüfen, ob sie gelöscht sind. Sobald er ein gelöschtes Bit festgestellt hat, holt er sich den entsprechenden Tasten-Code aus einer der Tabellen ab $EB81. Aus welcher Tabelle der Tasten-Code geholt wird, hängt davon ab, ob SHIFT oder COMMODORE-Taste gleichzeitig betätigt wurden. Waren alle Bits des Port B gesetzt, wird das nächste Bit des Port A gelöscht und alle Übrigen gesetzt. Daraufhin wiederholt sich die Kontrolle des Port B.

Sind mehrere Tasten gedrückt, so werden entsprechend mehr Bits des Port B gelöscht. Daraus wird ersichtlich, daß es möglich ist, mehrere Tasten gleichzeitig abzufragen.

Als erstes nun ein Programm, daß es erlaubt, auf eine Taste oder auf eine Betätigung des Joysticks zu warten, ohne daß eine der Routinen zur Tastenabfrage angesprungen werden muß. Es ist also möglich, auf einen Tastendruck zu warten, obwohl der Interrupt, der die Tastatur abfragt, verhindert ist.

```
C000 LDA #$00
C002 STA $DC00     ; Port A auf Low legen
C005 LDA $DC01     ; Port B holen
C008 CMP #$FF      ; Taste gedrückt
C00A BEQ $C000     ; Nein, dann wieder zum Anfang
C00C RTS           ; Rücksprung vom Unterprogramm
```

Das folgende Programm fragt mehrere Tasten auf einmal ab. Das Programm springt nur aus der Schleife, falls DEL, W, und die RUN/STOP-Taste gleichzeitig gedrückt sind.

```
C000 SEI           ; Interrupt verhindern
C001 LDA #$00
C003 STA $DC00     ; Port A auf Low setzen
```

```
C006 LDA $DC01      ; Port B holen
C009 CMP #$FF       ; Taste gedrückt
C00B BEQ $C001      ; Nein, dann wieder von Anfang
C00D LDX #$00       ; Zähler auf $00 setzen
C00F LDA #$FE       ; Alle Bits bis auf das erste gesetzt
C011 PHA            ; Wert speichern
C012 STA $DC00      ; an Port A übergeben
C015 LDA $DC01      ; Wert von Port B holen
C018 CMP $C02D,X    ; Mit Wert aus Tabelle vergleichen
C01B BEQ $C021      ; Ja, dann weiter
C01D PLA            ; Ansonsten Stapel rücksetzen
C01E CLC            ; und zum
C01F BCC $C001      ; Anfang springen
C021 PLA            ; Gespeicherten Wert holen
C022 INX            ; Zähler erhöhen
C023 CPX #$08       ; Mit Endwert vergleichen
C025 BEQ $C02B      ; Ja, dann zum Ende
C027 SEC            ; Carry setzen
C028 ROL            ; gelöschtes Bit um eins verschieben
C029 BNE $C011      ; unbedingter Sprung
C02B CLI            ; Interrupt wieder ermöglichen
C02C RTS            ; Rücksprung vom Unterprogramm

C02D FE FD FF FF FF FF FF 7F; Daten für die Tasten
```

Die Bits des Port A werden einzeln, der Reihe nach gelöscht und die dazugehörige Bitkombination des Port B überprüft. Daraus ergeben sich acht Werte für Port B, die ab Adresse $C02D abgelegt sind.

Wie schon erwähnt, holt sich das Betriebssystem den ASCII Wert der gedrückten Taste aus einer Tabelle. Zuvor wird jedoch die soeben gedrückte Taste in einem anderen Code in der Adresse $CB und eine Kopie davon in der Adresse $CB abgelegt. Dieser Code entspricht der Stellung des entsprechenden ASCII-Wertes in der eben erwähnten Tabelle ($EB81).

Wenn es auf Geschwindigkeit ankommt, so empfiehlt es sich, die gedrückte Taste direkt in einer dieser Adressen abzufragen. Die

entsprechende Tabelle zur Identifizierung der Tasten finden Sie am Ende dieses Kapitels.

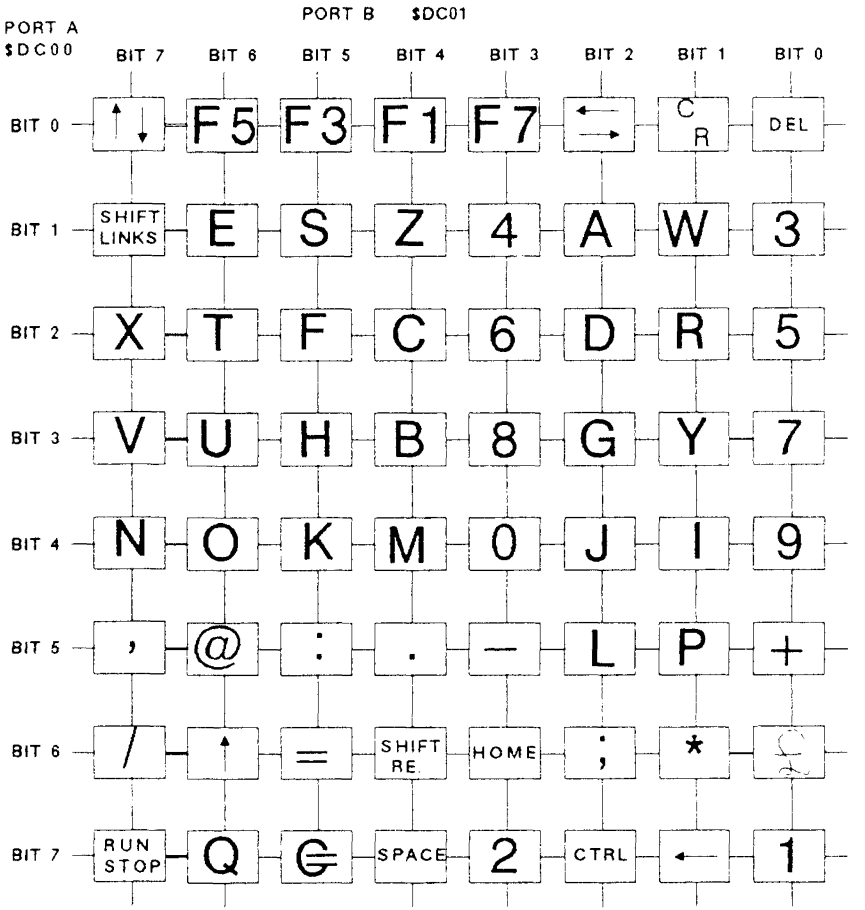


Abb. 5.4.1.1: Tastatur-Matrix

Tabelle der Tastencodes von $C5 und $CB

DEZ	HEX	BED.	DEZ	HEX	BED.	DEZ	HEX	BED.	DEZ	HEX	BED.
0	00	INST DEL	16	10	5 %	32	20	9)	48	30	£
1	01	CR	17	11	R	33	21	I	49	31	*
2	02	← →	18	12	D	34	22	J	50	32	;]
3	03	F7	19	13	6 &	35	23	0	51	33	CLR HOME
4	04	F1	21	14	C	36	24	M	52	34	Nicht belegt
5	05	F3	20	15	F	37	25	K	53	35	=
6	06	F5	22	16	T	38	26	O	54	36	↑ π
7	07	↓ ↑	23	17	X	39	27	N	55	37	? /
8	08	3 #	24	18	7 .	40	28	+	56	38	1 !
9	09	W	25	19	Y	41	29	P	57	39	←
10	0A	A	26	1A	G	42	2A	L	58	3A	Nicht belegt
11	0B	4 $	27	1B	8 (	43	2B	-	59	3B	2 "
12	0C	Z	28	1C	B	44	2C	>	60	3C	Space
13	0D	S	29	1D	H	45	2D	[	61	3D	Nicht belegt
14	0E	E	30	1E	U	46	2E	@	62	3E	Q
15	0F	Nicht belegt	31	1F	V	47	2F	<	63	3F	RUN STOP

64	$40 = keine Taste

Abb. 5.4.1.2: Tastencodierung

5.4.2 Joystick

Die Joystickabfrage erledigt der CIA 1.

Dazu werden für den Joystickport 1 die Bits PB0 bis PB4 und für Joystickport 2 die Bits PA0 bis PA4 der CIA 1 belegt. Über die Schalter des Joysticks wird Masse, das heißt ein Low-Signal, auf die entsprechende Portleitung gelegt.

Es ist selbstverständlich auch möglich, mehrere Leitungen auf low zu legen und dies dort abzufragen, sofern die entsprechenden Portleitungen im DDRA als Eingang definiert waren. Da der Joystickport 1 mit Port B der CIA 1 verbunden ist, wird auch verständlich, daß beim Betätigen des Joysticks "wunderliche" Zeichen auf dem Bildschirm erscheinen. Hier nun eine Übersicht über die Zuordnung der Portleitungen:

	Kontrollport 1	Kontrollport 2
	Adresse $DC01	Adresse $DC00
oben:	Port B0	Port A0
unten:	Port B1	Port A1
links:	Port B2	Port A2
rechts:	Port B3	Port A3
Feuer:	Port B4	Port A4

Mit dem folgenden BASIC-Programm kann man den Joystick in Joystickport 2 abfragen.

```
10 POKE 56322,224
20 J=PEEK (56320)
30 IF (J AND 1)=0THEN PRINT"OBEN"
40 IF (J AND 2)=0THEN PRINT"UNTEN"
50 IF (J AND 4)=0THEN PRINT"LINKS"
```

```
60 IF (J AND 8)=0THEN PRINT"RECHTS"
70 IF (J AND 16)=0THEN PRINT"KNOPF"
80 GOTO 20
```

In Zeile 10 werden die Leitungen PA0 bis PA4 des PRA als Eingang geschaltet. Dies geschieht im DDRA, der Register 2 der CIA 1 ($DC02) belegt. Daraufhin werden die Daten von Port A geholt und kontrolliert, welche Bits gelöscht wurden. Anschließend wird wieder zur Abfrage gesprungen. Das Programm können Sie nicht mit der STOP-Taste unterbrechen, da keine Tastenabfrage mehr möglich ist. Falls Sie das Programm in Ihr eigenes Programm einbauen wollen, so ist es nicht unbedingt nötig, auf eine Tastenabfrage zu verzichten. Durch POKE 56322,255 wird die Tastenabfrage wieder ermöglicht. Um die Joystickabfrage auf Joystickport 1 zu verlegen, brauchen Sie nur zu jeder Adresse eine 1 zu addieren.

5.4.3 Die Verwendung von Paddles

An den C64 können handelsübliche Paddles angeschlossen werden. Sie werden einfach in den Controlport 1 auf der rechten Seite des Gerätes eingestöpselt. Hinter einem Paddle verbirgt sich nichts weiter als ein Potentiometer, welches auf die im Kapitel 4 erläuterte Weise mit dem SID verbunden ist, und eine Taste, welche auf die Joystickposition LINKS für das eine Paddle und RECHTS für das andere Paddle wirkt.

Ein wenig problematisch ist das Verfahren, die Paddlewerte programmtechnisch auszuwerten, da sich einige Bits zur Paddlesteuerung und -abfrage die CIAs 1 und 2 mit der Tastatur teilen müssen.

Probleme bereiten zum einem die Tasten an den Paddles und zum anderen die Tatsache, daß Anschlußmöglichkeiten für zwei Paddlepaare (also vier Paddles) bestehen, der SID aber nur über zwei Analog/Digital-Wandler verfügt. Letzteres Problem wird

dadurch gelöst, indem die Paddles über einen Analogschalter geführt werden.Um diesen zu bedienen, werden zwei weitere Bits der CIA 1 herangezogen.

Es muß also auch hier zur Auswertung der A/D-Wandler die Tastatur lahmgelegt werden, allerdings nur für die Zeit des tatsächlichen Zugriffs, da man sonst mit der Tastatur nicht mehr arbeiten könnte.

Die Lösung bringt folgendes kleine Maschinenprogramm. Es wird als DATA-Statements in ein BASIC-Programm eingebunden und erlaubt, wie weiter unten gezeigt, den komfortablen Zugriff auf alle Parameter von vier angeschlossenen Paddles.
Das Programm ist in einem Bereich angelegt, der vom Betriebssystem nicht benutzt wird.

Die Rückmeldungen belegen einige Speicherzellen, die nur während einer Kassettenoperation anderweitig belegt sind.

```
CFBE  SEI       ;Tastaturabfrage verhindern
CFBF  LDA #$80  ;Parameter für Paddlesatz A
CFC1  JSR $CFEC ;A/D-Werte A1 und A2 holen
CFC4  STX $033C ;und sicherstellen
CFC7  STY $033D
CFCA  LDA $DC00 ;Tasten A aus CIA 1 holen
CFCD  AND #$0C  ;benötigte Bits filtern
CFCF  STA $029F ;und sicherstellen
CFD2  LDA #$40  ;Parameter für Paddlesatz B
CFD4  JSR $CFEC ;A/D-Werte B1 und B2 holen
CFD7  STX $033E ;und sicherstellen
CFDA  STY $033F
CFDD  LDA $DC01 ;Tasten B aus CIA 2 holen
CFE0  AND #$0C  ;benötigte Bits filtern
CFE2  STA $02A0 ;und sicherstellen
CFE5  LDA #$FF  ;alle Bits Ausgang in CIA 1
CFE7  STA $DC02 ;um Tastaturabfrage wieder
CFEA  CLI       ;zu erlauben
CFEB  RTS       ;Rückkehr ins Basicprogramm
CFEC  STA $DC00 ;Paddlesatz auswählen
```

```
CFEF  ORA #$C0  ;und entsprechende Bits
CFF2  STA $DC02 ;auf Ausgang setzen
CFF4  LDX #$0   ;Verzögerungsschleife zur
CFF6  DEX       ;Beruhigung des
CFF7  BNE $CFF6 ;A/D-Eingangs
CFF9  LDX $D419 ;A/D 1 holen
CFFC  LDY $D41A ;A/D 2 holen
CFFF  RTS       ;Rückkehr ins Hauptprogramm
```

Hier nun das BASIC-Programm. Schließen Sie die Paddles an, laden und starten Sie das Programm und schauen Sie, was geschieht.

```
10  DATA 120,169,128,32,236,207,142,60,3,140,61,3,173
20  DATA 0,220,41,12,141,159,2,169,64,32,236,207,142
30  DATA 62,3,140,63,3,173,1,220,41,12,141,160,2,169
40  DATA 255,141,2,220,88,96,141,0,220,9,192,141,2
50  DATA 220,162,0,202,208,253,174,25,212,172,26,212,96
60  FOR M=53182 TO 53247
70  READ A: POKE M,A: NEXT: REM Maschinenprogramm einlesen
80  AX=830: REM Paddle 1 am Controlport 1
90  AY=831: REM Paddle 2 am Controlport 1
100 BA=672: REM Tasten von Paddlepaar A
110 BX=828: REM Paddle 1 am Controlport 2
120 BY=829: REM Paddle 2 am Controlport 2
130 BB=830: REM Tasten von Paddlepaar B
140 SYS 53182: REM Alle Werte holen
150 PRINT PEEK(AX)" "PEEK(AY)" "PEEK(BA)" ";
160 PRINT PEEK(BX)" "PEEK(BY)" "PEEK(BB)
170 GOTO 140
```

5.4.4 MAUS

Die Maus ist ein sehr leicht zu bedienendes Eingabeinstrument, das leider noch kaum zum C64 durchgedrungen ist. Vor kurzem sind jedoch einige Mäuse auf den Markt erschienen, die ohne weiteres an dem C64 anschließbar sind.

Sie belegen die gleichen Anschlüsse wie auch der Joystick und legen den Anschluß für die jeweilige Richtung auf Masse. Im Prinzip ist dies nämlich ein als Maus "getarnter" Joystick. Durch das Joystick-kompatible Signal ergibt sich auch umgehend ein Problem. Es ist nicht möglich, die Geschwindigkeit, mit der die Maus bewegt wird, festzustellen. Man kann lediglich mit einem konstanten Wert für die Geschwindigkeit arbeiten.

Doch wie funktioniert eigentlich so eine Maus? Die Bewegung der in der Maus befindlichen Kugel wird auf zwei Rollen übertragen. Die eine Rolle übernimmt die Bewegung in X- und die andere in Y-Richtung. An jeder der Rollen ist ein Kranz mit Löchern befestigt. Die Löcher lassen einen Lichtstrahl, der auf einen Fotowiderstand fällt, ungehindert durch. Durch Drehen des Kranzes entsteht ein ständiger Wechel zwischen hell und dunkel. Die Bewegung der Maus wird somit nicht wie beim Joystick durch das Vorhanden- und Nichtvorhandensein des Signals, sondern durch das ständig gesendete, sich ständig ändernde Signal erkannt.

Bei der Maus für den C64 werden diese Impulse durch eine zusätzliche Schaltung in ein Joysticksignal umgewandelt. Diese Umwandlung ermöglicht es, die Maus einzustöpseln und die Tastatur trotzdem abzufragen.

Das folgende Maschinenprogramm ermöglicht es, eine Maus, die nicht speziell für den C64 entwickelt wurde, abzufagen. In diesem Fall ist es die Maus für den Atari 520 ST. Aufgrund der Tatsache, daß diese Maus ständig ein Signal an den Port schickt, ist es nicht möglich, zusätzlich die Tastatur abzufragen. Das ist jedoch auch der einzige Nachteil. Dafür bekommen sie eine "richtige" Maus und keine Joystick-"Imitation".

Das Sprite, das durch die Maus gesteuert wird, muß zusätzlich zu diesem Maschinenprogramm noch ab Adresse $0340 (832) abgelegt werden. Im angefügten BASIC-Lader ist dies bereits geschehen.

```
C000 SEI            ; Tastaturabfrage verhindern
C001 LDA #$01
C003 STA $D015      ; Sprite eins einschalten
C006 LDA #$60       ; Koordinaten festlegen
C008 STA $D000
C00B STA $D001
C00E LDA #$0D
C010 STA $07F8      ; Spritezeiger auf $0340 (832) stellen
C013 LDX #$00
C015 STX $DC02      ; Port A auf Eingang stellen
C018 LDA $DC00      ; Port lesen
C01B PHA            ; und merken
C01A AND #$03       ; erste 2 Bits isolieren (Horizontale)
C01E CMP $FB        ; mit altem Wert vergleichen
C020 BEQ $C036      ; ja, dann zur nächsten Abfrage
C022 LDY $FB        ; ansonsten alten Wert lesen
C024 STA $FB        ; und neuen Wert speichern
C026 CMP $C0C0,Y    ; Wert für entsprechende Bewegung?
C029 BNE $C02E      ; nein, andere Bewegung vergleichen
C02B JSR $C060      ; Sprite nach rechts bewegen
C02E CMP $C0C4,Y    ; entgegengesetzte Bewegung?
C031 BNE $C036      ; nein, dann andere Bewegung
C033 JSR $C07B      ; Sprite nach links bewegen
C036 PLA            ; gespeicherten Wert holen
C037 PHA            ; und erneut abspeichern
C038 LSR            ; Bits verschieben, um Bewegung
C039 LSR            ; in vertikaler Richtung abzufragen
C03A AND #$03       ; Bits isolieren
C03C CMP $FC        ; mit altem Wert vergleichen
C03E BEQ $C054      ; ja, dann zur nächsten Abfrage
C040 LDY $FC        ; alten Wert holen
C042 STA $FC        ; und neuen Wert speichern
C044 CMP $C0C0,Y    ; Wert für entsprechende Bewegung
C047 BNE $C04C      ; nein, dann andere Richtung
C049 JSR $C09E      ; Sprite nach unten verschieben
C04C CMP $C0C4,Y    ; Wert mit anderer Richtung vergleichen
C04F BNE $C054      ; nein, dann Feuerknopf
C051 JSR $C0AB      ; Sprite nach oben verschieben
C054 PLA            ; gespeicherten Portwert holen
C055 AND #$10       ; mit Feuertaste vergleichen
```

```
C057 BNE $C018      ; nein, dann zum Anfang der Abfrage
C059 INC $D020      ; Wert für Rahmenfarbe erhöhen
C05C JMP $C018      ; zum Anfang der Abfrage springen
C05F NOP
```

Bewegungsroutinen für das Sprite

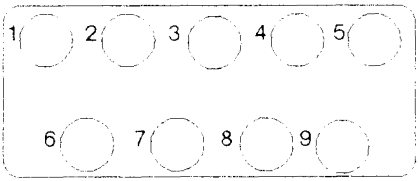
```
C060 PHA            ; Akku retten
C061 LDA $D010      ; Spriteüberlaufregister laden
C064 CMP #$01       ; ist es gesetzt
C066 BNE $C06F      ; nein, dann verzweigen
C068 LDA $D000      ; Spritekoordinate laden
C06B CMP #$4D       ; mit Maximalwert vergleichen
C06D BCS $C079      ; verzweige, wenn größer
C06F INC $D000      ; X-Koordinate erhöhen
C072 BNE $C079      ; verzweige, wenn kein Überlauf
C074 LDA #$01
C076 STA $D010      ; Überlaufregister setzen
C079 PLA            ; Gespeicherten Wert holen
C07A RTS            ; und zurück zum Hauptprogramm

C07B PHA            ; Akku retten
C07C LDA $D010      ; Überlaufregister laden
C07F CMP #$01       ; Register gesetzt
C081 BNE $C092      ; nein, dann verzweige
C083 LDA $D000      ; Koordinate laden
C086 BNE $C08D      ; wenn nicht gleich null, verzweige
C088 LDA #$00       ; Überlaufregister löschen
C08A STA $D010
C08D DEC $D000      ; X-Koordinate verringern
C090 PLA            ; gespeicherten Akku holen
C091 RTS            ; und zurück ins Hauptprogramm
C092 LDA $D000      ; Koordinate laden
C095 CMP #$18       ; Minimalwert
C097 BCC $C090      ; kleiner dann Rücksprung
C099 DEC $D000      ; sonst X-Koordinate verringern
C09C BNE $C090      ; unbedingter Sprung
```

```
C09E PHA          ; Akku retten
C09F LDA $D001    ; Y-Koordinate laden
C0A2 CMP #$F1     ; Maximalwert
C0A4 BCS $C0A9    ; größer, dann Rücksprung
C0A6 INC $D001    ; Koordinate erhöhen
C0A9 PLA          ; geretteten Akku holen
C0AA RTS          ; und zurück ins Hauptprogramm

C0AB PHA          ; Akku retten
C0AC LDA $D001    ; Y-Koordinate laden
C0AF CMP #$31     ; Minimalwert
C0B1 BCC $C0B6    ; kleiner, dann Rücksprung
C0B3 DEC $D001    ; sonst Koordinate verringern
C0B6 PLA          ; Akku holen
C0B7 RTS          ; Rücksprung ins Hauptprogramm

C0C0 02 00 03 01 01 03 00 02      ; Daten zur Feststellung
                                    der Richtung der Maus
```

Hier jetzt der BASIC-Lader, in dem das Sprite schon vorhanden ist.

```
10 FOR N=0TO63:READ X:POKE832+N,X:NEXT
20 READ X:SUM=SUM+X:IFX=-1THEN40
30 POKE49152+Y,X:Y=Y+1:GOTO20
40 IFY<>200THEN PRINT"ERROR IN DATA":END
50 SYS49152
90 REM
91 REM SPRITE DATEN
92 REM
100 DATA 0,0,0,0,0,0,0,0,0,15,224,0,15,192,0,15,128,0,15,192,0,13,
224,0
101 DATA 8,240,0,0,120,0,0,60,0,0,30,0,0,15,0,0,7,128,0,3,128,0,0,
0,0,0,0
102 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
197 REM
198 REM  PROGRAMM
199 REM
```

```
200 DATA 120,169,1,141,21,208,169,96,141,0,208,141,1,208,169,13,14
1,248,7
201 DATA 162,0,142,2,220,173,0,220,72,41,3,197,251,240,20,164,251,
133,251
202 DATA 217,192,192,208,3,32,96,192,217,196,192,208,3,32,123,192,
104,72,74
203 DATA 74,41,3,197,252,240,20,164,252,133,252,217,192,192,208,3,
32,158,192
204 DATA 217,196,192,208,3,32,171,192,104,41,16,208,191,238,32,208
,76,24,192
205 DATA 0,72,173,16,208,201,1,208,7,173,0,208,201,77,176,10,238,0
,208,208
206 DATA 5,169,1,141,16,208,104,96,72,173,16,208,201,1,208,15,173,
0,208,208
207 DATA 5,169,0,141,16,208,206,0,208,104,96,173,0,208,201,24,144,
247,206
208 DATA 0,208,208,242,72,173,1,208,201,241,176,3,238,1,208,104,96
,72,173
209 DATA 1,208,201,49,144,3,206,1,208,104,96,0,0,0,0,0,0,0,0,2,0,3
,1,1,3,0,2,-1
```



Control Port 1:

Pin	Signal
1	JOY A0
2	JOY A1
3	JOY A2
4	JOY A3
5	PADDLE POTENTIOMETER AY
6	BUTTON A/LIGHT PEN
7	+ 5V
8	GND
9	PADDLE POTENTIOMETER AX

Control Port 2:

Pin	Signal
1	JOY B0
2	JOY B1
3	JOY B2
4	JOY B3
5	PADDLE POTENTIOMETER BY
6	BUTTON B
7	+ 5V
8	GND
9	PADDLE POTENTIOMETER BX

Abb. 5.4.4: Control Port

5.4.5 Die 64 Maus 1351

Die Maus ist ein sehr leicht zu bedienendes Eingabeinstrument. Sie belegt die gleichen Anschlüsse wie auch der Joystick.

Doch wie funktioniert eigentlich die Maus? Die Bewegung der in der Maus befindlichen Kugel wird auf zwei Rollen übertragen. Die eine Rolle übernimmt die Bewegung in X- und die andere in Y-Richtung. An jeder der Rollen ist ein Kranz mit Löchern befestigt. Die Löcher lassen einen Lichtstrahl, der auf einen Fotowiderstand fällt, ungehindert durch. Durch Drehen des Kranzes entsteht ein ständiger Wechsel zwischen hell und

dunkel. Die Bewegung der Maus wird somit nicht wie beim Joystick durch das Vorhanden- und Nichtvorhandensein des Signals erkannt.

Das folgende Maschinenprogramm ermöglicht es, die Commodore-Maus 1351 in Port 1, die speziell für den C64 entwickelt wurde, abzufragen. Es liegt in $8000, von wo es nach $C000 kopiert wird. In $C000 liegt das Interrupt-Programm, das die Maus steuert. Das Mausprogramm erkennt, welches Zeichen sich unter dem Mauszeiger befindet. Außerdem lassen sich die X- und Y-Koordinaten in den Adressen $037D und $037E abfragen.

Nachdem das Maschinenprogramm in den Speicher geladen wurde, kann es mit SYS 32832 gestartet werden. Im BASIC-Loader ist dies bereits im Programm eingebunden und braucht nur mit RUN gestartet zu werden.

Nun folgt das Maschinenlisting und der BASIC-Loader:

Sprite Daten

```
8000 F8 00 00 90 00 00 B8 00
8008 00 DC 00 00 8E 00 00 07
8010 00 00 02 00 00 00 00 00
8018 00 00 00 00 00 00 00 00
8020 00 00 00 00 00 00 00 00
8028 00 00 00 00 00 00 00 00
8030 00 00 00 00 00 00 00 00
8038 00 00 00 00 00 00 00 00
```

Programmanfang

```
8040 LDY §$00     Zähler auf Null
8042 LDA $8000,Y  Spritedaten
8045 STA $0E00,Y  nach Puffer 1 kopieren
8048 INY          Zähler erhöhen
8049 CPY §$40     wenn fertig,
804B BNE $8042    dann weiter
```

```
804D LDA §$01     Sprite 1
804F STA $D015    einschalten
8052 STA $D027    und Farbe setzen
8055 LDA §$64     Sprite
8057 STA $D000    X-Richtung
805A STA $D001    Y-Richtung
805D LDA §$00     Überlaufregister
805F STA $D010    setzen
8062 LDA §$38     Zeiger auf
8064 STA $07F8    Puffer 1
8067 LDY §$00     Zähler auf Null
8069 LDA $807C,Y  Interruptroutine nach
806C STA $C000,Y  $C000 kopieren
806F INY          Zähler erhöhen
8070 CPY §$D1     wenn fertig,
8072 BNE $8069    dann weiter
8074 LDA §$C0     Wert
8076 STA $0A04    zwischenspeichern
8079 JMP $C000    Sprung zur Interruproutine
807C NOP
807D NOP
807E NOP
807F NOP
8080 NOP
8081 NOP
8082 NOP
8083 NOP
8084 NOP
8085 NOP
8086 NOP
8087 NOP
8088 NOP
8089 NOP
808A NOP
808B NOP
808C NOP
808D NOP
808E NOP
808F SEI          Interrupt sperren
8090 LDA §$21     IRQ auf
```

```
8092 STA $0314   $C021
8095 LDA §$C0    setzen
8097 STA $0315   und
809A PLP         Register holen
809B CLI         Interrupt freigeben
809C RTS         Rücksprung
809D NOP         keine Operation
809E LDA $D419   Mauskoordinate holen
80A1 LDY $C0D1
80A4 JSR $C058
80A7 STY $C0D1
80AA CLC         Carry für Addition löschen
80AB ADC $D000   zur alten Position addieren
80AE STA $D000   und speichern
80B1 TXA         X-Reg. nach Akku
80B2 ADC §$00    auf
80B4 AND §$01    Überlauf
80B6 EOR $D010   prüfen
80B9 STA $D010   und speichern
80BC LDA $D41A   Mauskoordinaten holen
80BF LDY $C0D2
80C2 JSR $C058
80C5 STY $C0D2
80C8 SEC         Carry für Addition
80C9 EOR §$FF    Bits umdrehen
80CB ADC $D001   Zur alten Position addieren
80CE STA $D001   und speichern
80D1 JMP $C083   Rücksprung
80D4 STY $C0D4   Werte
80D7 STA $C0D3   zwischenspeichern
80DA LDX §$00    X-Reg. löschen
80DC SEC         Carry für subtraktion löschen
80DD SBC $C0D4   subtrahieren
80E0 AND §$7F    Bit 8 löschen
80E2 CMP §$40    wenn werte gleich,
80E4 BCS $80ED   dann verzweige
80E6 LSR         mal 2
80E7 BEQ $80FB   verzweige, wenn kein überlauf
80E9 LDY $C0D3   alten wert laden
80EC RTS         Rücksprung
```

```
80ED ORA §$C0    Bits setzen
80EF CMP §$FF    und vergleichen
80F1 BEQ $80FB   verzweige, wenn kein überlauf
80F3 SEC         Carry für Rechenoperation
80F4 ROR         Bits verschieben
80F5 LDX §$FF    alte werte
80F7 LDY $C0D3   holen
80FA RTS         Rücksprung
80FB LDA §$00    Akku löschen
80FD RTS         Rücksprung
80FE BRK
80FF LDA $D000   X-Position laden
8102 SBC §$18    und subtrahieren
8104 LDX §$FF    wert laden
8106 INX         und erhöhen
8107 SBC §$08    solange subtrahieren
8109 BCS $8106   bis Überlauf
810B TXA         X-Reg. wieder nach Akku
810C SBC §$01    subtrahieren
810E STX $03FD   und speichern
8111 SEC         Carry setzen
8112 LDA $D001   Y-Position laden
8115 SBC §$32    und subtrahieren
8117 LDX §$FF    Wert laden
8119 INX         und erhöhen
811A SBC §$08    solange subtrahieren
811C BCS $8119   bis Überlauf
811E TXA         X-Reg. nach Akku
811F SBC §$01    subtrahieren
8121 STX $03FE   und speichern
8124 LDA $D010   Prüfen ob
8127 CMP §$01    überlauf
8129 BNE $8148   verzweige, wenn ja
812B LDA $03FD   Puffer
812E CMP §$1D    richtig ?
8130 BEQ $8148   verzweige, wenn nein
8132 LDA $03FD   Puffer
8135 CMP §$1E    richtig ?
8137 BEQ $8148   verzweige, wenn nein
8139 LDA $03FD   Puffer
```

```
813C CMP §$1F    richtig ?
813E BEQ $8148   verzweige, wenn nein
8140 LDA $03FD   Puffer laden
8143 ADC §$20    richtig stellen
8145 STA $03FD   und speichern
8148 JMP $EA31   Sprung zur Interruptroutine
```

```
100 FORI=32768TO33100:READA:POKEI,A:NEXT
105 SYS 32832
110 DATA248,0,0,144,0,0,184,0,0,220,0,0,142,0,0,7,0,0
120 DATA2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
130 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
140 DATA0,0,0,0,0,0,0,0,0,0,160,0,185,0,128,153,0,14
150 DATA200,192,64,208,245,169,1,141, 21,208,141, 39,208,169,100,141,
0,208
160 DATA141,1,208,169,0,141,16,208,169,56,141,248,7,160,0,185,124,128
170 DATA153,0,192,200,192,209,208,245,169,192,141,4,10,76,0,192,234,234
180 DATA234,234,234,234,234,234,234,234,234,234,234,234,234,234,234,
234,120
190 DATA169,33,141,20,3,169,192,141,21,3,40,88,96,234,173,25,212,172
200 DATA209,192,32,88,192,140,209,192,24,109,0,208,141,0,208,138,105,0
210 DATA41,1,77,16,208,141,16,208,173,26,212,172,210,192,32, 88,192,140
220 DATA210,192,56,73,255,109,1,208,141,1,208,76,131,192,140,212,192,141
230 DATA211,192,162,0,56,237,212,192,41,127,201,64,176,7,74,240,18,172
240 DATA211,192,96,9,192,201,255,240,8,56,106,162,255,172,211,192,96,169
250 DATA0,96,0,173,0,208,233,24,162,255,232,233,8,176,251,138,233,1
260 DATA142,253,3,56,173,1,208,233,50,162,255,232,233,8,176,251,138,233
270 DATA1,142,254,3,173,16,208,201,1,208,29,173,253,3,201,29,240,22
280 DATA173,253,3,201,30,240,15,173,253,3,201,31,240,8,173,253,3,105
290 DATA32,141,253,3,76,49,234,0,0
```

5.5 Timer

Jede der zwei CIAs verfügt über zwei frei programmierbare 16-Bit-Timer. Diese zählen von ihrem gesetzten Wert an bis Null und lösen daraufhin einen Interrupt aus. Das Auslösen eines solchen Interrupts kann jedoch auch vom Programmierer verhindert werden. Normalerweise wird Timer A von CIA 1 vom

Betriebssystem dazu verwendet, den jede 1/60 Sekunde aufgerufenen Systeminterrupt zu steuern. Aus diesem Grunde ist es ratsam, nur den unbenutzten Timer B für eigene Experimente zu verwenden.

Man stellt den Timer B, indem man den Wert, von dem heruntergezählt werden soll, in die Register 6 und 7 des CIA 1 (Adressen $DC06 und $DC07) schreibt. Register 6 enthält das Low-Byte und Register 7 das High-Byte des Wertes. Für Timer A der CIA 1 sind es die Register 4 und 5 (Adrssen $DC04 und $DC05). Jeder der beiden Timer besteht aus einem Vorspeicher (engl. Latch) und einem Zähler (Counter). Schreibt man einen Wert in eines dieser Register, so wird er im Latch gespeichert. Beim Lesen der Register erhält man den derzeitigen Wert des Timers. Beide Timer können unabhängig voneinander betrieben werden. Es besteht jedoch auch die Möglichkeit, sie zu kombinieren, und dadurch einen 32-Bit-Timer zu erhalten. Dies kann dann von Vorteil sein, falls lange Verzögerungsschleifen benötigt werden.

Zum Einstellen des Arbeitsmodus existieren zwei voneinander unabhängige Register. Für Timer A ist dieses Register 14 (Adresse $DC0E) und für Timer B 15 (Adresse $DC0F). Auf die Bedeutung der einzelnen Bits dieser Register, die im wesentlichen gleich ist, wird jetzt näher eingegangen.

Bit 0: START/STOP

Es ist möglich, durch Löschen dieses Bits die Timer anzuhalten. Durch Setzen des Bits werden sie wieder gestartet.

Bit 1: PB ON/OF

Ist dieses Bit gesetzt, so wird bei jedem Unterlauf (das heißt beim Durchlaufen von Null) des Timers ein Signal an die entsprechende Portleitung ausgegeben. Für Timer A ist das PB 6 und für Timer B PB 7. Ist das Bit gelöscht, so hat ein Unterlauf des Timers keinen Einfluß auf die Portleitung. Dieses Bit hat

Priorität vor dem DDRB. Das bedeutet, daß bei gesetztem Bit die entsprechende Portleitung immer als Ausgang fungiert.

Bit 2: TOGGLE/PULSE

Dieses Bit arbeitet nur im Zusammenhang mit Bit 1 dieses Registers. Ist es gesetzt, so wird bei jedem Timerunterlauf das Signal auf Leitung PB6 invertiert. Beim Register 15 ist es Leitung PB7. Es ist also möglich, ein Rechtecksignal auszugeben, dessen Frequenz durch den entsprechenden Timer festgelegt ist. Bei gesetztem Bit wird bei jedem Unterlauf auf die entsprechende Portleitung ein positives Signal in Länge eines Taktzyklus ausgegeben.

Bit 3: ONE-SHOT/CONTINUOUS

Im One-Shot-Modus (in diesem Fall ist das Bit gesetzt) zählt der Timer von dem im Latch gespeicherten Wert herunter, erzeugt einen Interrupt, lädt den Timer erneut mit dem gespeicherten Wert und stoppt daraufhin. Ist das Bit nicht gesetzt (Continuous-Modus) stoppt der Timer nicht, sondern beginnt erneut den Wert abzuzählen.

Bit 4: FORCE LOAD

Wird dieses Bit gesetzt, so wird der Counter des Timers, unabhängig davon, ob er gerade läuft oder nicht, mit dem Wert des Latch geladen.
Die folgenden Bits sind bei Register 14 und 15 unterschiedlich und werden deshalb einzeln behandelt.

Bit 5 (Register 14): IN MODE

Dises Bit bestimmt die Quelle des Timer-Triggers (Trigger = Auslöser), also das Signal, das den Timer dazu veranlaßt, seinen Wert um eins zu verringern. Bei gesetztem Bit wird der Trigger durch positive Signale an der Leitung CNT ausgelöst. Ist das Bit gelöscht, so ist der Systemtakt der Auslöser.

Bit 5 und 6 (Register 15): IN MODE

Diese Bits erfüllen dieselbe Aufgabe wie in Register 14. Allerdings können sie (es sind ja zwei Bits), mehr Quellen angeben.

00=Trigger wird durch Systemtakt ausgelöst.
01=Trigger wird durch positive Signale am Pin CNT ausgelöst.
10=Trigger wird durch den Unterlauf von Timer A ausgelöst.
11=Wie 10, jedoch wird Timer A durch das Signal an CNT ausgelöst

5.6 Echtzeituhr

Jeder CIA verfügt über eine 24 Echtzeituhr, die allerdings vom Computer nicht benutzt wird. Sie entspricht nicht der Betriebssystemuhr TI$, die über den Systeminterrupt gesteuert wird. Im Gegensatz zur CIA-Uhr ist die TI$-Uhr sehr ungenau. Ihre Abweichung beträgt ca. 1/2-Stunde pro Tag, da sie von der Häufigkeit des Interrupts abhängt.

Bei der Uhr des CIA besteht die Möglichkeit, eine Alarmzeit festzulegen. Beim Erreichen dieser Zeit, wird ein Interrupt ausgelöst. Die Uhr ist in vier Register aufgeteilt. Es sind die Register 8 bis 11, die folgende Aufgaben haben:

In Register 8 sind die Zehntelsekunden gespeichert, in Register 9 sind es die Sekunden, in Register 10 die Minuten und in Register 11 die Stunden. Zusätzlich gibt Bit 7 des Registers 11 noch AM oder PM (Vor- oder Nachmittag) an. Die Zahlendarstellung in all diesen Registern erfolgt im BCD-Format, wodurch das Umwandeln durch Maschinenprogramme erleichtert wird. Nähere Erklärungen zu diesen Format und eine Tabelle zur Umwandlung der beiden Formate finden Sie am Ende dieses Abschnitts 5.6.

Die Uhr des CIA ist über die Netzfrequenz getaktet und deshalb recht genau. Es besteht die Möglichkeit, die Uhr an die jeweilige Netzfrequenz anzupassen. Dies kann durch Ändern von Bit

7 des Registers 14 geschehen. Ist es gesetzt, so arbeitet die Uhr mit der bei uns verwendeten Netzfrequenz von 50 Hertz, ansonsten sind es 60 Hertz.

Die Alarmzeit wird in denselben Registern angegeben, die zum Stellen der Uhr benutzt werden. Der einzige Unterschied besteht darin, daß beim Stellen der Alarmzeit Bit 7 des Registers 15 gesetzt ist. Beim Erreichen der Alarmzeit wird ein NMI ausgelöst. Bei einem Lesezugriff erhält man immer den Wert des Counters. Es ist also nicht möglich, die Alarmzeit abzufragen.

Soll die Uhr gestellt oder gelesen weden, müssen ein paar Punkte beachtet werden. Wenn die Uhr gestellt wird, so ist es ratsam, das Register, das für die Stunden zuständig ist, als erstes zu beschreiben. Sobald ein Schreibzugriff auf dieses Register erfolgt, bleibt die Uhr stehen und beginnt erst dann wieder zu laufen, wenn das Register, das für die Zehntelsekunden zuständig ist, beschrieben wurde. Dies ist sicher sinnvoll, denn wer kann schon eine laufende Uhr auf Zehntelsekunden genau einstellen. Die Uhr stoppt allerdings nicht, wenn ein Zugriff auf eines der anderen Register erfolgte.

Ähnlich wie beim Stellen verhält es sich auch beim Lesen der Uhrzeit. Erfolgt ein Lesezugriff auf das Stundenregister, so wird die aktuelle Zeit im Latch gespeichert. Die Uhrzeit in den Registern ändert sich nicht, die Zeit läuft jedoch intern (im Latch) weiter. Somit ist es möglich, die genaue Zeit zum Zeitpunkt des Lesezugriffs zu erhalten. Sobald die Zehntelsekunden gelesen wurden, wird die gelatchte Zeit wieder in die Register übertragen. Wenn nur das Stundenregister gelesen werden soll, so ist es unerläßlich, die Zehntelsekunden auch zu lesen, damit die Uhr weiterläuft.

Bevor wir Ihnen anhand eines Beispiels zeigen, wie das Stellen und das Lesen der Uhr geschieht, wird das BCD-Format genauer erklärt. Wie Sie vielleicht wissen, bietet der 6510-Prozessor die Möglichkeit, in diesem Format zu rechnen. Dazu muß lediglich das Dezimalflag des Prozessors im Statusregister mit dem Befehl SED gesetzt werden.

Wie der Name Dezimalflag schon sagt, können wir jetzt Zahlen im Dezimalsystem darstellen und mit ihnen rechnen. In diesem Modus kann man mit einem Byte nur Zahlen von 0 bis 99 darstellen. Ein Byte wird zu diesem Zweck in LOW- und HIGH-Nibble aufgespalten. Mit jedem der Nibble kann eine Zahl zwischen 0 und 9 dargestellt werden. Jedes Nibble stellt also eine Ziffer der Dezimalzahl dar. Hier nun eine Tabelle, aus der Sie die entsprechenden Werte entnehmen können.

DEZIMAL	BCD-FORMAT
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Wenn die Zahl 35 im BCD-Format dargestellt werden soll, so ergibt sich die folgende Bitfolge: 00110101. Es folgen nun zwei Programme, die es ermöglichen, die Zeit einzustellen und auch wieder abzurufen.

Zunächst das Programm zum Stellen. Der Wert für 1/10sec wird bei diesem Programm immer auf 0 gesetzt.

```
10 C=56328: REM Basisadr. der Uhr in CIA 1
20 REM C=56584 für Uhr in CIA 2
30 POKE C+7,PEEK(C+7)AND127
35 POKE C+6,PEEK(C+6)OR128
40 INPUT"ZEIT IM FORMAT HHMMSS EINGEBEN";A$
50 IF LEN(A$)<>6 THEN 40
60 H=VAL(LEFT$(A$,2))
```

```
70 M=VAL(MID$(A$,3,2))
80 S=VAL(RIGHT$(A$,2))
90 IF H>23 THEN 40
100 IF H>11 THEN H=H+68
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10
120 IF M>59 THEN 40
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10
140 IF S>59 THEN 40
150 POKE C+1,16*INT(S/10)+S-INT(S/10)*10
160 POKE C,0
```

Das Lesen der Uhrzeit ermöglicht folgendes Programm:

```
10 C=56328: REM Basisadr. der Uhr in CIA 1
20 PRINT CHR$(147):REM C=56584 für Uhr in CIA 2
30 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=PEEK(C)
40 FL=1
50 IF H>32 THEN H=H-128:FL=0
60 H=INT(H/16)*10+H-INT(H/16)*16:ON FL GOTO 80
65 IF H=12 THEN 85
70 H=H+12
80 IF H=12 THEN H=0
85 M=INT(M/16)*10+M-INT(M/16)*16
90 S=INT(S/16)*10+S-INT(S/16)*16
100 T$=STR$(T)
110 H$=STR$(H):IF LEN(H$)=2 THEN H$=" 0"+RIGHT$(H$,1)
120 M$=STR$(M):IF LEN(M$)=2 THEN M$=" 0"+RIGHT$(M$,1)
130 S$=STR$(S):IF LEN(S$)=2 THEN S$=" 0"+RIGHT$(S$,1)
140 PRINT CHR$(19);
150 PRINT RIGHT$(H$,2)":"RIGHT$(M$,2)":"RIGHT$(S$,2)":0";
160 PRINT RIGHT$(T$,1)
170 GOTO 30
```

Nach Drücken von STOP/RESTORE müssen Sie die Uhrzeit wieder neu setzen, da das Betriebssystem alle Register auf den Ausgangswert setzt. Davon ist leider auch das Bit für die 50/60Hz-Auswahl betroffen. Ihre Uhr würde stark zurückbleiben.

5.7 Die Programmierung der RS-232 Schnittstelle

RS-232 ist die Bezeichnung für eine Schnittstelle zur seriellen Datenübertragung. Auch die europäische Bezeichnung V 24 ist gebräuchlich. Was bedeutet nun serielle Übertragung und wann wird sie benutzt?

Bei der seriellen Übertragung werden nicht wie bei der parallelen Schnittstelle jeweils 8 Bits auf verschiedenen Leitungen gleichzeitig, sondern Bit für Bit nacheinander übertragen. Daraus ergeben sich Vor- und Nachteile der verschiedenen Übertragungsweisen. Die serielle Schnittstelle kommt mit weniger Leitungen aus; außerdem ist Datenübertragung über eine Telefonleitung möglich, dafür geht es jedoch nicht so schnell wie bei der parallelen Übertragung, die über mehr Leitungen verfügt.

Das Betriebssystem des Commodore 64 enthält bereits die komplette Software zur Bedienung einer seriellen RS-232 Schnittstelle. Die Schnittstelle selbst ist als RS-232 Steckmodul erhältlich, das auf den USER-Port gesetzt wird und die erfoderliche Pegelumwandlung auf +/- 12 Volt übernimmt. Dadurch können Sie mit Ihrem Commodore 64 auch mit Geräten mit serieller Schnittstelle kommunizieren.

Das Betriebssystem hat der RS-232 Schnittstelle die Geräteadresse 2 zugeordnet. Wird ein logisches File mit Gerätenummer 2 eröffnet, so legt das Betriebssystem zwei Puffer zu je 256 Byte als Ein- und Ausgabepuffer für die zu übertragenden Daten an.

Dieser Pufferbereich liegt normalerweise am Ende des BASIC-RAMs. Wird die RS-232 Schnittstelle in einem BASIC-Programm verwendet, sollte der OPEN-Befehl zuerst gegeben werden, da dabei alle Variablen gelöscht werden. Auch wird nicht geprüft, ob noch ausreichend Speicher vorhanden ist. Beim CLOSE-Befehl wird dieser Puffer wieder freigegeben. Auch dabei wird in BASIC ein CLR-Befehl ausgeführt, so daß Sie die Datei erst am Ende Ihres Programms schließen sollten. Zu einer Zeit kann immer nur ein Datenkanal für RS-232 offen sein.

Beim Schließen eines RS-232 Datenkanals wird eine eventuell laufende Übertragung abgebrochen und die Pufferzeiger zurückgesetzt. Der Befehl

```
SYS 61604
```

wartet solange, bis der komplette Pufferinhalt übertragen ist und sollte vor dem CLOSE-Befehl benutzt werden (Maschinenspracheprogrammierer können JSR $F0A4 benutzen).

Die Parameter für die Datenübertragung werden durch ein Kontrollregister und ein Befehlsregister festgelegt. Diese beiden Register werden als die ersten beiden Zeichen des 'Filenamens' übergeben.

Das Kontrollregister dient zur Definition der Baud-Rate sowie der Anzahl der zu übertragenden Daten- und Stopbits. Die Baud-Rate bestimmt die Geschwindigkeit der Datenübertragung in Bits pro Sekunde, die Stopbits werden nach jedem übertragenen Datenwort (5-8 Bits) gesandt.

Das Befehlsregister bestimmt Übertragungsart, Paritätsprüfung und Art des Handshake.

Beim Kontrollregister bestimmen die untersten 4 Bits die Baud-Rate nach folgender Tabelle:

Bit	3	2	1	0	dezimal	Baud-Rate
	0	0	0	0	0	Anwender-Baud-Rate, s.u.
	0	0	0	1	1	50
	0	0	1	0	2	75
	0	0	1	1	3	110
	0	1	0	0	4	134.5
	0	1	0	1	5	150
	0	1	1	0	6	300
	0	1	1	1	7	600
	1	0	0	0	8	1200
	1	0	0	1	9	1800
	1	0	1	0	10	2400

1	0	1	1	11	3600	(nicht implementiert)
1	1	0	0	12	4800	(nicht implementiert)
1	1	0	1	13	7200	(nicht implementiert)
1	1	1	0	14	9600	(nicht implementiert)
1	1	1	1	15	19200	(nicht implementiert)

Sie können also Baud-Raten zwischen 50 und 2400 programmieren. Die Anzahl der Datenbits wird durch Bit 5 und 6 bestimmt:

Bit 6	5	dezimal	Anzahl der Datenbits
0	0	0	8 Bits
0	1	32	7 Bits
1	0	64	6 Bits
1	1	96	5 Bits

Die Anzahl der Stopbits schließlich wird durch Bit 7 bestimmt:

Bit 7	dezimal	Anzahl der Stopbits
0	0	1 Stopbit
1	128	2 Stopbits

Das Befehlsregister ist folgendermaßen organisiert:

Bit 0	dezimal	Handshake
0	0	3-Draht-Handshake
1	1	X-Draht-Handshake

Bit 4	dezimal	Übertragungsart
0	0	Vollduplex
1	16	Halbduplex

Bit 7	6	5	dezimal	Paritätsprüfung
X	X	0	0	keine Paritätsprüfung, kein 8. Datenbit
0	0	1	32	ungerade Parität
0	1	1	96	gerade Parität
1	0	1	160	keine Paritätsprüfung, 8. Datenbit immer 1
1	1	1	224	keine Paritätsprüfung, 8. Datenbit immer 0

Hier noch eine Bemerkung zum Handshake-Modus: Haben Sie '3 Draht Handshake' gewählt, so werden die Steuerleitungen CTS (Clear To Send) und DSR (Data Set Ready) beim Senden und Empfangen nicht ausgewertet, das heißt, der Rechner sendet die Daten z.B. an einen Drucker unabhängig davon, ob der Drucker die Daten schon verarbeitet hat oder nicht. Wollen Sie dagegen, daß das angeschlossene Gerät die Möglichkeit hat, die Datenübertragung anzuhalten, so müssen Sie 'X Draht Handshake' wählen. Dazu müssen die oben erwähnten Steuerleitungen verdrahtet werden und der Drucker natürlich in der Lage sein, diese Steuerleitungen überhaupt bedienen zu können. Wenn man dagegen Daten zwischen zwei Rechnern austauscht, kommt man oft mit dem '3 Draht Handshake' aus. Beachten Sie dazu auch die Hinweise weiter unten.
Sie wollen nun einen RS-232 Datenkanal mit folgenden Parametern eröffnen:

Übertragungsrate 2400 Baud
7 Bit ASCII Daten
2 Stopbits
keine Paritätsprüfung
8. Datenbit immer 0
Vollduplex
3-Draht-Handshake

Die OPEN-Anweisung sieht dann so aus:

```
OPEN 1, 2, 0, CHR$(10+0+128)+CHR$(0+0+224)
```

Die Programmierung eigener Baud-Raten

Da die Realisierung der verschiedenen Baud-Raten über die Programmierung der Timer in den CIAs geschieht, können Sie bei Bedarf andere als die vorgegebenen Baud-Raten verwenden. Die obere Grenze von 2400 Baud läßt sich dabei jedoch nicht überschreiten, da die softwaremäßige Realisierung der RS 232 Übertragung dafür nicht schnell genug ist. Die Timer lösen nach einer Zeit, die abhängig von der Baud-Rate ist, einen nicht maskierbaren Interrupt (NMI) aus, während dessen der Empfang der einzelnen Bits stattfindet. Wollen Sie nun eigene Baud-Raten verwenden, so können Sie die entsprechenden Timerwerte als drittes und viertes Zeichen des Filenamens beim OPEN-Befehl übergeben. Die Timerwerte lassen sich nach folgender Formel aus der Baud-Rate ermitteln:

$$T = 492662/BAUD - 101$$

Den erhaltenen Wert müssen wir in Low- und High-Byte zerlegen und als drittes und viertes Zeichen des Filenamens übergeben. Ins Kontrollregister müssen wir anstelle des Kodes für die Baud-Rate eine Null setzen (Anwender Baud-Rate), damit das Betriebssystem weiß, daß wir eine eigene Baud-Rate verwenden wollen. Das folgende Beispiel benutzt die gleichen Parameter wie oben, verwendet jedoch eine Baud-Rate von 1000.

```
100 BAUD = 1000
110 T = 492662/BAUD - 101
120 TH = INT(T/256) : TL = T - TH*256
130 OPEN 1,2,0, CHR$(128)+CHR$(224)+CHR$(TL)+CHR$(TH)
```

Bei eigenen Baud-Raten lassen sich Werte von ca. 8 bis 2400 Baud erreichen.

Das Statusregister beim Verkehr über die RS-232 Schnittstelle hat eine andere Bedeutung als in der normalen Datenübertragung. Es läßt sich in BASIC zwar auch über die Variable ST abfragen, wird jedoch bei jedem Lesen gelöscht. Soll der

Statuswert daher für mehrere Abfragen benutzt werden, muß er erst einer anderen Variablen zugeordnet werden. ST gibt nur den RS-232 Status wieder, wenn der letzte Datenverkehr über RS-232 lief. Von einem Maschinenprogramm läßt sich der Status jedoch auch ohne Löschen lesen. Die Bedeutung der einzelnen Bits des RS-232 Status ist im folgenden beschrieben. Ein gesetztes Bit bedeutet dabei, daß die Bedingung eingetreten ist.

Bit	Beschreibung
0	Paritätsfehler
1	Rahmenfehler
2	Empfängerpuffer voll
3	Empfängerpuffer leer
4	CTS (Clear to send) Signal fehlt
5	unbenutzt
6	DSR (Data set ready) Signal fehlt
7	Break Signal empfangen

Geschieht die Programmierung der RS-232 Schnittstelle in Maschinensprache, so kann man die Ein- und Ausgabepuffer für die Datenübertragung in einen beliebigen Speicherbreich legen. Die Zeiger auf die Puffer werden beim OPEN-Befehl einmal gesetzt und können danach jedoch in einen anderen Speicherbereich gelegt werden. Die entsprechenden Zeiger liegen in der Zeropage, und zwar zeigt $F7/$F8 auf den Eingabepuffer und $F9/$FA auf den Ausgabepuffer. Die Programmierung der Ein-/Ausgabe auf die RS-232 Schnittstelle geschieht genauso wie bei anderen Ausgabegeräten, als Geräteadresse wird 2 gewählt. Siehe dazu das Kapitel über Ein-/Ausgabeprogrammierung.

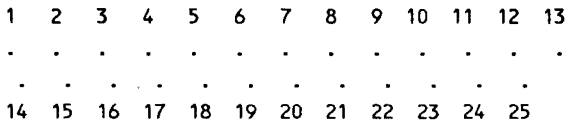
Dazu noch einige Adressen für die RS-232 Ein-/Ausgabe

```
$0293        659      Kontrollwort
$0294        660      Befehlswort
$0295/$0296 661/662 Timerwert für Baud-Rate
$0297        663      RS-232 Statuswort
$0298        664      Anzahl der Datenbits, wird bei OPEN berechnet
$0299/$029A 665/666 Timerwert für Baud-Rate beim Senden
$029B        667      Schreibzeiger Empfangspuffer
```

$029C	668	Lesezeiger Empfangspuffer
$029D	669	Lesezeiger Sendepuffers
$029E	670	Schreibzeiger Sendepuffer

Weiterhin werden in der Zero-Page noch einige Adressen als Arbeitsspeicher während der Datenübertragung benötigt.
Der physikalische Anschluß an die RS-232 Schnittstelle

Als Normstecker für die RS-232 Schnittstelle dient ein 25poliger Cannonstecker, der folgendermaßen aufgebaut und belegt ist:



Pinbelegung des 25poligen Cannonsteckers

Pin	Abk.	Richtung	Bedeutung
1	GND		Protective Ground
2	TXD	OUT	Transmited Data
3	RXD	IN	Received Data
4	RTS	OUT	Request To Send
5	CTS	IN	Clear To Send
6	DSR	IN	Data Set Ready
7	GND		Signal Ground
8	DCD	IN	Data Carrier Detected
20	DTR	OUT	Data Terminal Ready
22	RI	IN	Ring Indicator

Wollen Sie zwei Rechner untereinander verbinden, so muß die Leitung folgendermaßen aussehen:

Rechner 1 Beschreibung	Pin	Pin	Rechner 2 Beschreibung
Transmitted Data	2 ---->	3	Received Data
Received Data	3 <----	2	Transmitted Data
Ready To Send	4 ---->	5	Clear To Send
Clear To Send	5 <----	4	Ready To Send
Data Set Ready	6 <---	20	Data Terminal Ready
Data Terminal Ready	20 ---->	6	Data Set Ready
Ground	7 <--->	7	Ground

Die Pfeile geben jeweils die Richtung der Übertragung an. Bei dieser Verdrahtung können Sie den 'X-Line-Handshake' wählen. Wenn Sie im 3-Draht-Betrieb arbeiten, so genügt es, wenn Sie Pin 2, 3 und 7 wie oben beschrieben verbunden haben. Wollen Sie z.B. einen Drucker anschließen, dessen Übertragungsparameter Sie nicht kennen, so kann man folgenden Trick anwenden:

Verbinden Sie wie oben die Pins 2, 3 und 7 und verbinden Sie an jedem Stecker die Pins 4, 5 und 8 sowie die Pins 6 und 20. Damit können Sie die meistens Geräte zum Arbeiten veranlassen.

Sie werden jetzt Daten empfangen oder senden können. Falls die übertragenen Daten verstümmelt oder verkehrt ankommen, so wird die Baud-Rate und/oder die Parity nicht stimmen. Variieren Sie diese Parameter nun so lange, bis die Daten richtig übermittelt werden. Durch das Kurzschließen der Steuerleitungen arbeiten Sie praktisch im 3-Line-Betrieb. Falls Sie die Daten nicht schnell genug senden oder empfangen können, müssen Sie evtl. auf Maschinensprache ausweichen.

5.8 Der IEC-BUS

Ganz algemein ist der IEC-Bus eine Schnitstelle zwischen Computer und den Peripheriegeräten. Man unterscheidet zwischen dem 'großen' und den 'seriellen' IEC-Bus. Beim 'großen' IEC-Bus werden die Daten zwischen den Geräten paralel (alle Bits auf einmal) übertragen. Im Gegensatz dazu werden die Daten beim 'seriellen' IEC-Bus, wie der Name schon sagt seriell (alls Bits nacheinander) übertragen. Der 'serielle' kommt aufgrund

dessen mit fünf Leitungen aus, wärend es beim 'großen' 24 Leitungen gibt. Der Unterschied liegt natürlich nicht nur in der Anzahl der Leitungen, sondern auch in der Geschwindigkeit des Datentransfers. Der C64 verfügt nur über den 'seriellen' IEC-Bus, und deswegen meinen wir in Zukunft wenn wir vom IEC-Bus sprechen, den eben genannten.

Doch nun wollen wir uns einmal die Arbeitsweise dieses IEC-Busses näher ansehen.

Übrigens, wieso wird eigentlich immer von 'Bus' geredet und was hat das mit dem bekannten Nahverkehrsmittel zu tun?

Sie werden es kaum glauben, aber die beiden Dinge haben sehr viel gemein.

Stellen Sie sich eine lange Straße mit mehreren Bushaltestellen vor. Die Haltestellen sollen die an der Leitung angeschlossenen Geräte sein. Auf dieser Straße fährt nun ein richtiger Bus. An den Haltestellen stoppt er entweder dann, wenn dort Leute stehen, die mitgenommen werden wollen, oder wenn im Bus befindliche Personen auszusteigen wünschen. Kurz gesagt ist der Bus ein Transportmittel, mit dem Leute von einer Haltestelle zu irgendeiner anderen befördert werden. Wo ein- oder ausgestiegen wird, bestimmen die Fahrgäste selbst, denn woher sollte der Busfahrer das wissen. Dieser ist nur für den ordnungsgemäßen Transport und die Einhaltung der Verkehrsregeln zuständig.

Das obige trifft auch ziemlich genau den Sachverhalt beim IEC-Bus. Der Bus-CONTROLLER ist der Fahrer (von dem es natürlich nur jeweils einen gibt); Haltestellen, bei denen sich der Bus füllt, werden TALKER genannt, und die Stellen, bei denen Fahrgäste den Bus verlassen, sind die LISTENER.

In der Microcomputertechnik wird also ganz allgemein unter Bus ein Leitungssystem verstanden, welches in der Regel von einem Controller gesteuert wird, und an dem mehrere Einheiten gleichberechtigt angeschlossen sind.

5.8.1 Begriffsbestimmungen

Damit wir in den folgenden Abschnitten nicht beim Auftauchen von neuen Begriffen den Fluß des Textes durch Erläuterungen aufhalten müssen, wollen wir Ihnen an dieser Stelle die häufig verwendeten Bezeichnungen vorstellen und ausführlich erklären. Die Kenntnis dieser Begriffe lohnt sich auf jeden Fall, da diese auch in anderweitiger Fachliteratur sehr verbreitet sind.

EOI = End Or Identify

Dies ist ein Signal, das zwei Zwecken dient. Erstens sendet der TALKER dieses Signal bei der Übertragung des letzten Datenbytes, zweitens benutzt es der CONTROLER in Verbindung mit ATN, um ein Gerät, welches mit SRQ eine Bedingung verlangt, zum Senden einer Geräteadresse zu veranlassen.

Hier noch zusätzlich die Belegung der fünf Leitungen des IEC-Busses:

1 SRQ = Service ReQuest

Hat ein Gerät irgendeine Aufgabe erledigt und braucht neue Daten oder hat welche abzugeben, so kann es dem CONTROLER durch dieses Signal mitgeteilt werden. Dieser wird daraufhinn einen Identify-Zyklus (mit EOI und ATN) einleiten, um festzustellen, um welches Gerät es sich handelt. Diese Funktionsweise wird beim C64 nicht verwendet.

2 GND = Masse

3 ATN = ATteNtion

Immer, wenn der CONTROLLER einen Befehl übermitteln will, aktiviert er diese Leitung. Dadurch soll erreicht werden, daß alle am Bus angeschlossenen Geräte diesen Befehl mitbekommen, da ja von vornherein noch nicht feststeht, welches Gerät gemeint ist. Das stellt sich erst bei Übermittlung der Geräteadresse heraus, weshalb diese auch immer zuerst übermittelt wird, damit sich die anderen Geräte wieder vom Bus 'abhängen' können.

4 CLK = CLOCK

Da die Daten bitseriell übertragen werden, gibt der TALKER jedem Bit einen Taktimpuls auf die Leitung CLK mit auf den Weg, womit die Gültigkeit der Datenleitung angezeigt wird.

5 DATA

Ist die einzige Datenleitung, über die ein Datenbyte mit dem niedrigstwertigen Bit voran seriell geschoben wird.

6 Reset

Diese Leitung hat für die Vorgänge auf dem Bus keine Bedeutung. Sie dient lediglich dazu, den an einem beliebigen Gerät aufgetretenen Resetimpuls weiterzuleiten.

Serielle E / A :

Pin	Signal
1	SRQIN
2	GND
3	ATN OUT
4	CLK IN / OUT
5	DATA IN / OUT
6	$\overline{\text{RESET}}$

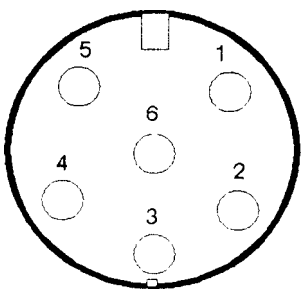


Abb. 5.8.1.1: Serielle E/A

5.8.2 Geräteadressen

Aus der Tatsache, daß alle Geräte gleichberechtigt am IEC-Bus angeschlossen sind, ergibt sich die Notwendigkeit, ein bestimmtes Gerät für eine bestimmte Aktion zu selektieren, an der die anderen Geräte unbeteiligt bleiben sollen.

Zu diesem Zweck ist jedem Gerät eine "Hausnummer" zugeteilt, die eben besagte Geräteadresse. Diese Adresse enthält aber nicht nur die Hausnummer, sondern auch schon die Aktion, die vom Gerät verlangt wird, nämlich ob das Gerät die Daten senden oder empfangen oder die Aktivitäten beenden soll.

Dazu wird das Adressierungsbyte in zwei Hälften geteilt. Das niederwertige Halbbyte enthält nur die Adresse (deshalb ein Bereich von 0-15), das höherwertige Halbbyte die Aktion (ein Bereich von 16-240).

Die möglichen Aktionen:

$20 (32) Das Gerät wird als LISTENER adressiert, das heißt, es soll Daten empfangen. Dieses Kommando resultiert z.B. aus einem PRINT#-Befehl in BASIC.

$40 (64) Das Gerät soll TALKER sein, also Daten senden. Die Ursache dieses Kommandos kann ein GET# oder INPUT# sein.

$30 (48) Die Betriebsart LISTEN wird beendet. Das niederwertige Halbbyte ist hierbei immer =15.

$50 (80) Die Betriebsart TALK wird beendet. Auch hierbei ist der Adressteil =15.

Das vollständige Adressierungsbyte für beispielsweise einen Drucker mit der Hausnummer 4 ist also 32+4=36 ($24).

5.8.3 Sekundäradressen

Sie kennen sicher die Möglichkeit, auf einer Floppy mehrere Dateien zu eröffnen. Dazu müssen Sie bekanntlich außer unterschiedlichen logischen Filenummern auch jeweils andere Datenkanäle (Sekundäradressen) im OPEN-Befehl angeben, z.B.:

```
10 OPEN 1,8,6,"DATEI1"
20 OPEN 2,8,7,"DATEI2"
```

Die Sekundäradresse dient also dazu, gerätespezifische Funktionen auszulösen. Sie hilft im Falle der Floppy, verschiedene Dateien auseinanderzuhalten, zu welchem Zweck die Sekundäradresse bei jeder Aktion im Anschluß an die Geräteadresse übermittelt wird.

Wie Sie sehen, hat die Sekundäradresse also keine generelle Steuerfunktion auf dem IEC-Bus, sondern eine von Gerät zu Gerät unterschiedliche Bedeutung. Die für die Floppy relevanten Sekundäradressen wollen wir nachfolgend vorstellen.
Auch hier findet eine Zweiteilung statt, wobei der niederwertige Teil die eigentliche Sekundäradresse enthält, die im OPEN-Befehl angegeben wurde (Bereich 0-15), und der höherwertige Teil diejenige Information enthält, die im Zusammenhang mit der Sekundäradresse auftritt:

```
$60 (96)  PRINT, INPUT oder GET
$E0 (224) CLOSE
$F0 (240) OPEN
```

Eine Besonderheit stellen die Sekundäradressen 0 und 1 dar. Die Floppy interpretiert die 0 als LOAD-Befehl, und die 1 als SAVE. Deshalb dürfen diese Werte nie in einem OPEN-Befehl angewendet werden.

Bei den Commodore-Druckern kann über die Sekundäradresse u.a. der Zeichensatz ausgewählt werden.

5.8.4 Die Systemvariable ST

Die numerische Variable ST gibt darüber Auskunft, wie eine Aktion auf dem IEC-Bus ausging. Für den Programmierer heißt das, daß er tunlichst nach jedem PRINT, INPUT oder dergleichen diese Variable abfragen sollte, da bei Störungen auf dem Bus sonst Daten verlorengehen könnten.

Diese Variable ist vom BASIC aus mit unter der Variablen ST abzurufen und belegt die Adresse $90 in der Zero-Page, wo Sie auch die entsprechenden Werte zu den entsprechenden Ereignissen nachschlagen können.

Die Ereignisse werden durch Setzen der entsprechenden Bits gekennzeichnet. Es ist möglich, daß mehrere Bits gesetzt sind, weshalb es nötig ist die abzufragenden Bits zu isolieren

```
100 GET#1,A$:IF (ST AND 64) THEN .....
```

Durch die serielle Ausgabe der Zeichen und durch die wenigen Leitungen ,die zur Verfügung stehen, sind alle Abläufe im Zusammenhang mit der Datenübertragung sehr zeitkritisch. Da das Timing genau eingehalten werden muß, ist es verständlich, daß es bei den anfangs auf den Markt gekommenen Interfaces von Fremdherstellern oft zu Fehlinterpretationen des Leitungszustands kam. Commodore selbst hatte geringe Schwierigkeiten, wie man am Drucker VC-1526 sehen konnte.

Die folgenden Ausführungen sollen deshalb auch nur der Beleuchtung des Prinzips dienen. Zur Konstruktion eigener Peripherie sind sie nicht unbedingt geeignet. Sie sollten dazu schon über einen gewissen Meßgerätepark verfügen, mit dem Sie die Vorgänge auf dem seriellen Bus sichtbar machen und daraus mit Hilfe dieser Beschreibung Ihre Schlüsse ziehen können.

5.8.5 Adressierung

Als Beispiel für die folgenden Illustrationen der Vorgänge auf dem IEC-Bus sollen die Basic-Zeilen

```
10 OPEN 1,8,15
20 PRINT#1,"I"
```

dienen. Es soll also einfach das Zeichen A auf einem Drucker mit der Geräteadresse 8 ausgegeben werden. Auslöser für die Aktionen auf dem Bus ist nur die Zeile 20. Von dem OPEN

merkt der Drucker noch nichts, da, anders als bei der Floppy, hiermit kein Dateiname verbunden ist, der übermittelt werden müßte.

1. Als erstes wird hier ATN HIGH zum Zeichen, daß ein Befehl folgt.

2. Daraufhin wird die Leitung DATA auf LOW gesetzt um sofort danach von der Hardware der Floppy auf HIGH gelegt zu werden. Wird die Leitung nicht auf HIGH gesetzt, erfolgt ein "DEVICE NOT PRESENT".

3. Die Leitung DATA übernimmt jetzt die Funktion, dem TALKER zu verstehen zu geben, daß er zur Zeit noch nicht in der Lage ist ein Zeichen zu empfangen, weil er beispielsweise mit der Verarbeitung vorheriger Daten beschäftigt ist. Das heißt, daß, solange die DATA-Leitung HIGH ist, das Gerät keine Daten aufnehmen kann. Ist es soweit, wird DATA=LOW.

4. Der Computer signalisiert jetzt mit mit CLOCK=1 das er bereit ist Daten zu senden.

5. Als nächstes kommt die Geräteadresse über die Leitung DATA, und zwar ein Bit mit jedem LOW-Impuls auf der Leitung CLK.

6. Innerhalb von 1ms muß die Floppy jetzt die Leitung DATA nach HIGH bringen, um zu signalisieren daß sie die Daten empfangen hat und verarbeitet.

Sie sehen hier schon, daß DATA drei Aufgaben hat, nämlich zu signalisieren, wann der Empfänger bereit, ist Daten zu empfangen, die Daten zu schicken und nach Beendigung dieses Vorgangs eine Rückmeldung zu übermitteln. Die DATA-Leitung wird also sowohl vom TALKER als auch vom LISTENER verwendet.

5.8.6 Der Datentransfer

Der Datentransfer geschieht analog zu der Adressierung nur mit dem Unterschied, daß die ATN-Leitung nicht benötigt wird. Auf die gleiche Weise folgt der abschließende CHR$(13).
Wo bleibt das Signal EOI?

Das folgt jetzt, womit die Leitung DATA eine vierte Aufgabe bekommt.

7. Die Vereinbarung sah bis jetzt so aus, daß nach Runtersetzen der Leitung DATA durch den LISTENER der TALKER durch CLK=HIGH den Datentransfer einleitete. Nun bleibt aber CLK länger LOW, was schlicht bedeutet, daß das zuvor übertragene Byte das letzte war.

Auch dieser Umstand muß vom LISTENER mit DATA=HIGH quittiert werden.

8. Sobald DATA wieder LOW wird, überträgt der TALKER ein Leerbyte (CHR$(0) ab Punkt 16). Danach geht der Bus in den Ruhezustand.

Die Adressierung erfolgt analog zu Punkt 1-6, wobei das Datenbyte den Wert $3F (48+15) aufweist, um, wie im Punkt 5.8.2 beschrieben, die Betriebsart LISTEN zu beenden.

Wie Sie sehen, funktioniert es wunderbar. Den gravierenden Nachteil, den ein serieller Bus systembedingt gegenüber dem parallelen hat, nämlich die wesentlich geringere Geschwindigkeit, wird durch drastisch verkürzte Wartezeiten (1ms) teilweise aufgefangen.

Nach diesen Erläuterungen muß noch auf die Adressierung der einzelnen Leitungen hingewiesen werden. Hier muß man zwischen den Empfangs- und den Sendeleitungen unterscheiden. Die Leitungen CLK und DATA werden aufgespalten und auf vier Portleitungen verteilt. Somit existieren die Leitungen CLK-IN, DATA-IN, CLK-OUT und DATA-OUT. Die Sendeleitungen sind mit Dioden so abgeschirmt, daß kein empfangenes

Signal ihre Portleitungen beeinflussen kann. Die von ihnen geschickten Signale werden umgepolt, bevor sie den Empfänger erreichen (aus LOW wird HIGH und aus HIGH wird LOW). Sie beeinflussen jedoch außer den Eingangsleitungen der anderen Geräte auch noch die des eigenen.

Weil der Firma Commodore der Einbau eines zweiten Umpolers bei den hereinkommenden Signalen zu teuer gewesen ist, ließ man ihn weg und überbrückte die somit entstandene Lücke durch nicht standardgemäße Software. Die hier erläuterte Arbeitsweise des IEC-Bus bezieht sich auf den Standart IEC-Bus und nicht auf den "Krüppel" des C64. Das Rom wurde zum besserem Verständniß so dokumentiert, wie es dem Standard-Bus entspricht. Der Programmierer muß hier umdenken, da alle hereinkommenden Signale invertiert sind.

Die folgende Tabelle gibt Ihnen diejenigen Bits an, mit denen Sie die Leitungen beeinflussen oder abfragen können. Alle Leitungen sind am Port A der CIA 2 mit der Adresse $DD00 angeschlossen.

Leitung	Eingang	Ausgang
DATA	Bit 7	Bit 5
CLK	Bit 6	Bit 4
ATN		Bit 3

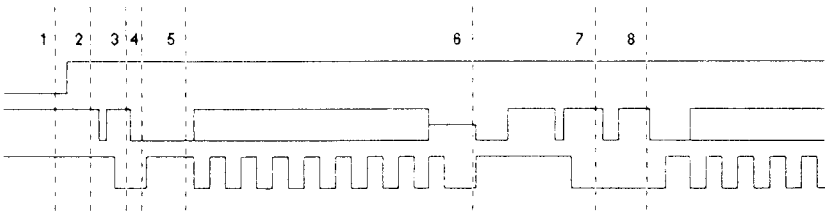


Abb. 5.8.5: Zeitdiagramm

5.8.7 Die Programmierung des IEC-Bus

Wie läßt sich die Bedienung des IEC-Bus in Maschinensprache programmieren? Für alle Aufgaben stehen im Betriebssystem Unterprogramme zur Verfügung, die in der Sprungtabelle im obersten ROM-Bereich zusammengefaßt sind. Siehe dazu die letzte Seite des ROM-Listings. Als Beispiel wollen wir uns ansehen, wie wir die Fehlermeldung der Floppy-Disk einlesen können. Schauen wir erst, wie dies in BASIC gemacht wird.

```
10 OPEN 15,8,15 : REM öffnen des Fehlerkanals
20 INPUT#15,A$,B$,C$,D$ : REM Fehlermeldung holen
30 PRINT A$;",";B$;",";C$;",";D$ : REM und ausgeben
40 CLOSE 15 : REM Kanal schließen
```

In BASIC ist dies wegen des INPUT-Befehls nur im Programmodus möglich. Hier ist ein Maschinenprogramm, das die gleichen Dienste tut:

```
C000  A9 08     LDA  #8         ; Geräteadresse der Floppy
C002  85 BA     STA  $BA
C004  20 B4 FF  JSR  $FFB4      ; Talk senden
C007  A9 6F     LDA  #15 + $60  ; Sekundäradr. 15 plus $60
C009  85 B9     STA  $B9
C00B  20 96 FF  JSR  $FF96      ; Sekundäradresse für Talk
C00E  20 A5 FF  JSR  $FFA5      ; Zeichen von Floppy holen
C011  20 D2 FF  JSR  $FFD2      ; auf Bildschirm ausgeben
C014  C9 0D     CMP  #$0D       ; ist es carriage return ?
C016  D0 F6     BNE  $C00E      ; nein, weitere Zeichen
C018  20 AB FF  JSR  $FFAB      ; Untalk senden
C01A  60        RTS             ; fertigzln1
```

Hier ein Ladeprogramm in BASIC:

```
100 FOR I = 49152 TO 49179
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169,  8,133,186, 32,180,255,169,111,133,185, 32
130 DATA 150,255, 32,165,255, 32,210,255,201, 13,208,246
```

```
140 DATA  32,171,255, 96
150 IF S <> 4169 THEN PRINT "FEHLER IN DATAS !!" : END
160 PRINT "OK !"
```

Von BASIC aus läßt das Programm sich mit SYS 12*4096 aufrufen. Mit einem etwas längeren Maschinenprogramm kann man auf einfache Weise auch das Inhaltsverzeichnis der Diskette anzeigen. Man erspart sich auf diese Weise, das Directory mit LOAD "$",8 als BASIC-Programm zu laden, wobei immer das jeweilige BASIC-Programm im Speicher verloren geht.

```
C000 A9 24      LDA  #$24      ; Dollarzeichen als
                               Filename
C002 85 FB      STA  $FB       ; speichern
C004 A9 FB      LDA  #$FB      ; Adresse des Filenamens
C006 85 BB      STA  $BB
C008 A9 00      LDA  #$00      ; high Byte
C00A 85 BC      STA  $BC
C00C A9 01      LDA  #$01      ; Länge des Filenamens
C00E 85 B7      STA  $B7
C010 A9 08      LDA  #$08      ; Gerätenummer der Floppy
C012 85 BA      STA  $BA
C014 A9 60      LDA  #$60      ; Sekundäradresse für LOAD
C016 85 B9      STA  $B9
C018 20 D5 F3   JSR  $F3D5     ; File mit Namen eröffnen
C01B A5 BA      LDA  $BA
C01D 20 B4 FF   JSR  $FFB4     ; Talk senden
C020 A5 B9      LDA  $B9
C022 20 96 FF   JSR  $FF96     ; Sekundäradresse senden
C025 A9 00      LDA  #$00
C027 85 90      STA  $90       ; Status löschen
C029 A0 03      LDY  #$03      ; ersten 3 Byte überlesen
C02B 84 FB      STY  $FB       ; als Zähler merken
C02D 20 A5 FF   JSR  $FFA5     ; Byte von Floppy holen
C030 85 FC      STA  $FC       ; und merken
C032 A4 90      LDY  $90       ; Status testen
C034 D0 2F      BNE  $C065
C036 20 A5 FF   JSR  $FFA5     ; Byte von Floppy holen
```

```
C039 A4 90      LDY  $90       ; Status testen
C03B D0 28      BNE  $C065
C03D A4 FB      LDY  $FB       ; Zähler holen
C03F 88         DEY            ; und erniedrigen
C040 D0 E9      BNE  $C02B
C042 A6 FC      LDX  $FC       ; Byte zurückholen
C044 20 CD BD   JSR  $BDCD     ; 16-Bit Zahl ausgeben
C047 A9 20      LDA  #$20      ; Zahl der belegten Blocks
C049 20 D2 FF   JSR  $FFD2     ; Leerzeichen ausgeben
C04C 20 A5 FF   JSR  $FFA5     ; nächstes Byte holen
C04F A6 90      LDX  $90       ; Status testen
C051 D0 12      BNE  $C065
C053 AA         TAX            ; Byte testen
C054 F0 06      BEQ  $C05C     ; Null? dann Zeilenende
C056 20 D2 FF   JSR  $FFD2     ; sonst ausgeben
C059 4C 4C C0   JMP  $C04C     ; und nächstes Zeichen
                               holen
C05C A9 0D      LDA  #$0D      ; carriage return
C05E 20 D2 FF   JSR  $FFD2     ; ausgeben
C061 A0 02      LDY  #$02      ; zwei Bytes für
                               Linkadresse
C063 D0 C6      BNE  $C02B     ; weitermachen
C065 20 42 F6   JSR  $F642     ; Datei schließen
C068 60         RTS
```

Hier wieder das Ladeprogramm:

```
100 FOR I = 49152 TO 49256
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 36,133,251,169,251,133,187,169,  0,133,188
130 DATA 169,  1,133,183,169,  8,133,186,169, 96,133,185
140 DATA  32,213,243,165,186, 32,180,255,165,185, 32,150
150 DATA 255,169,  0,133,144,160,  3,132,251, 32,165,255
160 DATA 133,252,164,144,208, 47, 32,165,255,164,144,208
170 DATA  40,164,251,136,208,233,166,252, 32,205,189,169
180 DATA  32, 32,210,255, 32,165,255,166,144,208, 18,170
190 DATA 240,  6, 32,210,255, 76, 76,192,169, 13, 32,210
```

```
200 DATA 255,160,  2,208,198, 32, 66,246, 96
210 IF S <> 15343 THEN PRINT "FEHLER IN DATAS !!" : END
220 PRINT "OK !"
```

Der Aufruf von BASIC geschieht wieder mit SYS 12*4096. Es wird dann das Inhaltsverzeichnis der Diskette auf dem Bildschirm angezeigt, ohne daß ein gespeichertes BASIC-Programm verloren geht.

Statt des seriellen IEC-Bus kann es sich vor allem aus Geschwindigkeitsgründen lohnen, ein IEC-Bus Modul einzusetzen, das auf den Memory Expansion Port gesetzt wird. Sie können dann sämtliche Peripheriegeräte der großen Commodoregeräte benutzen, wie z.B. die großen Diskettendoppellaufwerke.

Bei der Programmierung des parallelen IEC-Bus ändern sich lediglich die Adressen der Routinen wie z.B. Ein- und Ausgabe eines Bytes auf den IEC-Bus oder Senden von TALK oder LISTEN. Programmieren Sie dagegen Ihre Ein- und Ausgabe auf den IEC-Bus über logische Dateien mit BASIN ($FFCF) und BSOUT ($FFD2), so sind überhaupt keine Änderungen erforderlich.

5.9 Das serielle Schieberegister

Der serielle Datenport SDR ist ein synchrones 8-Bit Schieberegister. CRA Bit6 bestimmt Ein- oder Ausgabemodus.

Im Eingabemodus werden die Daten an SP mit der steigenden Flanke eines an CNT liegenden Signals in ein Schieberegister übernommen. Nach 8 CNT-Pulsen wird der Inhalt des Schieberegisters nach SDR gebracht und das SP-Bit im ICR gesetzt.

Im Ausgabemodus fungiert Timer A als Baudrate-Generator. Die Daten aus SDR werden mit der halben Unterlauffrequenz von Timer A nach SP hinausgeschoben. Die theoretisch höchste Baudrate beträgt demnach 1/4 des Systemtaktes.

Die Übertragung beginnt, nachdem Daten ins SDR geschrieben wurden, vorausgesetzt, Timer A läuft und befindet sich im Continuous-Modus (CRA Bit 0=1 und Bit 3=0).

Er von Timer A abgeleitete Takt erscheint an CNT. Die Daten aus SDR werden in das Schieberegister geladen und dann mit jeder fallenden Flanke an CNT aus SP hinausgeschoben. Nach 8 CNT-Pulsen wird der SP-Interrupt erzeugt. Wird jedoch SDR vor diesem Ereignis mit neuen Daten geladen, so werden diese nun automatisch ins Schieberegister geladen und hinausgeschoben. In diesem Falle erscheint kein Interrupt.

Die Daten aus SDR werden mit dem höchstwertigen Bit voran hinausgeschoben. Eingehende Daten sollten dasselbe Format aufweisen.

5.10 Pinbelegung der CIA

Das Blockschema der CIA 6526 finden Sie auf der nächsten Seite. Pinbelegung des 40-poligen Gehäuses:

1	Masse
2 - 9	I/O-Port A; 8 Bit bidirektional
10-17	I/O-Port B; 8 Bit bidirektional. Die Bits 6+7 können zur Anzeige des Unterlaufs der beiden Timer programmiert werden.
18	-PC (Port Control); nur Ausgang; signalisiert die Verfügbarkeit von Daten am Port B oder an beiden Ports.
19	TOD (Time Of Day); nur Eingang 50/60 Hz; triggert die Echtzeituhr.
20	+5V; Betriebsspannung
21	-IRQ(Interrupt Request); nur Ausgang; wird 0 bei übereinstimmung eines gesetzten Bits im ICR mit dem Eintreffen des zugehörigen Ereignisses.
22	R/W(Read/-Write); nur Eingang; 0=Übernahme des Datenbus, 1=Ausgabe auf den Datenbus.
23	-CS (Chip Select); nur Eingang; 0=Datenbus gültig, 1=Datenbus hochohmig (Tri-State).
24	-FLAG; nur Eingang; Bedeutung wie -PC.

25	02 (Systemtakt 2); nur Eingang; alle Datenbusaktionen finden nur bei 02=1 statt.
26-33	DB7-DB0(Datenbus); bidirektional; Schnittstelle zum Prozessor.
34	-RES(Reset); nur Eingang; 0=Rücksetzen der CIA in den Grundzustand.
35-38	RS3-RS0(Register Select); nur Eingang; dient zur Auswahl eines der 16 Register der CIA; nur gültig mit -CS=0.
39	SP(Serial Port); bidirektional; dient als Ein-/ Ausgang des Schieberegisters.
40	CNT(Count); bidirektional; Ein-/ Ausgang des Schieberegistertakts oder Triggereingang für die Intervalltimer.

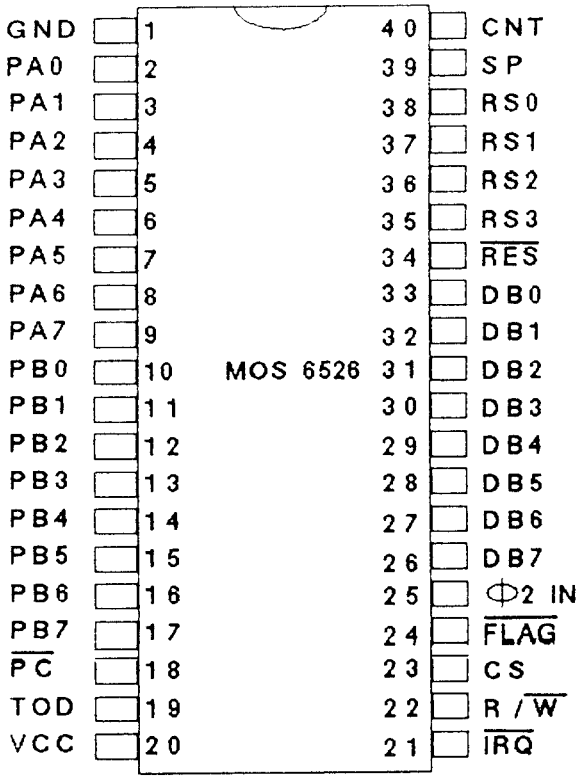


Abb. 5.10: Der 6526

5.11 Der User-Port

Mit dem User-Port bietet der C64 eine Schnittstelle, die zur Steuerung, der Name sagt es, anwenderspezifischer Peripheriegeräte dient.

Das wären im einfachsten Falle über Treibertransistoren angeschlossene Lampen, das könnte aber auch ein Drucker mit 8-Bit Parallelschnittstelle (Centronics) sein, den Sie vielleicht zufällig besitzen und gerne am C64 betreiben würden.

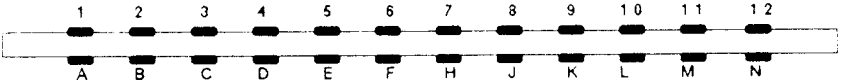
Der User-Port besteht im wesentlichen aus einem 8-Bit-Port und diversen Steuerleitungen, die im folgenden näher vorgestellt werden:

```
1   GND
2   +5V     ;mit max. 100mA belastbar
3   -RESET  ;mit der gleichnamigen Prozessorleitung verbunden
4   CNT1    ;verbunden mit CNT von CIA 1
5   SP1     ;mit SP von CIA 1 verbunden
6   CNT2    ;Leitung CNT von CIA 2
7   SP2     ;verbunden mit SP von CIA 2
8   -PC2    ;Handshake-Ausgang von CIA 2
9   ATN OUT;Steuerleitung des seriellen IEC-Bus, stammt
             von PA3 der CIA 2
0   9V AC   ;Wechselspannung; mit max. 100 mA belastbar.
11          ;Gegenpol zu oben
12          ;GND
A           ;GND
B   -FLAG2  ;Handshake-Eingang von CIA 2
C-L PB0-PB7;I/O-Lines von CIA 2
M   PA2     ;I/O-Line von CIA 2. Diese Leitung ersetzt den von
             den anderen CBMs bekannten CB2 der VIA 6522.z
N   GND
```

Einige der oben aufgeführten Leitungen haben im C64 bereits fest zugeordnete Funktionen. Hierzu und auch zur Handhabung der CIAs lesen Sie sich bitte Kapitel 5 ab Punkt 5.3 durch.

Wollen Sie an den User-Port Geräte anschließen, die eigentlich zum Anschluß an den User-Port des VC-20 oder CBM 8032 vorgesehen sind, so ist das prinzipiell nur bei den Geräten möglich, die zu ihrem Betrieb ausschließlich die Pins A-N, 1 und 12 benötigen.

USER – PORT :



Pin	Signal
1	GND
2	+ 5V
3	RESET
4	CNT1
5	SP1
6	CNT2
7	SP2
8	PC2
9	ATN IN
10	9 VAC
11	9 VAC
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

Abb. 5.11.1: User-Port

Auf jeden Fall müssen Sie die programmtechnisch unterschiedliche Handhabung der Pins B und M beachten.

Sollten Sie eigene Projekte am User-Port anschließen wollen, so beachten Sie bitte, um eine Beschädigung des Rechners zu vermeiden, unbedingt folgendes:

Bei Verwendung des User-Port als Eingang darf die Eingangsspannung nur im Bereich 0-5 Volt liegen. Eine Spannung im Bereich 0-0,6 Volt wird beim Auslesen des Datenports als 0

interpretiert, eine solche von 1,6-5 Volt als 1. Der Bereich von 0,7 bis 1,5 Volt ist indifferent, d.h. er kann zufällig als 0 oder 1 erkannt werden.

Bei Verwendung als Ausgang beachten Sie bitte, daß die Ausgänge nur eine Belastung eines TTL-Eingangs aushalten. Sie könnten also keinesfalls eine Leuchtdiode direkt anschließen, was langfristig zur Zerstörung der CIA führen würde. Es empfiehlt sich immer eine Pufferstufe, wie auch in unserem Beispiel.

Vermeiden Sie unbedingt, ein auf Ausgabe programmiertes Portbit mit einer Fremdspannung von außen zu beaufschlagen, was zur unmittelbaren Zerstörung führt. Überlegen Sie sich daher gut, welchen Wert Sie in das Datenrichtungsregister laden, damit Sie nicht versehentlich ein für die Eingabe vorgesehenes Bit auf Ausgabe programmieren.

Wenn Sie die Stromversorgung für Ihr Projekt dem User-Port entnehmen wollen, beachten Sie bitte, daß Sie die beiden zur Verfügung stehenden Spannungen nicht mit mehr als je 100mA belasten. Bei leichten Übertretungen wird zunächst der Kassettenrecorder streiken, danach verabschiedet sich die Sicherung im Innern des C64 und evtl. auch die Primärsicherung im Trafogehäuse. Zerstört wird dabei jedoch nichts weiter.

5.12 Registerbeschreibung der CIA

REG 0	PRA	(Port Register A)
	Zugriff:	READ/WRITEzpg65
	Bit 0-7	Dieses Register entspricht dem Zustand der Pins PA0-7.
REG 1	PRB	(Port Register B)
	Zugriff:	READ/WRITE
	Bit 0-7	Dieses Register entspricht dem Zustand der Pins PB0-7.

REG 2	DDRA	(Datenrichtung Register A)
	Zugriff:	READ/WRITE
	Bit 0-7	Diese Bits bestimmen die Datenrichtung der korrespondierenden Datenbits des Ports A. 0=Eingang, 1=Ausgang.
REG 3	DDRB	(Datenrichtung Register B)
	Zugriff:	READ/WRITE
	Bit 0-7	Diese Bits bestimmen die Datenrichtung der entsprechenden Datenbits des Ports B. 0=Eingang, 1=Ausgang.
REG 4	TA LO	(Timer A LO-Byte)
	Zugriff:	READ
	Bit 0-7	Dieses Register gibt den augenblicklichen Zustand des niederwertigen Bytes von Timer A wieder.
	Zugriff:	WRITE
	Bit 0-7	In dieses Register wird das niederwertige Byte des Werts geladen, von dem der Timer auf null zählen soll.
REG 5	TA HI	(Timer A HI-Byte)
	Zugriff:	READ
	Bit 0-7	Dieses Register gibt den augenblicklichen Zustand des höherwertigen Bytes von Timer A wieder.
	Zugriff:	WRITE
	Bit 0-7	In dieses Register wird das höherwertige Byte Werts geladen, von dem der Timer auf null zählen soll.
REG 6	TB LO	(Timer B LO-Byte) Zugriff und Belegung entspricht REG 4.
REG 7	TB HI	(Timer B HI-Byte) Zugriff und Belegung entspricht REG 5.

REG 8 TOD 10 (Uhr 1/10 sec)
Zugriff: READ
Bit 0-3 Zehntelsekunden der Echtzeituhr im BCD-Format.
Bit 4-7 Immer 0.
Zugriff: WRITE und CRB Bit 7=0
Bit 0-3 Zehntelsekunden im BCD-Format.
Bit 4-7 Müssen 0 sein.
Zugriff: WRITE und CRB Bit 7=1
Bit 0-3 Vorwahl der Zehntelsekunden der Alarmzeit im BCD-Format.
Bit 4-7 Müssen 0 sein.

REG 9 TOD SEC (Uhr sec)
Zugriff: READ
Bit 0-3 Einersekunden der Uhr im BCD-Format.
Bit 4-6 Zehnersekunden der Uhr im BCD-Format.
Bit 7 Immer 0.
Weitere Zugriffsarten analog zu REG 8.

REG 10 TOD MIN (Uhr min)
Zugriff: READ
Bit 0-3 Einerminuten der Uhr im BCD-Format.
Bit 4-6 Zehnerminuten der Uhr im BCD-Format.
Bit 7 Immer 0.
Weitere Zugriffsarten analog zu REG 8.

REG 11 TOD HR (Uhr Std)
Zugriff: READ
Bit 0-3 Einerstunden der Uhr im BCD-Format.
Bit 4 Zehnerstunde der Uhr.
Bit 5-6 Immer 0.
Bit 7 0=vormittags(AM), 1=nachmittags(PM).
Weitere Zugriffsarten analog zu REG 8.

REG 12 SDR (Serial Data Register)
Zugriff: READ/WRITE
Bit 0-7 Aus diesem Register werden die Daten bitweise zum Pin SP hinausgeschoben, bzw.

vom Pin SP in dieses Register hineingeschoben.

REG 13 ICR (Interrupt Control Register)

Zugriff: READ (INT DATA)

Bit 0 1=Unterlauf Timer A.

Bit 1 1=Unterlauf Timer B.

Bit 2 1=Gleichheit von Uhrzeit und gewählter Alarmzeit.

Bit 3 1=SDR voll/leer (abhängig von der Betriebsart).

Bit 4 1=Signal am Pin FLAG aufgetreten.

Bit 5-6 Immer 0.

Bit 7 Übereinstimmung mindestens eines Bits von INT MASK und INT DATA aufgetreten.

Achtung: Beim Lesen dieses Registers werden alle Bits gelöscht.

Zugriff: WRITE (INT MASK)
Bedeutung der Bits wie oben, ausgenommen Bit 7

Bit 7 1=jedes 1-Bit setzt das korrespondierende Masken-Bit. Die anderen bleiben unberührt.
0=jedes 1-Bit löscht das korrespondierende Masken-Bit. Die anderen bleiben unberührt.

REG 14 CRA (Control Register A)

Zugriff: READ/WRITE

Bit 0 1=Timer A Start, 0=Stop

Bit 1 1=Unterlauf von Timer A wird an Pin PB6 signalisiert.

Bit 2 1=jeder Unterlauf von Timer A kippt PB6 in die jeweils andere Lage, 0=jeder Unterlauf von Timer A erzeugt an PB6 einen HI-Puls mit der Länge eines Systemtakts.

Bit 3 1=Timer A zählt nur einmal vom Ausgangswert auf null und hält dann an, 0=Timer A zählt fortlaufend vom Ausgangswert auf null.

Bit 4	1=unbedingtes Laden eines neuen Startwertes in Timer A. Dieses Bit fungiert als Strobe. Es muß bei jedem unbedingten Laden neu gesetzt werden.
Bit 5	Dieses Bit bestimmt die Quelle des Timer-Triggers. 1=Timer zählt steigende CNT-Flanken, 0=Timer zählt Systemtaktpulse.
Bit 6	1=SP ist Ausgang, 0=SP ist Eingang.
Bit 7	1=Echtzeituhr-Trigger beträgt 50Hz, 0=Trigger beträgt 60 Hz.

REG 15 CRB (Control Register B)

Zugriff:	READ/WRITE
Bit 0-4	Diese Bits haben die gleiche Bedeutung wie in REG14, allerdings bezogen auf Timer B und Pin PB7.
Bit 5-6	Diese Bits bestimmen die Quelle des Triggers für Timer B. 00=Timer zählt Systemtakte, 01=Timer zählt steigende CNT-Flanken, 10=Timer B zählt Unterläufe von Timer A, 11=Timer B zählt Unterläufe von Timer A, wenn CNT=1 ist.
Bit 7	1=Alarm setzen, 0=Uhrzeit setzen.

6. Das ROM-Listing

6.1 Nutzung des ROM-Listings

Ein Vorteil der Assemblerprogrammierung ist es, daß der Anwender auf die Routinen im BASIC-ROM und im Betriebssystem zurückgreifen kann. Das Benutzen dieser Routinen erspart viel Arbeit und ist außerdem noch platzsparend. Häufig macht man sich unnötig viele Gedanken über ein programmspezifisches Problem und rauft sich am Ende die Haare, wenn es dann doch nicht tadellos funktioniert, obwohl im Betriebssystem eine entsprechende Routine schon vorhanden ist.

Wir wollen Ihnen an dieser Stelle anhand von einigen Beispielen demonstrieren, wie man diese Routinen am besten nutzt und was dabei zu beachten ist. Eine Zusammenfassung der wichtigsten Routinen finden Sie in Kapitel 6.2.

Ein Problem, das sich häufig stellt, ist das Verschieben von Speicherbereichen. Dieses Problem kann zwar durch Verschachtelung mehrerer Schleifen gelöst werden, viel eleganter und vor allem günstiger ist es jedoch, die fertige Routine im BASIC-ROM zu verwenden. Diese Routine wird intern zum Beispiel benutzt, wenn eine neue BASIC-Zeile eingefügt wurde und alle Variablen verschoben werden müssen. Sie steht im ROM ab der Adresse $A3BF (41919). Vor dem Einsprung müssen jedoch einige Bedingungen erfüllt werden: Der alte Blockanfang sowie das alte und neue Blockende müssen in bestimmten Adressen in der Zeropage abgelegt werden. Eine Anwendung dieser Routine, in der der Bereich von $1000 bis $1500 in den Bereich $2000 verschoben werden soll, könnte folgendermaßen gestaltet werden:

```
LDX #$00    ; alter Blockanfang (LOW-Byte)
LDY #$10    ; alter Blockanfang (HIGH-Byte)
STX $5F     ; abspeichern
STY $60     ; abspeichern
LDX #$01    ; altes Blockende (LOW-Byte +1)
LDY #$15    ; altes Blockende (High-Byte)
```

```
STX $5A     ; abspeichern
STY $5B     ; abspeichern
LDX #$01    ; neues Blockende (Low-Byte + 1)
LDY #$15    ; neues Blockende (High-Byte)
STX $58     ; abspeichern
STY $59     ; abspeichern
JSR $A3BF   ; Verschieberoutine anspringen
RTS         ; Rücksprung
```

Wie Sie sehen, ist diese Methode recht einfach und auch relativ kurz. Poked man die Werte in die Zeropage, so findet die Routine auch im BASIC Verwendung, wenn man sie mit SYS 41919 aufruft.

Eine weitere nützliche Routine beginnt ab $E50A (58634). Sie kann benutzt werden, um die Cursorposition abzurufen oder diese zu verändern. Bei gesetztem Carry-Flag wird die aktuelle Position des Cursors geholt. Die Zeile wird in das X-Register und die Spalte in das Y-Register übertragen. Ist das Carry-Flag hingegen gelöscht, wird der Cursor mit den Werten aus den beiden Registern neu positioniert. Auf den ersten Blick scheint die Routine für BASIC nicht verwendbar, doch hier zeigt sich wieder, daß die Nutzung des ROM-Listings auch etwas Eigeninitiative voraussetzt. Wenn man die einzelnen Routinen der Hauptroutine selbständig aufruft und das bewährte Hilfsmittel Poke hinzuzieht, läßt sich die Routine auch aus dem BASIC heraus benutzen. Es empfiehlt sich also immer, die benötigten Routinen vorher sorgfältig durchzuarbeiten und sich mit ihnen vertraut zu machen.

Die Nutzung des ROM-Listings beschränkt sich aber nicht nur auf die fertigen Routinen. Es ist durchaus möglich, diese Routinen zu verändern, um sie seinen Wünschen anzupassen oder auch um sie zu verbessern. Da sich das ROM aber nicht verändern läßt, muß man es vorher in einen RAM-Bereich kopieren. Hier bietet sich das RAM an, das sozusagen parallel unter dem ROM liegt. Die hierzu benögigte Kopierroutine sieht etwas absurd aus, da zum Beispiel

POKE 49000, PEEK (49000)

auf den ersten Blick keinen Sinn hat. Dieser Befehl bewirkt aber, daß der Inhalt dieser Speicherzelle aus dem ROM gelesen und an die gleiche Stelle in das RAM geschrieben wird. Nach dem Kopiervorgang muß dem Prozessor noch mitgeteilt werden, daß sich das gesamte ROM nun im RAM befindet. Zu dem Kopiervorgang lesen Sie sich bitte auch das Kapitel über die Speicheraufteilung durch, da Sie dort noch weitere, wichtige Informationen bekommen.

Nun können Sie alle gewünschten Veränderungen am ROM vornehmen. Wie wäre es zum Beispiel mit einer neuen Laderoutine, die einen Autostart unterdrückt ? Da ein Autostart nur dann erfolgen kann, wenn bestimmte Vektoren beim Laden überschrieben werden, wie in Kapitel 1 erwähnt, läßt er sich einfach unterdrücken, indem man das Programm in einen anderen Speicherbereich lädt. Hier stößt man auf Schwierigkeiten, wenn es sich um ein Kassettenprogramm handelt, das mit der Sekundäradresse 3 abgespeichert wurde, da dieses wie in Kapitel 4 erwähnt immer in den Orginalbereich geladen wird.

Da das Betriebssystem während des Ladens auf die Sekundäradresse abfragt, ist es uns möglich, die Abfrage an dieser Stelle so zu modifizieren, daß das Programm trotzdem an die von uns angegebene Adresse geladen wird. Sehen wir uns dazu die entsprechenden Adressen der Routine an:

```
$F568  CPX #$03   ; Überprüfung auf Sekundäradresse 3
$F56A  BNE $F549  ; wenn nicht 3, dann verzweige
```

Das ist die Stelle, an der wir eingreifen können, indem wir die Verzweigung so abändern, daß bei Sekundäradresse 3 so verfahren wird wie bei Sekundäradresse 1, wir also angeben können, in welchen Bereich geladen werden soll. So sähe die Verzweigung nach der Änderung aus:

```
$F56A  BEQ $F579  ; Verzweigung wie bei Sekundäradresse 1
```

Auch andere Änderungen können nach demselben Muster vorgenommen werden, wie zum Beispiel die Änderung der Repeatfunktion bei der Tastaturabfrage, oder das Einbringen eigener, deutscher Fehlermeldungen. Hier sind Ihrer Kreativität keine Grenzen gesetzt.

6.2 Verzeichnis der wichtigsten ROM-Routinen

Im folgenden sind die wichtigsten Routinen des BASIC-ROMs und des Betriebssystems zusammengefaßt:

Adresse	Funktion
$A3BF	Speicherinhalte verschieben
$A437	Fehlermeldung ausgeben, Fehlernummer muß im X-Register stehen
$A480	Eingabewarteschleife
$A560	Eingabe einer Zeile per Tastatur
$A613	Speicheradresse einer Programmzeile berechnen
$A644	BASIC-Befehl NEW
$A660	BASIC-Befehl CLR
$A68E	CHRGET-Zeiger auf Anfang des BASIC-Speichers stellen
$A69C	BASIC-Befehl LIST
$A7E1	BASIC-Befehl ausführen
$AB24	String ausgeben, Ausgabegerät ist in $9A festgelegt
$AB3F	Ausgabe eines Leerzeichens
$AB42	Ausgabe Cursor rechts
$AB47	Ausgabe eines ASCII-Zeichens, Akku muß ASCII-Wert enthalten
$AD9E	Beliebigen Ausdruck holen, ist $14 (Typflag) = 0, dann numerisch, ist $14 = FF, dann String
$AEF7	Prüft auf Klammer zu ")"
$AEFA	Prüft auf Klammer auf "("
$AEFD	Prüft auf Komma
$B113	Prüft auf Buchstabe
$B395	16-Bit-Interzahl in A/Y-Reg. nach Fließkommazahl
$B7EB	Holt eine 16-Bit-Interzahl und einen 8-Bit-Wert

$B850 FAC = KONSTANTE - FAC , Akku und Y-Register zeigen auf KONSTANTE (Low- und High-Byte)
$B853 FAC = ARG - FAC
$B867 FAC = KONSTANTE (A/Y) + FAC
$B86A FAC = ARG + FAC
$BA28 FAC = KONSTANTE (A/Y) * FAC
$BA2B FAC = ARG * FAC
$BA8C Fließkommazahl nach ARG bringen, Zeiger auf Zahl in Akku und Y-Register
$BB0F FAC = KONSTANTE (A/Y) / FAC
$BB12 FAC = ARG / FAC
$BBA2 Fließkommazahl nach FAC bringen, Zeiger auf Zahl in Akku und Y-Register
$BBC7 FAC nach Akku #4 übertragen
$BBCA FAC nach Akku #3 übertragen
$BBD4 FAC nach Variable übertragen, Zeiger auf Zieladresse in Akku und Y-Register
$BBFC ARG nach FAC übertragen
$BC0C FAC nach ARG übertragen
$BC2B Vorzeichen von FAC holen, Zero- und Carry-Flag werden beeinflußt
$BC5B Vergleich KONSTANTE (A/Y) mit FAC , Flags werden beeinflußt
$BC9B Umwandlung Fließkomma nach Integer im FAC
$BCF3 Umwandlung ASCII nach Fließkomma
$BDCD Positive Integerzahl aus Akku und X-Reg. ausgeben
$BDDD Umwandlung FAC nach ASCII-Format, ASCII-Werte werden ab $0100 abgelegt
$BF71 SQR, zieht Quadratwurzel aus der Zahl im FAC
$BF78 FAC = ARG ^ KONSTANTE (A/Y)
$BF7B FAC = ARG ^ FAC

$E10C ASCII-Zeichen ausgeben , Wert muß im Akku stehen
$E112 ASCII-Zeichen holen (Eingabegerät wählbar)
$E124 Zeichen aus Tastaturpuffer in Akku holen
$E156 SAVE-Routine
$E165 VERIFY-Routine
$E168 LOAD-Routine
$E1BC OPEN-Befehl
$E1C7 CLOSE-Befehl

$E1D4	Parameter für LOAD und SAVE holen
$E219	Parameter für OPEN und CLOSE holen
$E264	COS, berechnet den Cosinuswert im FAC
$E26B	SIN, berechnet den Sinuswert im FAC
$E2B4	TAN, berechnet den Tangenswert im FAC
$E50A	Cursor setzen/holen, wenn Carry-Flag =0, dann Cursor setzen, sonst Cursor holen (X-Register = Zeile, Y-Register = Spalte)
$E518	Bildschirmreset wird durchgeführt
$E544	CLR, löscht den Bildschirm und setzt Cursor HOME
$E566	HOME, bringt den Cursor in die linke obere Ecke des Bildschirms
$E56C	berechnet die Cursorposition
$E5A0	Videocontroller initialisieren, lädt den Videocontroller mit den Standardwerten
$E5B4	holt ein Zeichen aus dem Tastaturpuffer
$E5CA	wartet auf Tastatureingabe
$E8EA	Bildschirm scrollen, schiebt Bildschirm um eine Zeile nach oben
$E9FF	löscht eine Bildschirmzeile
$EA1C	setzt ein Zeichen mit Farbe auf dem Bildschirm (Bildschirmcode im Akku, Farbe in X)
$FF81	Video-Reset
$FF84	CIAs initialisieren
$FF87	RAM löschen/testen
$FF8A	I/O initialisieren
$FF8D	I/O Vektoren initialisieren
$FF90	Setzt Flag für Ausgabe von Systemmeldung
$FF93	schickt Sekundäradresse nach einem LISTEN-Befehl auf den IEC-Bus
$FF96	schickt Sekundäradresse nach einem TALK-Befehl auf den IEC-Bus
$FF99	holt bei gesetzem Carry-Flag die höchste RAM-Adresse nach X und Y, bei gelöschtem Carry-Flag wird die Adresse von X und Y gesetzt.
$FF9C	dieselbe Funktion wie $FF99, jedoch für den RAM-Anfang
$FF9F	fragt die Tastatur ab
$FFA2	setzt das Time-out-Flag für den IEC-Bus
$FFA5	holt ein Byte vom IEC-Bus in den Akku

$FFA8 gibt ein Byte aus dem Akku an den IEC-Bus aus

$FFAB sendet UNTALK-Befehl auf den IEC-Bus

$FFAE sendet UNLISTEN-Befehl auf den IEC-Bus

$FFB1 sendet LISTEN-Befehl auf den IEC-Bus

$FFB4 sendet TALK-Befehl zum IEC-Bus

$FFB7 holt das Statuswort in den Akku

$FFBA setzt die Fileparameter, Akku muß logische Filenummer enthalten, X = Gerätenummer und Y = Sekundäradresse

$FFBD setzt Parameter des Filenamens, Akku muß Länge des Namens enthalten, X und Y enthalten die Adresse des Filenamens(Low- und High-Byte)

$FFC0 OPEN-Befehl, öffnet logische Datei

$FFC3 CLOSE-Befehl, schließt logische Datei, Akku muß logische Filenummer enthalten

$FFC6 CHKIN setzt folgende Eingabe auf logische Datei, die in X übergeben wird. Die logische Datei muß vorher mit der OPEN-Routine geöffnet werden.

$FFC9 CKOUT setzt folgende Ausgabe auf logische Datei, die in X übergeben wird. Die logische Datei muß vorher mit der OPEN-Routine geöffnet werden.

$FFCC CLRCH setzt die Ein- und Ausgabe wieder auf Standard (Tastatur/Bildschirm)

$FFCF BASIN Eingabe, holt ein Zeichen in den Akku

$FFD2 BSOUT Ausgabe, gibt Zeichen im Akku aus

$FFD5 LOAD, lädt Programm in den Speicher

$FFD8 SAVE, speichert Programm ab

$FFDB setzt die laufende Zeit neu

$FFDE holt die laufende Zeit

$FFE1 fragt die STOP-Taste ab

$FFE4 GET, holt ein Zeichen in den Akku

$FFE7 CLALL, setzt alle Ein-/Ausgabekanäle zurück, die Dateien werden jedoch nicht geschlossen

$FFEA erhöht die laufende Zeit um eine sechzigstel Sekunde

$FFED SCREEN holt die Anzahl der Zeilen und und Spalten des Bildschirms

$FFF0	bei gelöschtem Carry-Flag wird der Cursor auf die Position X/Y gesetzt, bei gesetztem Carry-Flag wird die Cursorposition nach X/Y geholt (X-Reg = Zeile, Y-Reg = Spalte)
$FFF3	holt die Startadresse des I/O-Bausteins

6.3 Alphabetisches Verzeichnis der ROM-Routinen

Abfrage auf gedrückte Bandtaste	$F82E
Adresse eines Arrayelements berechnen	$B30E
Adressen der Basic-Befehle (minus 1)	$A00C
Adressen der Basic-Funktionen	$A052
Adressen der Fehlermeldungen	$A328
Adreßzeiger erhöhen	$FCDB
Anfangswert für RND-Funktion	$E3BA
Arbeitsspeicher initialisieren	$FD50
Arrayelement suchen	$B2E9
Arrayvariable anlegen	$B261
Ausgabe der Zeilennummer bei Fehlermeldung	$BDC2
Ausgabe eines Fragezeichens	$AB45
Ausgabe eines Leerzeichens	$AB3B
Ausgabe in RS 232 Puffer	$F014
ARG = Konstanate (A/Y)	$BA8C
ARG nach FAC übertragen	$BBFC
ASCII-Code nach Bildschirmcode wandeln	$E691
Band für Lesen vorbereiten	$F8E2
Bandheader nach Namen suchen	$F7EA
Bandpuffer auf Band schreiben	$F864
Bandpufferzeiger erhöhen	$F80D
Basic CKOUT-Routine	$E4AD
Basic Kaltstart	$E394
Basic NMI-Einsprung	$E37B
Basic-Befehl CLOSE	$E1C7
Basic-Befehl CLR	$A65E
Basic-Befehl CMD	$AA86
Basic-Befehl CONT	$A857
Basic-Befehl DATA	$A8F8
Basic-Befehl DEF	$B3B3
Basic-Befehl DIM	$B081

Basic-Befehl END	$A831
Basic-Befehl FOR	$A742
Basic-Befehl GET	$AB7B
Basic-Befehl GOSUB	$A883
Basic-Befehl GOTO	$A8A0
Basic-Befehl IF	$A928
Basic-Befehl INPUT	$ABBF
Basic-Befehl INPUT#	$ABA5
Basic-Befehl LET	$A9A5
Basic-Befehl LIST	$A69C
Basic-Befehl LOAD	$E168
Basic-Befehl NEW	$A642
Basic-Befehl NEXT	$AD1D
Basic-Befehl ON	$A94B
Basic-Befehl ON	$A94B
Basic-Befehl OPEN	$E1BE
Basic-Befehl POKE	$B824
Basic-Befehl PRINT	$AAA0
Basic-Befehl PRINT#	$AA80
Basic-Befehl READ	$AC06
Basic-Befehl REM	$A93B
Basic-Befehl RESTORE	$A81D
Basic-Befehl RETURN	$A8D2
Basic-Befehl RUN	$A871
Basic-Befehl SAVE	$E156
Basic-Befehl STOP	$A82F
Basic-Befehl SYS	$E12A
Basic-Befehl VERIFY	$E165
Basic-Befehl WAIT	$B82D
Basic-Befehlsworte	$A09E
Basic-Fehlermeldungen	$A19E
Basic-Funktion ABS	$BC58
Basic-Funktion ASC	$B78B
Basic-Funktion ATN	$E30E
Basic-Funktion CHR$	$B6EC
Basic-Funktion COS	$E264
Basic-Funktion EXP	$BFED
Basic-Funktion FN	$B3F4
Basic-Funktion FRE	$B37D
Basic-Funktion INT	$BCCC

Basic-Funktion LEFT$	$B700
Basic-Funktion LEN	$B77C
Basic-Funktion LOG	$B9EA
Basic-Funktion MID$	$B737
Basic-Funktion PEEK	$B80D
Basic-Funktion POS	$B39E
Basic-Funktion RIGHT$	$B72C
Basic-Funktion RND	$E097
Basic-Funktion SGN	$BC39
Basic-Funktion SIN	$E26B
Basic-Funktion SQR	$BF71
Basic-Funktion STR$	$B465
Basic-Funktion TAN	$E2B4
Basic-Funktion VAL	$B7AD
Basic-Code in Klartext wandeln	$A717
Basic-Operator AND	$AFE9
Basic-Operator NOT	$AED4
Basic-Operator OR	$AFE6
Basic-Routine BASIN	$E112
Basic-Routine BSOUT	$E10C
Basic-Routine CHKIN	$E11E
Basic-Routine CKOUT	$E118
Basic-Routine GETIN	$E124
Basic-Statement ausführen	$A7ED
Basic-Vektoren laden	$E453
Basisadresse der CIAs holen	$E500
Betriebssystem-Meldungen	$E45F
Bildschirm löschen	$E544
Bildschirm scrollen	$E8EA
Bildschirm-Reset	$E518
Bildschirmformat holen	$E505
Bildschirmzeile löschen	$E9FF
Bit auf Band schreiben	$FBA6
Bitweise Multiplikation	$BA59
Bitzähler für serielle Ausgabe setzen	$FB97
Block vom Band lesen	$F841
Blockverschiebe-Routine	$A3B8
Byte auf seriellen Bus ausgeben	$ED40
Byte auf seriellen Bus ausgeben	$EDDD
Byte vom seriellen Bus holen	$EE13

Byte vom Band holen	$F199
Byte von RS 232 holen	$F1B8
Bytewert nach X holen, GETBYT	$B79B
BASIN-Routine	$F157
BSOUT-Routine	$F1CA
Cursor setzen/holen	$E50A
Cursor Home	$E566
Cursorposition berechnen	$E56C
CHKIN-Routine	$F20E
CKOUT-Routine	$F250
CLALL-Routine	$F32F
CLOSE-Routine	$F291
CLRCH-Routine	$F333
Datenbits für RS 232 berechnen	$EF4A
Dimensionierte Variable holen	$B1D1
Division FAC = ARG / FAC	$BB12
Division FAC = Konstante (A/Y) / FAC	$BB0F
Einfügen einer Fortsetzungszeile	$E965
Eingabe einer Zeile	$A560
Eingabe-Warteschleife	$A480
Fehlerbehandlung bei Eingabe	$AB4D
Fehlermeldung ausgeben	$A437
Fehlermeldung des Betriebssystems	$F6FB
Fileparameter setzen	$F31F
Flag für Systemmeldungen setzen	$FE18
Fließkommakonstante -32768	$B1A5
Fließkommakonstante 0.5	$BF11
Fließkommakonstante 10	$BAF9
FAC = FAC * 10	$BAE2
FAC = FAC + 0.5	$B849
FAC = FAC / 10	$BAFE
FAC = Konstante (A/Y)	$BBA2
FAC nach Akku#3 übertragen	$BBCA
FAC nach Akku#4 übertragen	$BBC7
FAC nach ARG übertragen	$BC0C
FAC nach ASCII wandeln und nach $100	$BDDD
FAC nach Variable übertragen	$BBD0
FN-Syntax prüfen	$B3E1
FRESTR	$B6A3
FRMEVL Auswerten eines beliebigen Ausdrucks	$AD9E

FRMNUM Ausdruck holen und auf numerisch prüfen	$AD8A
Garbage Collection	$B526
GETADR und GETBYT, 16- und 8-Bit-Wert holen	$B7EB
GETADR, FAC in positive 16-Bit-Zahl wandeln	$B7F7
GETIN-Routine	$F13E
Hardware und I/O-Vektoren setzen/holen	$FD15
Header auf Band schreiben	$F76A
Hierarchiekodes der Basic-Operatoren	$A080
Hilfsroutine für Arrayberechnung	$B34C
Hintergrundfarbe setzen	$E4DA
Interpreterschleife	$A7AE
Interruptroutine	$EA31
Interruptroutine für Band lesen	$F92C
Interruptroutine für Band schreiben	$FBCD, $FC6A
IRQ-Einsprung	$FF48
IRQ-Vektor setzen	$FCB8
IRQ-Vektoren	$FD98
Kernal Sprungtabelle	$FF81
Konstante Pi	$AEAD
Konstanten für ATN	$E33E
Konstanten für EXP	$BFBF
Konstanten für Fließkomma nach ASCII	$BF16
Konstanten für Fließkomma nach ASCII	$BDB3
Konstanten für LOG	$B9BC
Konstanten für RND	$E08D
Konstanten für SIN und COS	$E2E0
Konstanten für Umwandlung TI nach TI$	$BF3A
Kopie der CHRGET-Routine	$E3A2
Listen senden	$ED0C
Logische Filenummer suchen	$F30F
Löschen und Einfügen von Programmzeilen	$A49C
LOAD-Routine	$F49E
Mantisse von FAC invertieren	$B947
Meldungen des Betriebssystems	$F0BD
Meldungen des Betriebssystems ausgeben	$F12B
Meldungen des Interpreters	$A364
Minus FAC = ARG - FAC	$B853
Minus FAC = Konstante (A/Y) - FAC	$B850
Multiplikation FAC = ARG * FAC	$BA2B
Multiplikation FAC = Konstante (A/Y) * FAC	$BA28

MSB für Zeilenanfänge neu berechnen	$E6B6
Nächste Zeile suchen	$A909
Nächstes Element eines Ausdrucks holen	$AE83
Nächstes Statement suchen	$A906
NMI-Einsprung	$FE43
NMI-Routine für RS 232	$FED6
Obergrenze RAM setzen/holen	$FE25
OPEN-Routine	$F34A
Parameter für aktives File setzen	$FE00
Parameter für Filenamen setzen	$FDF9
Parameter für LOAD und SAVE holen	$E1D4
Parameter für OPEN holen	$E219
Platz für String reservieren	$B4F4
Plus FAC = ARG + FAC	$B86A
Plus FAC = Konstante (A/Y) + FAC	$B867
Polynomberechnung 1	$E043
Polynomberechnung 2	$E059
Positive Integerzahl in A/X ausgeben	$BDCD
Potenzierung FAC = ARG hoch FAC	$BF7B
Potenzierung FAC = ARG hoch Konstante (A/Y)	$BF78
Programm vom Band laden	$F84A
Programmheader vom Band lesen	$F72C
Programmzeiger auf Basic-Start	$A68E
Programmzeile einfügen	$A4ED
Programmzeile löschen	$A4A9
Programmzeilen neu binden	$A533
Prüfung auf numerisch	$AD8D
Prüfung auf Auto-Start-ROM	$FD02
Prüfung auf Buchstabe	$B113
Prüfung auf Erreichen der Endadresse	$FCD1
Prüfung auf Klammer auf	$AEFA
Prüfung auf Klammer zu	$AEF7
Prüfung auf Komma	$AEFD
Prüfung auf Platz im Speicher	$A3FB
Prüfung auf Shift, CTRL, Commodore	$EB48
Prüfung auf Steuerzeichen	$EC44
Prüfung auf Stop-Taste	$A82C
Prüfung auf String	$AD8F
Prüfung auf Systemvariable	$AF14
Prüfung auf Übereinstimmung mit laufendem Zeichen	$AEFD

Rechtsverschieben eines Registers	$B983
Rekordermotor ausschalten	$FCCA
Reset-Routine	$FCE2
RAM für BASIC initialisieren	$E3BF
ROM-Modul Identifizierung	$FD10
RS 232 Ausgabe	$EEBB
RS 232 Ausgabe	$F208
RS 232 CHKIN	$F04D
RS 232 GET	$F086
Schafft Platz im Speicher	$A408
Sekundäradresse nach Listen senden	$EDB9
Sekundäradresse nach Talk senden	$EDC7
Stapelsuch-Routine	$A38E
Startadresse des Bandpuffers holen	$F7D0
Startadresse einer Programmzeile berechnen	$A613
Status holen	$FE07
Stoptaste abfragen	$F6ED
String ausgeben	$AB1E
String holen, Zeiger nach A/Y	$B487
String in reservierten Bereich übertragen	$B67A
Stringparameter holen	$B782
Stringparameter vom Stack holen	$B761
Stringvergleich	$B02E
Stringverknüpfung	$B63D
Stringverwaltung, FRESTR	$B6A3
Stringzeiger berechnen	$B475
SAVE-Routine	$F5DD
Tabelle der BASIC-Vektoren	$E447
Tabelle der Farbkodes	$E8DA
Tabelle der Hardware- und I/O-Vektoren	$FD30
Tabelle der LSB der Bildschirmzeilen-Anfänge	$ECF0
Talk senden	$ED09
Tastatur Dekodiertabelle 1	$EB81
Tastatur Dekodiertabelle 2	$EBC2
Tastatur Dekodiertabelle 3	$EC03
Tastatur Dekodiertabelle 4	$EC78
Tastaturabfrage	$EA87
Term in Klammern holen	$AEF1
Test auf Direkt-Modus	$B3A6
Test auf Hochkomma	$E684

Test auf Stop-Taste	$F8D0
Time erhöhen	$F69B
Time holen	$F6DD
Time setzen	$F6E4
Timeout-Flag für seriellen Bus setzen	$FE21
Timerkonstanten für RS 232 Baud Rate, NTSC-Version	$FEC2
Timerkonstanten für RS 232 Baud Rate, PAL-Version	$E4EC
Umwandlung einer Zeile in Interpreterkode	$A579
Umwandlung ASCII nach Fließkommaformat	$BCF3
Umwandlung Fließkomma nach Integer	$B1B2
Umwandlung Fließkomma nach Integer	$BC9B
Unlisten senden	$EDFE
Untalk senden	$EDEF
Untergrenze RAM setzen/holen	$FE34
Variable anlegen	$B11D
Variable holen	$AF28
Variable holen	$B08B
Vergleich	$B016
Vergleich Konstante (A/Y) mit FAC	$BC5B
Videocontroller initialisieren	$E5A0
Vorzeichen von FAC holen	$BC2B
Warten auf Bandtaste	$F817
Warten auf Bandtaste für Schreiben	$F838
Warten auf Commodore-Taste	$E4E0
Warteschleife für Tastatureingabe	$E5CA
Wertzuweisung an normalen String	$AA2C
Wertzuweisung INTEGER	$A9C4
Wertzuweisung REAL	$A9D6
Wertzuweisung String	$A9D9
Zeichen auf Bildschirm ausgeben	$E716
Zeichen auf Ziffer prüfen	$AA1D
Zeichen aus Tastaturpuffer holen	$E5B4
Zeichen und Farbe auf Bildschirm setzen	$EA1C
Zeichen vom Bildschirm holen	$E632
Zeiger auf erstes Arrayelement berechnen	$B194
Zeiger auf Farb-RAM berechnen	$EA24
Zeiger auf Tastaturdekodiertabellen	$EB79
Zeile nach oben schieben	$E9C8
Zeilennummer holen und in Adressformat wandeln	$A96B
Zeit holen	$AF84

6.4 Die Belegung der Zeropage

Hexadresse	Dezimal	Belegung
00	0	Datenrichtungsregister für Prozessor-Port Bit 0 - 6; 0= Eingang 1= Ausgang
01	1	Im Prozessorport kann man angeben, welche Speicherbereiche ein- oder ausgeschaltet werden. Bitbeschreibung: Bit 0 1 = BASIC-ROM, 0= RAM Bit 1 1 = KERNAL-ROM, 0= RAM Bit 2 1 = I/O 0= Zeichensatz Bit 3 = Datenausgabe von Datasette Bit 4 0 = Taste bei Datasette gedrückt 1 = nicht gedrückt Bit 5 1 = Motor an, 0= Motor aus Die Bits 6 und 7 sind unbenutzt und immer 0
02	2	unbenutzt
03-04	3-4	Vektor für Umwandlung von Fließkomma nach Fest Von diesen Adressen aus beginnt der Interpreter, eine Gleitkommazahl in eine ganze Zahl umzuwandeln. Der Vektor deutet auf die Adresse $B1AA.
05-06	5-6	Vektor für Umwandlung von Fest nach Fließkomma Diese Routine verwandelt eine ganze Zahl in eine Fließkommazahl. Der Zeiger steht auf $B391.

07		Suchzeichen Die Speicherzelle $07 wird oft von BASIC-Programmen als Suchzeiger für Texteingaben verwendet.
08	8	Hochkomma-Flag Wärend der Umwandlung von BASIC-Befehlen in Tokens wird die Speicherzelle $08 als Zwischenspeicher für BASIC-Texteingaben verwendet.
09	9	Speicher für Spalte beim TAB-Befehl Nach der Ausführung von TAB oder SPC wird die Cursorposition in der Speicherzelle 9 zwischengespeichert.
0A	10	0= LOAD, 1= Verify, Flag des Interpreters Weil die Routine von LOAD und VERIFY identisch ist, wird ein Flag benötigt, um zu unterscheiden, ob ein LOAD oder ein VERIFY-Vorgang ausgeführt worden ist.
0B	11	Zeiger im Eingabepuffer, Anzahl der Dimensionen Die Speicherzelle $0B wird dazu verwendet, die Anzahl der Dimensionen zu berechnen. Außerdem wird noch die Länge der Tokenzeile hier angegeben.
0C	12	Flag für DIM Diese Speicherzelle wird benutzt, um festzustellen, ob die Variable ein Array oder schon eine dimensionierte Variable ist.

0D	13	Typflag $00= numerisch, $FF= String Das Flag zeigt dem BASIC-Interpreter an, ob es sich um Zahlenwerte oder um einen String handelt.
0E	14	$80= Interger, $00= Real Wenn eine Gleitkommazahl auftritt, steht in der Speicherzelle $00, bei einer ganzen Zahl eine $80.
0F	15	Hochkommaflag bei LIST Duch diese Speicherzelle wird beim LIST-Befehl duch ein Hochkomma erkannt, ob eine Textkette folgt. Zusätzlich wird in dieser Speicherzelle markiert, ob eine Garbage Collection durchgeführt werden muß oder nicht.
10	16	Flag für FN Hier wird angezeigt, ob es sich um eine Array-Variable oder um eine mit DEF FN definierte Variable handelt.
11	17	$00= INPUT, $40= GET, $98= READ Diese Speicherzelle gibt an, in welche Routine der BASIC-Interpreter verzweigen soll.
12	18	Vorzeichen bei ATN Die Speicherzelle $12 wird von den trigonometrischen Funktionen zur Bestimmung des Vorzeichens verwendet. Zusätzlich dient die Speicherzelle $12 als Vergleichsoperator für Vergleichsoperationen.
13	19	aktives I/O-Gerät $00= Direkteingabe Die Speicherzelle $13 wird als Zeiger für die Peripheriegeräte wie Tastatur,

		Datasette, RS232, User-Port, Bildschirm, Drucker und Floppy verwendet.
14-15	20-21	Integer-Adresse z.B. Zeilennummer In dieser Speicherzelle werden die Zeilennummern von den Befehlen wie ON..GOTO, GOTO, GOSUB, ON..GOSUB und der Zeilenausgabe beim LIST-Befehl gespeichert.
16	22	Zeiger auf Stringstack Die Speicherzelle $16 zeigt auf den nächsten freien Speicherplatz im Stringstack.
17-18	23-24	Zeiger auf zuletzt verwendeten String Der Inhalt dieser beiden Bytes zeigt auf den zuletzt verwendeten Speicherplatz.
19-21	25-33	Stringstack Die Angaben im Stringstack enthalten die Stringlänge sowie die Anfangs- und Endadressen des vorherigen Strings.
22-25	34-37	Zeiger für diverse Zwecke Diese Speicherzellen benutzt der Interpreter, um verschiedene Zwischenergebnisse zu speichern.
26-2A	38-42	Register für Funktionsauswertung und Arithmetik Diese Speicherzellen werden vom BASIC-Interpreter auch zum Speichern von Zwischenergebnissen bei der Multiplikation und Division benutzt.

2B-2C	43-44	Zeiger auf BASIC-Programm Anfang Der Anfangsbereich des BASIC ist in Low- und Highbyte angegeben. Man kann durch die beiden Bytes den BASIC-Start abfragen oder verändern.
2D-2E	45-46	Zeiger auf BASIC-Programmende Dieser Zeiger teilt dem Interpreter das BASIC-Ende mit, damit die Variablen hinter dem Programm abgelegt werden können.
2F-30	47-48	Zeiger auf Start der Arrays Das LOW- und HIGH-Byte der Adressen geben dem BASIC-Interpreter die Information, ab welcher Speicherzelle die Arrays eines BASIC-Programms gespeichert sind.
31-32	48-50	Zeiger auf Ende der Datenfelder Diese beiden Speicherzellen zeigen auf das Ende der Arrays. Zu beachten ist, daß die Zeichenketten rückwärts gespeichert werden.
33-34	51-52	Zeiger auf Stringgrenze Der Inhalt dieser Speicherzellen zeigt auf das Ende des Textspeichers, der aber noch zugleich das obere Ende des frei verfügbaren RAM-Bereichs anzeigt.
35-36	53-54	Hilfszeiger für Strings In diesen Zellen wird die Adresse der Zeichenkette verzeichnet, die als letzte von Routinen zur Stringmanipulation abgespeichert worden ist.

37-38	55-56	Zeiger auf BASIC-RAM-Ende Dieser Zeiger gibt dem Interpreter an, welches die höchste von BASIC verwendbare Speicheradresse ist.
39-3A	57-58	augenblickliche BASIC-Zeilennummer In diesen Speicherzellen wird die Zeilennummer verzeichnet, welche gerade ausgeführt wird.
3B-3C	59-60	Zeilennummer für CONT Falls eine Unterbrechung des Programmablaufs duch den Befehl STOP oder über die STOP-Taste erfolgt, wird in den Speicheradressen $3B-$3C die Zeilennummer gespeichert, die gerade abgearbeitet wurde.
3D-3E	61-62	Zeiger auf nächstes Statement für CONT Sobald eine neue BASIC-Zeile verarbeitet wird, holt sich das Betriebssystem die aktuelle Zeilennummer und speichert diese dann in $3D-$3E als LOW- und HIGH-Byte ab.
3F-40	63-64	augenblickliche Zeilennummer für DATA Diese beiden Speicherzellen enthalten die Zeilennummer einer DATA-Zeile, die gerade vom READ-Befehl ausgelesen wird.
41-42	65-66	Zeiger für nächstes DATA-Element Hier ist die Adresse aufgeführt, ab welcher der READ-Befehl nach der nächsten DATA-Zeile sucht.
43-44	67-68	Zeiger auf Herkunft der Eingabe Der Zeiger zeigt auf die jeweilige Adresse in diesem Eingabe-pufferspeicher.

45-46	69-70	Variablenname Falls während des Ablaufs eines Programms eine Variable auftaucht, wird deren Name hier zwischengespeichert.
47-48	71-72	Variablenadresse In diesen Speicherzellen wird der Zeiger auf den Variablenwert abgelegt.
49-4A	73-74	Zeiger auf Variablenelement Die Adresse einer Schleifenvariable wird zunächst hier gespeichert, bevor sie in den Stack gebracht wird.
4B-4C	75-76	Zwischenspeicher für Programmzeiger Diese Speicherzellen dienen als Zwischenspeicher für mathematische Operationen. Außerdem werden die Speicherzellen auch noch vom READ-Befehl als Zwischenspeicher verwendet.
4D	77	Maske für Vergleichsoperationen Dieser Zeiger wird von mathematischen Routinen als Vergleichsoperator verwendet, daß heißt um festzustellen, ob ein Wert kleiner, gleich oder größer ist.
4E-4F	78-79	Zeiger für FN In $4E-$4F ist die Adresse angegeben, wo die Variablen und ihr Wert abgelegt sind.
50-53	80-83	Stringdescriptor In diesen Speicherzellen wird unter anderem die Schrittweite für Garbage

		Collection und andere wichtige Informationen für den Interpreter festgelegt.
54	84	Konstante $4C JMP für Funktionen Hier ist die Konstante für JMP ($4C) festgelegt.
55-56	85-86	Sprungvektor für Funktionen In $55-$56 werden die Sprungvektoren für die Funktionen angegeben.
57-5B	87-91	Register für Arithmetik, Akku #3 Die Register werden für die Zwischenspeicherung von Polynomauswertungen (TAN) benötigt.
5C-60	92-96	Register für Arithmetik, Akku #4, siehe oben
61-65	97-101	Fließkommaakku #1, FAC Diese Register werden für die Berechnung von Fließkommazahlen verwendet.
66	102	Vorzeichen von FAC Der Zeiger gibt an, ob der Wert, der im FAC steht, positiv oder negativ ist.
67	103	Zähler für Polynomauswertung Diese Speicherzelle dient als Zähler für die Polynomauswertung.
68	104	Rundungsbyte für FAC Hier wird angegeben, ob der Wert, der im FAC steht, auf- oder abgerundet werden soll.

69-6D	105-109	Fließkommaakku#2, ARG Diese Register werden für die Berechnung von Fließkommazahlen verwendet.
6E	110	Vorzeichen von ARG Hier wird angegeben, ob der Wert, der im ARG steht, positiv oder negativ ist.
6F	111	Vergleichsbyte der Vorzeichen von FAC und ARG Diese Speicherzelle gibt dem Interpreter an, ob die Vorzeichen der beiden Akkus übereinstimmen.
71-72	113-114	Zeiger für Polynomauswertung Hier ist in LOW- und HIGH-Byte angegeben, was ausgewertet werden soll.
73-8A	115-138	CHRGET-Routine Diese Routine holt ein Zeichen aus dem BASIC-Text.
7A-7B	122-123	Programmzeiger In diesen Speicherzellen wird in LOW- und HIGH-Byte die Anfangsadresse des als nächstes auszuführenden Befehls im BASIC-RAM angegeben.
7C-8A	124-138	Unbenutzt
8B-8F	139-143	letzter RND-Wert In diesen Registern wird der letzte RND-Wert im Fließkommaformat abgelegt.
90	144	Statuswort ST In dieser Speicherzelle, die auch mit der BASIC-Variable ST identisch ist, sind die Fehlermeldungen der Datasette

und der Floppy verzeichnet:
Datasette:
Bit 0 = Unbenutzt
Bit 1 = Unbenutzt
Bit 2 = Kurzer Block
Bit 3 = Langer Block
Bit 4 = Lesefehler
Bit 5 = Prüfsummenfehler
Bit 6 = File-Ende
Bit 7 = Band-Ende
Floppy:
Bit 0 = Fehler beim Schreiben
Bit 1 = Fehler beim Lesen
Bit 2 = Unbenutzt
Bit 3 = Unbenutzt
Bit 4 = Unbenutzt
Bit 5 = Unbenutzt
Bit 6 = Daten-Ende
Bit 7 = DEVICE NOT PRESENT ERROR

91 145 Flag für STOP-Taste
In dieser Speicherzelle wird vermerkt, ob die Stoptaste gedrückt worden ist oder nicht.

92 146 Zeitkonstante für Band
Dieses Register hat die Aufgabe, kleine Unterschiede bei der Aufnahmegeschwindigkeit auszugleichen.

93 147 Flag für LOAD $00, oder für VERIFY $01
Dieses Flag dient dem Betriebssystem dazu, um zu unterscheiden, ob eine LOAD oder eine VERIFY Operation erfolgt.

94 148 Flag bei IEC-Ausgabe
Diese Adresse setzt bei Floppy und Drucker den "LISTEN" Zustand.

95	149	Ausgabepuffer für IEC-Bus Hier wird das Zeichen abgelegt, welches über den seriellen Port zur Floppy oder zum Drucker geschickt werden soll, sobald die Adresse $94 Bereitschaft zeigt.
96	150	Flag für EOT vom Band empfangen In $96 werden die Daten zwischengespeichert, die vom Band gelesen werden.
97	151	Zwischenspeicher für Register Beim Lesen von Band wird hier das X-Register zwischengespeichert.
98	152	Anzahl der offenen Files In dieser Speicherzelle wird festgehalten, wie viele Files gleichzeitig geöffnet sind.
99	153	aktives Eingabegerät In dieser Speicherzelle wird festgehalten, welches Gerät zur Eingabe verwendet werden soll. Die Nummern sind folgendermaßen festgelegt: 0 = Tastatur 1 = Datasette 2 = RS232 und User-Port 3 = Bildschirm 4-5 = Drucker 8-11 = Laufwerke
9A	154	aktives Ausgabegerät Diese Speicherzelle ist mit der vorherigen zu vergleichen, nur steht hier die Nummer des Geräts, über das die Ausgabe erfolgt.

9B	155	Parität für Band Über diese Speicherzelle findet eine Parity-Prüfung (Quersummenbildung) statt. Dies dient dazu, um Lese- und Schreibfehler zu vermeiden.
9C	156	Flag für Byte empfangen Hier wird festgelegt, ob das gelesene Byte die Quersumme richtig gebildet hat oder nicht.
9D	157	Flag für Direktmodus $80, Programm $00 In dieser Speicherzelle wird angegeben, welche Fehlermeldungen zugelassen werden und welche nicht. $00 unterdrückt alle Fehlermeldungen, $80 kommt dem normalen Eingabemodus gleich und $C0 läßt alle Fehlermeldungen zu. Diese Zustände können alle künstlich erzeugt werden.
9E	158	Bandpass 1 Checksumme Diese Speicherzelle wird zur Überprüfung von Bytes bei Kassettenoperationen benutzt.
9F	159	Bandpass 2 Fehlerkorrektur Hier werden die Fehler korrrigiert, die bei Kassettenoperationen aufgetreten sind.
A0-A2	160-162	Time In diesen Speicherzellen wird die Uhrzeit über die Interruptroutine erhöht.
A3	163	Bitzähler für serielle Ausgabe Dieses Register wird als Zwischenspeicher von Ein-/Ausgaberoutinen benutzt.

A4	164	Zähler für Band siehe oben
A5	165	Zähler für Band schreiben Diese Speicherzelle wird als Zähler des Synchron-Bits verwendet.
A6	166	Zeiger in Bandpuffer Dieses Register wird als Zähler benutzt, welcher angibt, wie viele Bytes aus dem Bandpuffer gelesen oder in den Bandpuffer geschrieben worden sind.
A7-AB	167-171	Arbeitsspeicher für Ein-/Ausgabe Diese Register werden häufig von Kassettenoperationen und der RS-232 Schnittstelle als Zwischenspeicher benutzt.
AC-AD	172-173	Zeiger für Bandpuffer und Scrolling Diese Speicherzellen dienen in erster Linie als Zeiger auf die Adresse, ab welcher ein Programm geladen oder gespeichert werden soll. Zweitens dienen sie auch als Zwischenspeicher während des Scrollings des Video-RAM und beim Einfügen zusätzlicher Zeilen.
AE-AF	174-175	Zeiger auf Programmende bei LOAD/SAVE Ähnlich wie $AC-$AD funktionieren diese beiden Speicherzellen. Sie Zeigen immer auf das Byte, das gerade gelesen oder gespeichert wurde.
B0-B1	176-177	Der Wert in den Speicherzellen wird benutzt, um die Zeitkonstante beim Lesen von Band einzustellen.

B2-B3	178-179	Zeiger auf Bandpuffer Diese beiden Speicherzellen zeigen auf den Bandpuffer ($033C)
B4	180	Bitzähler für Band Hier wird die Anzahl der übertragenden Bits gezählt.
B5	181	nächstes Bit für RS-232 Diese Speicherzelle enthält immer das nächste Bit, das bei RS-232 Operationen übertragen werden soll.
B6	182	Puffer für auszugebendes Byte Dieses Register wird als Ausgabezwischenspeicher benutzt.
B7	183	Länge des Filenamens In dieser Speicherzelle wird angegeben, aus wie vielen Zeichen der Filename besteht.
B8	184	logische Filenummer In dieser Speicherzelle wird die logische Filenummer verzeichnet.
B9	185	Sekundäradresse Hier steht die jeweilige Sekundäradresse.
BA	186	Gerätenummer Entsprechend ist auch in dieser Speicherzelle die Gerätenummer zu finden.
BB-BC	187-188	Zeiger auf Filenamen In diesen Speicherzellen steht ein Zeiger, der in LOW- und HIGH-Byte-Darstellung auf den Filenamen zeigt.

BD	189	Arbeitsspeicher serielle Ein-/Ausgabe Hier wird von den RS-232-Routinen ein Prüfbyte abgelegt (Parity-Prüfung).
BE	190	Passzähler für Band In dieser Speicherzelle ist angegeben, wie viele Blockteile von Band gelesen oder auf Band geschrieben werden sollen.
BF	191	Puffer für serielle Ausgabe Beim Laden eines Programms von Band wird diese Speicherzelle dazu benutzt, um die einzelnen Bits zu einem Byte zusammenzusetzen.
C0	192	Flag für Bandmotor Der Motor der Datasette kann nur eingeschaltet werden, wenn die Speicherzelle ungleich Null ist.
C1-C2	193-194	Startadresse für Ein-/Ausgabe In diesen Registern ist in LOW- und HIGH-Byte-Darstellung angegeben, ab welcher Adresse ein Programm geladen oder gespeichert wird.
C3-C4	195-196	Endadresse für Ein-/Ausgabe Hier steht in LOW- und HIGH-Byte der Zeiger auf den Tape-Header im Bandpuffer.
C5	197	Nummer der gedrückten Taste Hier wird die Nummer der gedrückten Taste gespeichert (64= keine Taste).
C6	198	Anzahl der gedrückten Tasten Hier steht die jeweilige Anzahl der Zeichen, die im Tastaturpuffer gespeichert sind.

C7	199	Flag für RVS-Modus Diese Speicherzelle gibt an, ob die auszugebenden Zeichen revers oder normal dargestellt werden sollen (0= normal, 1= revers).
C8	200	Zeilenende für Eingabe Dieses Register enthält die Position des letzen Zeichens in einer Zeile.
C9	201	Cursorzeile für Eingabe Diese Speicherzelle dient dazu, um die Zeile des letzten eingegebenen Zeichens festzustellen.
CA	202	Cursorspalte für Eingabe Diese Speicherzelle dient dazu, um die Spalte des letzten eingegebenen Zeichens festzustellen.
CB	203	gedrückte Taste Hier steht der jeweilige Code der gedrückten Taste. (64= keine Taste).
CC	204	Flag für Cursor Der Cursor wird ausgeschaltet, wenn in dieser Speicherzelle ein größerer Wert als Null steht.
CD	205	Zähler für Cursor blinken Diese Speicherzelle dient als Zähler für die Cursor-Blinkphase. Wenn der Wert 20 in dieser Speicherzelle abgezählt ist, wird der Cursor eingeschaltet.

CE	206	Zeichen unter dem Cursor Hier ist jeweils der Bildschirmcode eines Zeichens angegeben, das sich gerade unter dem Cursor befindet.
CF	207	Flag für Cursor In diesem Register wird festgehalten, in welcher Blink-Phase sich der Cursor gerade befindet.
D0	208	Flag für Eingabe von Tastatur oder Bildschirm Hier wird die Länge der zu übertragenden Zeichen gespeichert.
D1-D2	209-210	Zeiger auf Start der Bildschirmzeile In diesen Speicherzellen wird in LOW- und HIGH-Byte-Darstellung angezeigt, wo sich im Video-RAM die Zeile befindet, auf der der Cursor gerade steht.
D3	211	Cursorspalte Hier wird die Spaltenposition des Cursors festgehalten.
D4	212	Flag für Hochkommamodus Falls in dieser Speicherzelle eine Null steht, dann befindet sich der Computer im Hochkommamodus. Andere Werte bewirken den Normalmodus.
D5	213	Länge der Bildschirmzeile Der Inhalt dieser Speicherzelle entscheidet, ob eine neue Zeile angefangen werden muß oder nicht.
D6	214	Cursorzeile Hier wird die Zeilenposition des Cursors festgehalten.

D7	215	Speicher für ASCII-Tasten-Code Bevor ein Zeichen in den Tastaturpuffer gebracht wird, wird es vorher hier zwischengespeichert.
D8	216	Anzahl der Inserts Hier wird die Anzahl der Inserts festgelegt.
D9-F2	217-242	MSB der Bildschirmzeilenanfänge Alle 25 Speicherzellen enthalten Informationen über die Zeilen des Bildschirms.
F3-F4	243-244	Zeiger in Farb-RAM Diese Speicherzellen zeigen auf die Stelle im Farb-RAM, an der der Cursor auf der Zeile steht.
F5-F6	245-246	Zeiger auf Tastatur-Dekodiertabelle Diese Speicherzellen zeigen auf die Tastatur-Dekodiertabelle.
F7-F8	247-248	Zeiger auf RS-232 Eingabepuffer Diese Register zeigen auf die Anfangsadresse des Eingabepuffers.
F9-FA	249-250	Zeiger auf RS-232 Ausgabepuffer Diese Register zeigen auf die Anfangsadresse des Ausgabepuffers.
00FF-010A	255-266	Puffer für Umwandlung Fließkomma nach ASCII Diese Register werden für die Zwischenspeicherung von Fließkommazahlen benutzt.

0100-013E	256-318	Speicher für Korrektur bei Bandeingabe Beim Laden von Band werden hier die Daten zwischengespeichert, aus denen das Betriebssystem erkennen kann, welche Bytes fehlerhaft sind.
013F-01FF	256-511	Prozessorstack Der Stack ist generell ein Zwischenspeicher, in dem der Programmierer Daten ablegen kann. Außerdem wird er vom Prozessor dazu benutzt, bei einem Interrupt oder einem Unterprogrammaufruf die Adresse, von der aus verzweigt wurde, zwischenzuspeichern. Dies geschieht in der Reihenfolge HIGH- und LOW-Byte. Die Daten werden im Stack von der Adresse $01FF zur Adresse $0100 hin abgelegt. Bei einem BREAK wird zusätzlich der Prozessorstatus im Stack abgelegt.
0200-0258	512-600	BASIC-Eingabepuffer Nach der Eingabe eines Befehls oder einer Programmzeile werden diese Daten in diesen Bereich zwischengespeichert, um dann wieder weiterverarbeitet zu werden.
0259-0262	601-610	Tabelle der logischen Filenummern In dieser Tabelle werden die logischen Filenummern der Reihe nach, von 1-10, eingetragen. Beim Schließen einer Datei werden diese Einträge wieder entfernt.
0263-026C	611-620	Tabelle der Gerätenummern Diese Tabelle entspricht $0259-$0262, nur mit dem Unterschied, daß hier die Geräteadressen vermerkt werden.

026D-0276 621-630 Tabelle der Sekundäradressen
Diese Tabelle entspricht $0259-$0262, nur mit dem Unterschied, daß hier die Sekundäradressen vermerkt werden.

0277-0280 631-640 Tastaturpuffer
Hier werden die Tastencodes zwischengespeichert, die nicht sofort vom Betriebssystem weiterverarbeitet werden können.

0281-0282 641-642 Start des BASIC-RAM
Nach einem Reset oder einem Kaltstart wird dieser Zeiger auf den nächsten freien Speicherplatz gesetzt.

0283-0284 643-644 Ende des BASIC-RAM
Dieser Zeiger wird nach einem Reset oder einem Kaltstart auf den letzten verfügbaren freien RAM-Speicherplatz gesetzt.

0285 645 Timeout-Flag für seriellen IEC-Bus
Alle Zähler in dieser Speicherzelle, die größer als 128 sind, bedeuten, daß ein Gerät angeschlossen ist. Die kleineren Werte bedeuten das Gegenteil.

0286 646 augenblickliche Farbe
Hier wird die augenblickliche Zeichenfarbe festgelegt:
0 = schwarz
1 = weiß
2 = rot
3 = lila
4 = purpur
5 = grün
6 = blau

7 = gelb
8 = orange
9 = braun
10 = hellrot
11 = dunkelgrau
12 = mittelgrau
13 = hellgrün
14 = hellblau
15 = hellgrau

0287 647 Farbe unter dem Cursor
In dieser Speicherzelle merkt sich das Betriebssystem, welche Farbe gerade unter dem Cursor steht.

0288 648 HIGH-Byte Video-RAM
Dieses HIGH-Byte gibt dem Betriebssystem an, ab welcher Adresse das Video-RAM zu finden ist.

0289 649 Länge des Tastaturpuffers
Dieses Register gibt an, wie viele Speicherzellen des Tastaturpuffers belegt werden sollen.

028A 650 Flag für Repeatfunktion für alle Tasten
In dieser Speicherzelle wird dem Betriebssystem angegeben, welche Tasten eine Repeat-Funktion haben und welche nicht:
0 = nur Cursor-,Insert/Delete- und Leertaste
64 = keine Taste
128 = alle Tasten

028B 651 Zähler für Repeatgeschwindigkeit
Diese Speicherzelle dient als Zähler, die die Repeat-Geschwindigkeit festlegt.

028C	652	Zähler für Repeatverzögerung Hier wird angegeben, wie lange eine Taste gedrückt sein muß, bis die Repeat-Funktion einsetzt.
028D	653	Flag für SHIFT, Commodore und CTRL In diesem Register stehen die Tastencodes der Steuertasten: 1 = SHIFT 2 = Commodore 3 = SHIFT und Commodore 4 = CTRL 5 = SHIFT und CTRL 6 = Commodore und CTRL 7 = SHIFT, Commodore und CTRL
028E	654	SHIFT-Flag Hier steht die zuletzt gedrückte Steuertaste.
028F-0290	655-656	Zeiger für Tastatur-Dekodierung Hier steht ein Zeiger, der auf die Betriebssystemroutine für die Tastatur-Dekodierung zeigt.
0291	657	Flag für SHIFT/Commodore gesperrt Falls in der Speicherzelle eine 128 steht, wird eine Umschaltung mit SHIFT/Commodore verriegelt. Bei einer 0 wird die Umschaltung zugelassen.
0292	658	Flag für Scrollen Wenn in dieser Speicherzelle eine 0 steht, setzt der Scroll-Vorgang ein. Bei einem größeren Wert setzt dieser Vorgang nicht ein.

0293 659 RS-232 Kontrollwert

Hier wird die Übertragungs-geschwindigkeit der RS-232 Schnitt-stelle festgelegt:

Bits 0-3 steuern die Übertragungs-geschwindigkeit:

Bitmuster	Wert	Baudrate
0000	0	50 Baud
0001	1	50 Baud
0010	2	75 Baud
0011	3	110 Baud
0100	4	134.5 Baud
0101	5	150 Baud
0110	6	300 Baud
0111	7	600 Baud
1000	8	1200 Baud
1001	9	1800 Baud
1010	10	2400 Baud

Bit 4 nicht belegt

Die Bits 5 und 6 steuern die Länge der Übertragung:

Bitmuster	Wert	Länge
00	0	8-Bit
01	32	7-Bit
10	64	6-Bit
11	96	5-Bit

Bit 7 gibt die Anzahl der STOP-Bits an:

Bitmuster	Wert	Anzahl
0	0	1-STOP-Bit
1	128	2-STOP-Bits

0294 660 RS-232 Befehlswort

Die einzelnen Bits steuern das 'Handshake'-Protokoll.

0295-0296 661-662 Bit-Timing

Die Möglichkeit, eine freiwählbare Übertragungsgeschwindikeit einzu-stellen, wurde vorgesehen, aber nicht eingebaut.

0297 663 RS-232 Status
Hier werden die Fehlermeldungen der RS-232 Schnittstelle angezeigt:

Bit	Wert	Bedeutung
0	1	Fehler bei Parity-Prüfung
1	2	Fehler in der Bitfolge
2	4	Überlauf des Eingabepuffers
3	8	Eingabepuffer ist leer
4	16	das CTS-Signal fehlt
5	32	nicht belegt
6	64	Das DSR-Signal fehlt
7	128	Die Übertragung ist unterbrochen

0298 664 Anzahl der Datenbits für RS-232
Diese Speicherzelle wird verwendet, um die Wortlänge festzustellen.

0299-029A 665-666 RS-232 Baud-Rate
Die Übertragungsrate errechnet sich aus der Systemfrequenz (985.25) KHz dividiert duch die Baudrate.
Dieser Wert steht in LOW- und HIGH-Byte-Darstellung in den beiden Speicherzellen. Er wird vom Betriebssystem abgerufen.

029B 667 Zeiger für empfangenes Byte RS-232
Wenn man den Inhalt der Speicherzelle mit dem Wert in $F7-$F8 addiert, erhält man die Adresse des zuletzt im Eingabepuffer eingegebenen Bytes.

029C 668 Zeiger auf Input von RS-232
Wenn man den Inhalt der Speicherzelle mit dem Wert in $F7-$F8 addiert, erhält man die Adresse des ersten im Eingabepuffer eingegebenen Bytes.

029D	669	Zeiger auf zu Übertragendes Byte RS-232 Wenn man den Inhalt der Speicherzelle mit dem Wert in $F9-$FA addiert, erhält man die Adresse des ersten im Ausgabepuffer eingegebenen Bytes.
029E	670	Zeiger auf Ausgabe auf RS-232 Wenn man den Inhalt der Speicherzelle mit dem Wert in $F9-$FA addiert, erhält man die Adresse des zuletzt im Ausgabepuffer eingegebenen Bytes.
029F-02A0	671-672	Speicher für IRQ während Bandbetrieb Bei Kassettenoperationen wird hier in LOW- und HIGH-Byte-Darstellung der Vektor für die Interruptroutine gespeichert.
02A1	673	CIA 2 NMI-Flag Diese Speicherzelle erhält den Wert des Interruptsteuerregisters, das die RS-232 Schnittstelle steuert.
02A2	674	CIA 1 Timer A Bei Bandroutinen wird hier das HIGH-Byte von Timer A zwischengespeichert.
02A3	675	CIA 1 Interruptflag Bei Bandroutinen wird in dieser Speicherzelle festgelegt, welche IRQs freigegeben sind und welche nicht.
02A4	676	CIA 1 Flag für Timer A Hier wird bei Bandroutinen angegeben, ob Timer A läuft oder nicht. Wenn hier eine $00 steht, ist der Timer freigegeben, andernfalls ist er gesperrt.

02A5	677	Bildschirmzeile
02A6	678	Flag für PAL- (1) o. NTSC- Version (0) Hier steht ein Wert, der angibt, ob es sich um eine PAL- oder eine NTSC-Version handelt.
02C0-02FE	704-766	Sprite 11
0300-0301	768-769	$E38B Vektor für BASIC-Warmstart
0302-0303	770-771	$A483 Vektor für Eingabe einer Zeile
0304-0305	772-773	$A57C Vektor für Umwandlung in Interpretercode
0306-0307	774-775	$A71A Vektor für Umwandlung in Klartext (LIST)
0308-0309	776-777	$A7E4 Vektor für BASIC-Befehlsadresse holen
030A-030B	778-779	$AE86 Vektor für Ausdruck auswerten
030C	780	Akku für SYS-Befehl
030D	781	X-REG für SYS-Befehl
030E	782	Y-REG für SYS-Befehl
0310	783	Status-Register für SYS-Befehl
0311-0312	785-786	$B248 USR-Vektor
0314-0315	788-789	$EA31 IRQ-Vektor
0316-0317	790-791	$FE66 BRK-Vektor
0318-0319	792-793	$FE47 NMI-Vektor
031A-031B	794-795	$F34A OPEN-Vektor
031C-031D	796-797	$F291 CLOSE-Vektor
031E-031F	798-799	$F20E CHKIN-Vektor
0320-0321	800-801	$F250 CKOUT-Vektor
0322-0323	802-803	$F333 CLRCH-Vektor
0324-0325	804-805	$F157 INPUT-Vektor
0326-0327	806-807	$F1CA OUTPUT-Vektor
0328-0329	808-809	$F6ED STOP-Vektor
032A-032B	810-811	$F13E GET-Vektor
032B-032C	812-813	$F32F CLALL-Vektor
032E-032F	814-815	$FE66 Warmstart-Vektor
0330-0331	816-817	$F4A5 LOAD-Vektor
0332-0333	818-819	$F5ED SAVE-Vektor
033C-03FB	828-1019	Bandpuffer
0340-037E	832-894	Sprite 13

0380-03BE	896-958	Sprite 14
03C0-03FE	960-1022	Sprite 15

6.5 Die Speicheraufteilung des C64

Um die Speicherkonfiguration des C64 zu verändern, existieren die Bits 0, 1 und 2 der Speicherzelle $01, dem sogenannten Prozessorport. Zusätzlich gibt es noch zwei Leitungen, GAME und EXROM, die am Expansionsport ausgeführt sind. Wenn man diese Leitungen auf Masse legt, werden externe Speicherbereiche, zum Beispiel Module, eingeblendet. Mit den Bits des Prozessorports läßt sich lediglich die Verteilung von RAM und ROM im Speicher festlegen.

Den Zustand, den die jeweiligen Bits für die einzelnen Speicheraufteilungen haben müssen, entnehmen Sie bitte den nachfolgenden Bildern. Hierbei entsprechen die Bits des Prozessorports den folgenden Bezeichnungen:

Bit 1 = LORAM (LR)
Bit 2 = HIRAM (HR)
Bit 3 = CHAREN (CR)

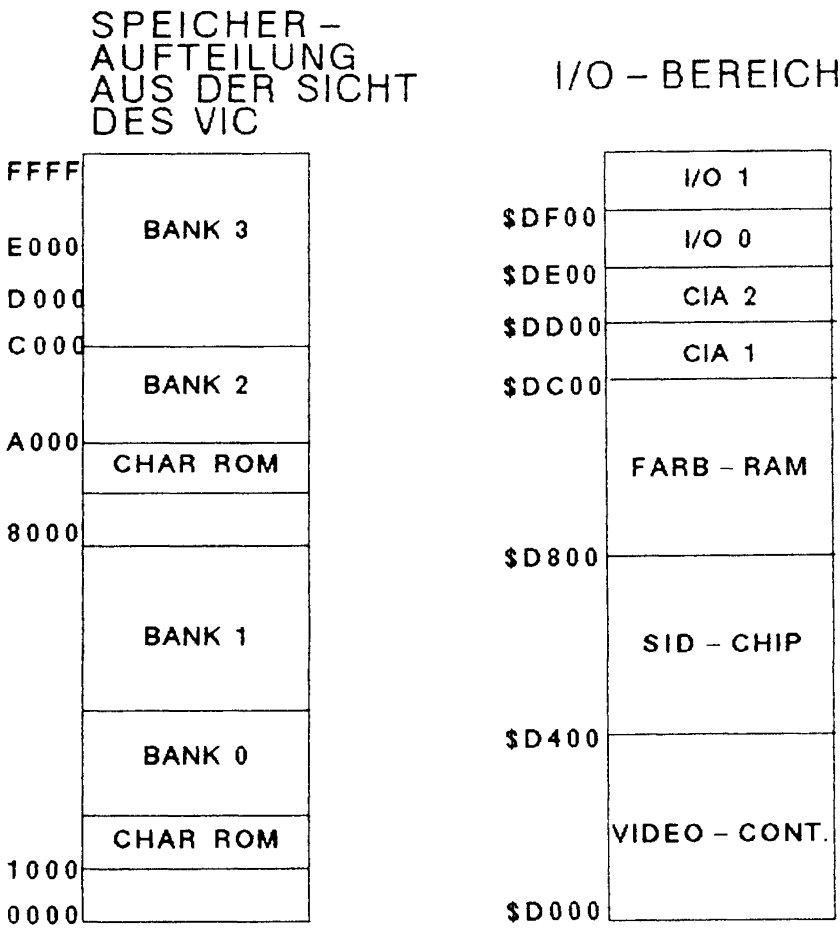


Abb. 6.5.1.1: Speicheraufteilung

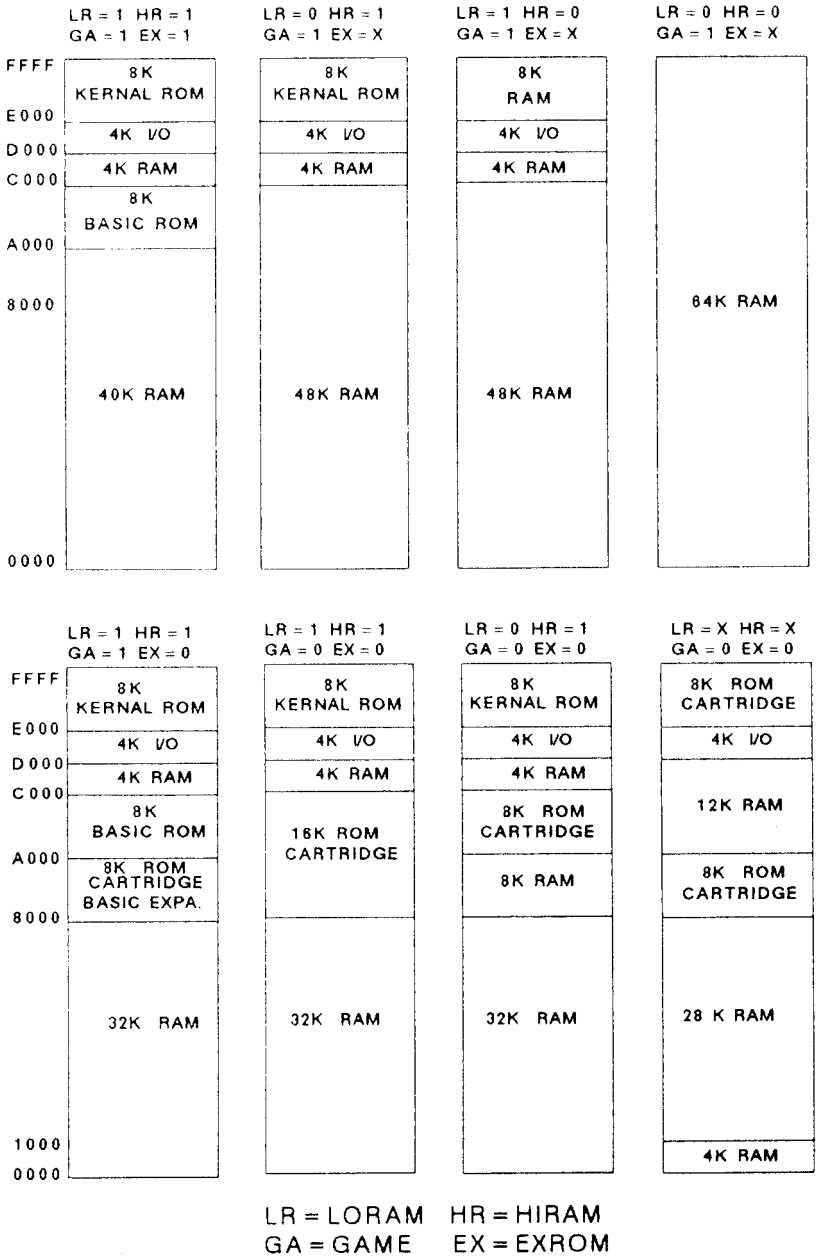


Abb. 6.5.1.2: Speicheraufteilung

6.5 Commodore 64 ROM-Listing

```
******************************
A000 94 E3                    Start-Vektor  $E394
A002 7B E3                    NMI-Vektor    $E37B

******************************
A004 43 42 4D 42 41 53 49 43  'cbmbasic'

****************************** Adressen der BASIC-Befehle -1
                              Interpreterkode Adresse Befehl
A00C 30 A8              $80     $A831   END
A00E 41 A7              $81     $A742   FOR
A010 1D AD              $82     $AD1E   NEXT
A012 F7 A8              $83     $A8F8   DATA
A014 A4 AB              $84     $ABA5   INPUT#
A016 BE AB              $85     $ABBF   INPUT
A018 80 B0              $86     $B081   DIM
A01A 05 AC              $87     $AC06   READ
A01C A4 A9              $88     $A9A5   LET
A01F 9F A8              $89     $A8A0   GOTO
A020 70 A8              $8A     $A871   RUN
A022 27 A9              $8B     $A928   IF
A024 1C A8              $8C     $A81D   RESTORE
A026 82 A8              $8D     $A883   GOSUB
A028 D1 A8              $8E     $A8D2   RETURN
A02A 3A A9              $8F     $A93B   REM
A02C 2E A8              $90     $A82F   STOP
A02F 4A A9              $91     $A94B   ON
A030 2C B8              $92     $B82D   WAIT
A032 67 E1              $93     $E168   LOAD
A034 55 E1              $94     $E156   SAVE
A036 64 E1              $95     $E165   VERIFY
A038 B2 B3              $96     $B3B3   DEF
A03A 23 B8              $97     $B824   POKE
A03C 7F AA              $98     $AA80   PRINT#
A03E 9F AA              $99     $AAA0   PRINT
A040 56 A8              $9A     $A857   CONT
A042 9B A6              $9B     $A69C   LIST
```

```
A044 5D A6              $9C     $A65E   CLR
A046 85 AA              $9D     $AA86   CMD
A048 29 E1              $9E     $E12A   SYS
A04A BD E1              $9F     $E1BE   OPEN
A04C C6 E1              $A0     $E1C7   CLOSE
A04E 7A AB              $A1     $AB7B   GET
A050 41 A6              $A2     $A642   NEW

****************************** Adressen der BASIC-Funktionen
A052 39 BC              $B4     $BC39   SGN
A054 CC BC              $B5     $BCCC   INT
A056 58 BC              $B6     $BC58   ABS
A058 10 03              $B7     $0310   USR
A05A 7D B3              $B8     $B37D   FRE
A05C 9E B3              $B9     $B39E   POS
A05E 71 BF              $BA     $BF71   SQR
A060 97 E0              $BB     $E097   RND
A062 EA B9              $BC     $B9EA   LOG
A064 ED BF              $BD     $BFED   EXP
A066 64 E2              $BE     $E264   COS
A068 6B E2              $BF     $E26B   SIN
A06A B4 E2              $C0     $E2B4   TAN
A06C 0E E3              $C1     $E30E   ATN
A06E 0D B8              $C2     $B80D   PEEK
A070 7C B7              $C3     $B77C   LEN
A072 65 B4              $C4     $B465   STR$
A074 AD B7              $C5     $B7AD   VAL
A076 8B B7              $C6     $B78B   ASC
A078 EC B6              $C7     $B6EC   CHR$
A07A 00 B7              $C8     $B700   LEFT$
A07C 2C B7              $C9     $B72C   RIGHT$
A07E 37 B7              $CA     $B737   MID$

****************************** Hierachiecodes und
                                Adressen-1 der Operatoren
A080 79 69 B8           $79, $B86A   Addition
A083 79 52 B8           $79, $B853   Subtraktion
A086 7B 2A BA           $7B, $BA2B   Multiplikation
A089 7B 11 BB           $7B, $BB12   Division
A08C 7F 7A BF           $7F, $BF7B   Potenzierung
```

```
A08F 50 E8 AF              $50, $AFE9   AND
A092 46 E5 AF              $46, $AFE6   OR
A095 7D B3 BF              $7D, $BFB4   Vorzeichenwechsel
A098 5A D3 AE              $5A, $AED4   NOT
A09B 64 15 B0              $64, $B016   Vergleich

****************************** BASIC-Befehlsworte
A09E 45 4E                    end
A0A0 C4 46 4F D2 4E 45 58 D4  for   next
A0A8 44 41 54 C1 49 4E 50 55  data  input#
A0B0 54 A3 49 4E 50 55 D4 44  input dim
A0B8 49 CD 52 45 41 C4 4C 45  read  let
A0C0 D4 47 4F 54 CF 52 55 CE  goto  run
A0C8 49 C6 52 45 53 54 4F 52  if    restore
A0D0 C5 47 4F 53 55 C2 52 45  gosub return
A0D8 54 55 52 CE 52 45 CD 53  rem   stop
A0E0 54 4F D0 4F CE 57 41 49  on    wait
A0E8 D4 4C 4F 41 C4 53 41 56  load  save
A0F0 C5 56 45 52 49 46 D9 44  verify def
A0F8 45 C6 50 4F 4B C5 50 52  poke  print#
A100 49 4E 54 A3 50 52 49 4E  print
A108 D4 43 4F 4E D4 4C 49 53  cont  list
A110 D4 43 4C D2 43 4D C4 53  clr   cmd   sys
A118 59 D3 4F 50 45 CE 43 4C  open  close
A120 4F 53 C5 47 45 D4 4E 45  get   new
A128 D7 54 41 42 A8 54 CF 46  tab(  to
A130 CE 53 50 43 A8 54 48 45  spc(  then
A138 CE 4E 4F D4 53 54 45 D0  not   stop
A140 AB AD AA AF DE 41 4E C4  + - * / ^  and
A148 4F D2 BE BD BC 53 47 CE  or <=> sgn
A150 49 4E D4 41 42 D3 55 53  int   abs   usr
A158 D2 46 52 C5 50 4F D3 53  fre   pos   sqr
A160 51 D2 52 4E C4 4C 4F C7  rnd   log
A168 45 58 D0 43 4F D3 53 49  exp   cos   sin
A170 CE 54 41 CE 41 54 CE 50  tan   atn   peek
A178 45 45 CB 4C 45 CE 53 54  len   str$
A180 52 A4 56 41 CC 41 53 C3  val   asc
A188 43 48 52 A4 4C 45 46 54  chr$  left$
A190 A4 52 49 47 48 54 A4 4D  right$   mid$
A198 49 44 A4 47 CF 00        go
```

```
****************************** BASIC-Fehlermeldungen
A19E 54 4F                        1  too many files
A1A0 4F 20 4D 41 4E 59 20 46
A1A8 49 4C 45 D3 46 49 4C 45      2  file open
A1B0 20 4F 50 45 CE 46 49 4C      3  file not open
A1B8 45 20 4E 4F 54 20 4F 50
A1C0 45 CE 46 49 4C 45 20 4E      4  file not found
A1C8 4F 54 20 46 4F 55 4E C4      5  device not present
A1D0 44 45 56 49 43 45 20 4E
A1D8 4F 54 20 50 52 45 53 45
A1E0 4E D4 4E 4F 54 20 49 4E      6  not input file
A1E8 50 55 54 20 46 49 4C C5
A1F0 4E 4F 54 20 4F 55 54 50      7  not output file
A1F8 55 54 20 46 49 4C C5 4D
A200 49 53 53 49 4E 47 20 46      8  missing filename
A208 49 4C 45 20 4E 41 4D C5
A210 49 4C 4C 45 47 41 4C 20      9  illegal device number
A218 44 45 56 49 43 45 20 4E
A220 55 4D 42 45 D2 4E 45 58      10 next without for
A228 54 20 57 49 54 48 4F 55
A230 54 20 46 4F D2 53 59 4E      11 syntax
A238 54 41 D8 52 45 54 55 52      12 return without gosub
A240 4E 20 57 49 54 48 4F 55
A248 54 20 47 4F 53 55 C2 4F      13 out of data
A250 55 54 20 4F 46 20 44 41
A258 54 C1 49 4C 4C 45 47 41      14 illegal quantity
A260 4C 20 51 55 41 4E 54 49
A268 54 D9 4F 56 45 52 46 4C      15 overflow
A270 4F D7 4F 55 54 20 4F 46      16 out of memory
A278 20 4D 45 4D 4F 52 D9 55      17 undef'd statement
A280 4E 44 45 46 27 44 20 53
A288 54 41 54 45 4D 45 4E D4
A290 42 41 44 20 53 55 42 53      18 bad subscript
A298 43 52 49 50 D4 52 45 44      19 redim'd array
A2A0 49 4D 27 44 20 41 52 52
A2A8 41 D9 44 49 56 49 53 49      20 division by zero
A2B0 4F 4E 20 42 59 20 5A 45
A2B8 52 CF 49 4C 4C 45 47 41      21 illegal direct
A2C0 4C 20 44 49 52 45 43 D4
```

```
A2C8 54 59 50 45 20 4D 49 53    22 type mismatch
A2D0 4D 41 54 43 C8 53 54 52    23 string too long
A2D8 49 4E 47 20 54 4F 4F 20
A2E0 4C 4F 4E C7 46 49 4C 45    24 file data
A2E8 20 44 41 54 C1 46 4F 52    25 formula too complex
A2F0 4D 55 4C 41 20 54 4F 4F
A2F8 20 43 4F 4D 50 4C 45 D8
A300 43 41 4E 27 54 20 43 4F    26 can't continue
A308 4E 54 49 4E 55 C5 55 4E    27 undef'd function
A310 44 45 46 27 44 20 46 55
A318 4E 43 54 49 4F CE 56 45    28 verify
A320 52 49 46 D9 4C 4F 41 C4    29 load

****************************** Adressen der Fehlermeldungen
A328 9E A1 AC A1 B5 A1 C2 A1
A330 D0 A1 E2 A1 F0 A1 FF A1
A338 10 A2 25 A2 35 A2 3B A2
A340 4F A2 5A A2 6A A2 72 A2
A348 7F A2 90 A2 9D A2 AA A2
A350 BA A2 C8 A2 D5 A2 E4 A2
A358 ED A2 00 A3 0E A3 1E A3
A360 24 A3 83 A3

****************************** Meldungen des Interpreters
A364 0D 4F 4B 0D                    OK
A368 00 20 20 45 52 52 4F 52        ERROR
A370 00 20 49 4E 20 00 0D 0A        IN
A378 52 45 41 44 59 2E 0D 0A        READY.
A380 00 0D 0A 42 52 45 41 4B        BREAK
A388 00 A0

****************************** Stapelsuch-Routine für
                                    FOR-NEXT- und GOSUB-Befehl

Einsprung von $A8D8, $A749, $AD2B

A38A   BA          TSX          Stapelzeiger in X-Register
A38B   E8          INX          4 mal erhöhen
A38C   E8          INX          (nächsten zwei Rücksprung-
A38D   E8          INX          adressen, Interpreter und
```

```
A38E   E8          INX          Routine, übergehen)
A38F   BD 01 01    LDA $0101,X  nächstes Byte holen
A392   C9 81       CMP #$81     Ist es FOR-Code ?
A394   D0 21       BNE $A3B7    Nein: dann RTS
A396   A5 4A       LDA $4A      Variablenzeiger holen
A398   D0 0A       BNE $A3A4    keine Variable (NEXT):$A3A4
A39A   BD 02 01    LDA $0102,X  Variablenzeiger aus
A39D   85 49       STA $49      Stapel nach $49/4A
A39F   BD 03 01    LDA $0103,X  (Variablenzeiger)
A3A2   85 4A       STA $4A      holen
A3A4   DD 03 01    CMP $0103,X  Mit Zeiger im Stapel vergl.
A3A7   D0 07       BNE $A3B0    Ungleich: nächste Schleife
A3A9   A5 49       LDA $49      Zeiger wieder holen
A3AB   DD 02 01    CMP $0102,X  Mit Zeiger im Stapel vergl.
A3AE   F0 07       BEQ $A3B7    Gleich: Schleife gefunden,RTS
A3B0   8A          TXA          Suchzeiger in Akku
A3B1   18          CLC          Carry für Addition löschen
A3B2   69 12       ADC #$12     Suchzeiger um 18 erhöhen
A3B4   AA          TAX          und wieder zurück ins X-Rg.
A3B5   D0 D8       BNE $A38F    nächste Schleife prüfen
A3B7   60          RTS          Rücksprung

****************************** Block-Verschiebe-Routine

Einsprung von $A749, $B15D

A3B8   20 08 A4    JSR $A408    prüft auf Platz im Speicher
A3BB   85 31       STA $31      Ende des Arraybereichs
A3BD   84 32       STY $32      als Beginn für freien Platz

Einsprung von $B628

A3BF   38          SEC          Carry löschen (Subtraktion)
A3C0   A5 5A       LDA $5A      Startadresse von Endad. des
A3C2   E5 5F       SBC $5F      Bereichs abziehen (LOW)
A3C4   85 22       STA $22      Ergebnis (=Länge) speichern
A3C6   A8          TAY          Gleiches System für  HIGH:
A3C7   A5 5B       LDA $5B      Altes Blockende (HIGH) und
A3C9   E5 60       SBC $60      davon alter Blockanfang sub
A3CB   AA          TAX          Länge nach X bringen
```

```
A3CC  E8        INX          Ist ein Rest ( Länge nicht
A3CD  98        TYA          256 Bytes)?
A3CE  F0 23     BEQ $A3F3    Nein: dann nur ganze Blöcke
A3D0  A5 5A     LDA $5A      Alte Endadresse (LOW) und
A3D2  38        SEC          davon Länge des Restab-
A3D3  E5 22     SBC $22      schnitts subtrahieren ergibt
                             Adresse des
A3D5  85 5A     STA $5A      Restabschnitts
A3D7  B0 03     BCS $A3DC    Berechnung für HIGH umgehen
A3D9  C6 5B     DEC $5B      Dasselbe System für HIGH
A3DB  38        SEC          Carry setzen (Subtraktion)
A3DC  A5 58     LDA $58      Alte Endadresse (HIGH) und
A3DE  E5 22     SBC $22      davon Länge des Rests sub-
A3E0  85 58     STA $58      trahieren ergibt neue Adresse
A3E2  B0 08     BCS $A3EC    Unbedingter Sprung zur
A3E4  C6 59     DEC $59      Kopierroutine für ganze
A3E6  90 04     BCC $A3EC    Blöcke
A3E8  B1 5A     LDA ($5A),Y  Kopierroutine für Rest-
A3EA  91 58     STA ($58),Y  abschnitt
A3EC  88        DEY          Zähler vermindern
A3ED  D0 F9     BNE $A3E8    Alles? wenn nicht: weiter
A3EF  B1 5A     LDA ($5A),Y  Kopierroutine für ganze
A3F1  91 58     STA ($58),Y  Blöcke
A3F3  C6 5B     DEC $5B      Adresszähler vermindern
A3F5  C6 59     DEC $59      Adresszähler vermindern
A3F7  CA        DEX          Zähler vermindern
A3F8  D0 F2     BNE $A3EC    Alles? Wenn nicht: weiter
A3FA  60        RTS          sonst Rücksprung

****************************** Prüfung auf Platz im Stapel

Einsprung von $A885 ,$ADAE

A3FB  0A        ASL          Akku muß die halbe Zahl an
A3FC  69 3E     ADC #$3E     erforderlichem Platz haben
A3FE  B0 35     BCS $A435    gibt 'OUT OF MEMORY'
A400  85 22     STA $22      Wert merken
A402  BA        TSX          Ist Stapelzeiger kleiner
A403  E4 22     CPX $22      (2 * Akku + 62)?
```

```
A405   90 2E      BCC $A435   Wenn ja, dann OUT OF MEMORY
A407   60         RTS         Rücksprung

****************************** Schafft Platz im Speicher

Einsprung von $A3B8, $B264, $B2B9, $E426

A408   C4 34      CPY $34     für Zeileneinfügung
A40A   90 28      BCC $A434   und Variablen
A40C   D0 04      BNE $A412   A/Y = Adresse, bis zu der
A40E   C5 33      CMP $33     Platz benötigt wird.
A410   90 22      BCC $A434   Kleiner als Stringzeiger
A412   48         PHA         Akku zwischenspeichern
A413   A2 09      LDX #$09    Zähler setzen
A415   98         TYA         Y-Register auf
A416   48         PHA         Stapel retten
A417   B5 57      LDA $57,X   Ab $57 zwischenspeichern
A419   CA         DEX         Zähler vermindern
A41A   10 FA      BPL $A416   Alle? sonst weiter
A41C   20 26 B5   JSR $B526   Garbage Collection
A41F   A2 F7      LDX #$F7    Zähler setzen, um
A421   68         PLA         Akku, Y-Register und andere
A422   95 61      STA $61,X   Register zurückholen
A424   E8         INX         Zähler vermindern
A425   30 FA      BMI $A421   Fertig? Nein, dann weiter
A427   68         PLA         Y-Register von Stapel
A428   A8         TAY         zurückholen
A429   68         PLA         Akku holen
A42A   C4 34      CPY $34     Ist jetzt genügend Platz?
A42C   90 06      BCC $A434   Ja, dann Rücksprung
A42E   D0 05      BNE $A435   kein Platz, dann Fehler-
A430   C5 33      CMP $33     meldung ' out of memory '
A432   B0 01      BCS $A435   ausgeben
A434   60         RTS         Rücksprung

Einsprung von $B30B

A435   A2 10      LDX #$10    Fehlernummer 'out of memory'
```

```
****************************** Fehlereinsprung

Einsprung von $A573, $A85F, $A8E5, $AB68, $AD32, $AD9B
$AF0A, $B24A, $B3B0, $B4D2, $B65A, $B980, $BB8C, $E109
$E19E

A437   6C 00 03   JMP ($0300) Zum BASIC-Warmstart ($E38B)

****************************** Fehlermeldung ausgeben

Einsprung von $E38E

A43A   8A         TXA         Fehlernummer im X-Register
A43B   0A         ASL         Akku * 2
A43C   AA         TAX         Akku als Zeiger nach X
A43D   BD 26 A3   LDA $A326,X und Adresse der
A440   85 22      STA $22     Fehlernummer aus Tabelle
A442   BD 27 A3   LDA $A327,X holen und
A445   85 23      STA $23     abspeichern
A447   20 CC FF   JSR $FFCC   I/O Kanäle zurücksetzen
A44A   A9 00      LDA #$00    und Eingabekanal auf
A44C   85 13      STA $13     Tastatur setzen
A44E   20 D7 AA   JSR $AAD7   (CR) und (LF) ausgeben
A451   20 45 AB   JSR $AB45   '?' ausgeben
A454   A0 00      LDY #$00    Zeiger setzen
A456   B1 22      LDA ($22),Y Fehlermeldungstext holen
A458   48         PHA         Akku retten
A459   29 7F      AND #$7F    Bit 7 löschen und
A45B   20 47 AB   JSR $AB47   Fehlermeldung ausgeben
A45E   C8         INY         Zähler vermindern
A45F   68         PLA         Akku zurückholen
A460   10 F4      BPL $A456   Fertig? Nein, dann weiter
A462   20 7A A6   JSR $A67A   BASIC-Zeiger initialisieren
A465   A9 69      LDA #$69    Zeiger A/Y auf Error-
A467   A0 A3      LDY #$A3    meldung stellen

Einsprung von $A851

A469   20 1E AB   JSR $AB1E   String ausgeben
A46C   A4 3A      LDY $3A     Auf Programmodus
```

```
A46E   C8          INY          (prog/direkt) prüfen
A46F   F0 03       BEQ $A474    Direkt: dann ausgeben
A471   20 C2 BD    JSR $BDC2    'in Zeilennummer' ausgeben

Einsprung von $E391

A474   A9 76       LDA #$76     Zeiger auf Ready-Modus
A476   A0 A3       LDY #$A3     setzen und
A478   20 1E AB    JSR $AB1E    String ausgeben
A47B   A9 80       LDA #$80     Wert für Direktmodus laden
A47D   20 90 FF    JSR $FF90    und Flag setzen

****************************** Eingabe-Warteschleife

Einsprung von $A530

A480   6C 02 03    JMP ($0302) JMP $A483

Einsprung von $A480

A483   20 60 A5    JSR $A560    BASIC-Zeile nach
                                Eingabepuffer
A486   86 7A       STX $7A      CHRGET Zeiger auf
A488   84 7B       STY $7B      Eingabepuffer
A48A   20 73 00    JSR $0073    nächstes Zeichen holen
A48D   AA          TAX          Puffer leer?
A48E   F0 F0       BEQ $A480    Ja: dann weiter warten
A490   A2 FF       LDX #$FF     Wert für
A492   86 3A       STX $3A      Kennzeichen für Direktmodus
A494   90 06       BCC $A49C    Ziffer? als Zeile einfügen
A496   20 79 A5    JSR $A579    BASIC-Zeile in Code wandeln
A499   4C E1 A7    JMP $A7E1    Befehl ausführen

****************************** Löschen und Einfügen von
                                Programmzeilen
A49C   20 6B A9    JSR $A96B    Zeilenr. nach Adressformat
A49F   20 79 A5    JSR $A579    BASIC-Zeile in Code wandeln
A4A2   84 0B       STY $0B      Zeiger in Eingabepuffer
A4A4   20 13 A6    JSR $A613    Zeilenadresse berechnen
A4A7   90 44       BCC $A4ED    Vorhanden? Ja: löschen
```

```
****************************** Programmzeile löschen
A4A9   A0 01     LDY #$01     Zeiger setzen
A4AB   B1 5F     LDA ($5F),Y  Startadresse der nächsten
A4AD   85 23     STA $23      Zeile (HIGH) setzen
A4AF   A5 2D     LDA $2D      Variablenanfangszeiger
A4B1   85 22     STA $22      (LOW) setzen
A4B3   A5 60     LDA $60      Startadresse der zu
A4B5   85 25     STA $25      löschenden Zeile (HIGH)
A4B7   A5 5F     LDA $5F      Startadresse der zu
A4B9   88        DEY          löschenden Zeile (LOW)
A4BA   F1 5F     SBC ($5F),Y  Startadresse der nächsten
A4BC   18        CLC          Zeile (LOW)
A4BD   65 2D     ADC $2D      Variablenanfangszeiger (LOW)
A4BF   85 2D     STA $2D      ergibt neuen Variablenan-
A4C1   85 24     STA $24      fangszeiger (LOW)
A4C3   A5 2E     LDA $2E      Gleiches System für
A4C5   69 FF     ADC #$FF     HIGH-Byte des Variablenan-
A4C7   85 2E     STA $2E      fangszeigers
A4C9   E5 60     SBC $60      minus Startadresse der zu
A4CB   AA        TAX          löschenden Zeile (LOW) ergibt
A4CC   38        SEC          die zu verschiebenden Blöcke
A4CD   A5 5F     LDA $5F      Startadresse (LOW) minus
A4CF   E5 2D     SBC $2D      Variablenanfangszeiger (LOW)
A4D1   A8        TAY          ergibt Länge des Restabschn.
A4D2   B0 03     BCS $A4D7    Größer als 255? Nein: $A4D7
A4D4   E8        INX          Zähler für Blöcke erhöhen
A4D5   C6 25     DEC $25      Transportzeiger vermindern
A4D7   18        CLC          Carry löschen
A4D8   65 22     ADC $22      Anfangszeiger (LOW)
A4DA   90 03     BCC $A4DF    Verminderung überspringen
A4DC   C6 23     DEC $23      Zeiger um 1 vermindern
A4DE   18        CLC          Carry löschen
A4DF   B1 22     LDA ($22),Y  Verschiebeschleife
A4E1   91 24     STA ($24),Y  Wert abspeichern
A4E3   C8        INY          Zähler um 1 erhöhen
A4E4   D0 F9     BNE $A4DF    Block fertig? Nein: weiter
A4E6   E6 23     INC $23      1.Adreßzeiger erhöhen (LOW)
A4E8   E6 25     INC $25      2.Adreßzeiger erhöhen (LOW)
```

```
A4EA   CA         DEX          Blockzähler um 1 vermindern
A4EB   D0 F2      BNE $A4DF    Alle Blöcke? Nein: weiter

**************************** Programmzeile einfügen
A4ED   20 59 A6   JSR $A659    CLR-Befehl
A4F0   20 33 A5   JSR $A533    Programmzeilen neu binden
A4F3   AD 00 02   LDA $0200    Zeichen im Puffer ?
A4F6   F0 88      BEQ $A480    nein, dann zur Warteschleife
A4F8   18         CLC          Carry löschen
A4F9   A5 2D      LDA $2D      Variablenanfangszeiger (LOW)
A4FB   85 5A      STA $5A      als Endadresse (Quellbereich)
A4FD   65 0B      ADC $0B      + Länge der Zeile als End-
A4FF   85 58      STA $58      adresse des Zielbereichs LOW
A501   A4 2E      LDY $2E      Variablenanfangszeiger als
A503   84 5B      STY $5B      Endadr. des Quellbereichs LOW
A505   90 01      BCC $A508    Kein Übertrag? dann $A508
A507   C8         INY          Übertrag addieren
A508   84 59      STY $59      Als Endadresse
                                des Zielbereichs
A50A   20 B8 A3   JSR $A3B8    BASIC-Zeilen verschieben
A50D   A5 14      LDA $14      Zeilennummer aus
A50F   A4 15      LDY $15      $14/15 vor
A511   8D FE 01   STA $01FE    BASIC-Eingabepuffer setzen
A514   8C FF 01   STY $01FF    (ab $0200)
A517   A5 31      LDA $31      Neuer Variablen-
A519   A4 32      LDY $32      endzeiger
A51B   85 2D      STA $2D      als Zeiger auf Programm-
A51D   84 2E      STY $2E      ende speichern
A51F   A4 0B      LDY $0B      Zeilenlänge holen
A521   88         DEY          und um 1 vermindern
A522   B9 FC 01   LDA $01FC,Y Zeile aus Eingabepuffer
A525   91 5F      STA ($5F),Y ins Programm kopieren
A527   88         DEY          Schon alle Zeichen?
A528   10 F8      BPL $A522    Nein: dann weiterkopieren

Einsprung von $E1B2

A52A   20 59 A6   JSR $A659    CLR-Befehl
A52D   20 33 A5   JSR $A533    Programmzeilen neu binden
A530   4C 80 A4   JMP $A480    zur Eingabe-Warteschleife
```

```
**************************** BASIC-Zeilen neu binden

Einsprung von $A4F0, $A52D, $E1B8

A533   A5 2B      LDA $2B     Zeiger auf BASIC-Programm-

A535   A4 2C      LDY $2C     start holen und
A537   85 22      STA $22     und als Suchzeiger nach
A539   84 23      STY $23     $22/23 speichern
A53B   18         CLC         Carry löschen
A53C   A0 01      LDY #$01    Zeiger laden
A53E   B1 22      LDA ($22),Y Zeilenadresse holen
A540   F0 1D      BEQ $A55F   =0? Ja: dann RTS
A542   A0 04      LDY #$04    Zeiger auf erstes BASIC-
A544   C8         INY         zeichen setzen
A545   B1 22      LDA ($22),Y Zeichen holen
A547   D0 FB      BNE $A544   =0? (Zeilenende) nein: weiter
A549   C8         INY         Zeilenlänge nach
A54A   98         TYA         Akku schieben
A54B   65 22      ADC $22     + Zeiger auf aktuelle Zeile
A54D   AA         TAX         (LOW) ins X-Register
A54E   A0 00      LDY #$00    Zeiger laden
A550   91 22      STA ($22),Y Akku als Adr.zeiger (LOW)
A552   A5 23      LDA $23     Zeiger auf aktuelle
                               Zeile (HIGH)
A554   69 00      ADC #$00    Übertrag addieren
A556   C8         INY         Zähler um 1 erhöhen
A557   91 22      STA ($22),Y Adresszeiger (HIGH) speichern
A559   86 22      STX $22     Startadresse der nächsten
A55B   85 23      STA $23     Zeile abspeichern
A55D   90 DD      BCC $A53C   Zum Zeilenanfang
A55F   60         RTS         Rücksprung

**************************** Eingabe einer Zeile

Einsprung von $A483, $AC03

A560   A2 00      LDX #$00    Zeiger setzen
A562   20 12 E1   JSR $E112   ein Zeichen holen
```

```
A565   C9 0D      CMP #$0D     RETURN-Taste ?
A567   F0 0D      BEQ $A576    ja, dann Eingabe beenden
A569   9D 00 02   STA $0200,X Zeichen nach Eingabepuffer
A56C   E8         INX          Zeiger um 1 erhöhen
A56D   E0 59      CPX #$59     89. Zeichen  ?
A56F   90 F1      BCC $A562    nein, weitere Zeichen holen
A571   A2 17      LDX #$17     Nummer für 'string too long'
A573   4C 37 A4   JMP $A437    Fehlermeldung ausgeben
A576   4C CA AA   JMP $AACA    Puffer mit $0 abschließen, CR

**************************** Umwandlung einer Zeile in den
                              Interpreter-Code

Einsprung von $A496, $A49F

A579   6C 04 03   JMP ($0304) JMP $A57C

Einsprung von $A579

A57C   A6 7A      LDX $7A      Zeiger setzen, erstes Zeichen
A57E   A0 04      LDY #$04     Wert für codierte Zeile
A580   84 0F      STY $0F      Flag für Hochkomma
A582   BD 00 02   LDA $0200,X Zeichen aus Puffer holen
A585   10 07      BPL $A58E    kein BASIC-Code ? kleiner 128
A587   C9 FF      CMP #$FF     Code für Pi ?
A589   F0 3E      BEQ $A5C9    Ja: dann speichern
A58B   E8         INX          Zeiger erhöhen
A58C   D0 F4      BNE $A582    nächstes Zeichen überprüfen
A58E   C9 20      CMP #$20     ' ' Leerzeichen?
A590   F0 37      BEQ $A5C9    Ja: dann speichern
A592   85 08      STA $08      in Hochkomma-Flag speichern
A594   C9 22      CMP #$22     '"' Hochkomma?
A596   F0 56      BEQ $A5EE    Ja: dann speichern
A598   24 0F      BIT $0F      Überprüft auf Bit 6
A59A   70 2D      BVS $A5C9    gesetzt: ASCII speichern
A59C   C9 3F      CMP #$3F     '?' Fragezeichen?
A59E   D0 04      BNE $A5A4    Nein: dann weiter prüfen
A5A0   A9 99      LDA #$99     PRINT-Code für ? laden
A5A2   D0 25      BNE $A5C9    und abspeichern
A5A4   C9 30      CMP #$30     Kleiner $30 ? (Code für 0)
```

```
A5A6  90 04     BCC $A5AC   Ja: dann $A5AC
A5A8  C9 3C     CMP #$3C    Mit $3C vergleichen
A5AA  90 1D     BCC $A5C9   wenn größer, dann $A5C9
A5AC  84 71     STY $71     Zeiger zwischenspeichern
A5AE  A0 00     LDY #$00    Zähler für Tokentabelle
A5B0  84 0B     STY $0B     initialisieren
A5B2  88        DEY
A5B3  86 7A     STX $7A     Zeiger auf Eingabepuffer
A5B5  CA        DEX         zwischenspeichern
A5B6  C8        INY         X- und Y-Register
A5B7  E8        INX         um 1 erhöhen
A5B8  BD 00 02  LDA $0200,X Zeichen aus Puffer laden
A5BB  38        SEC         Carry für Subtr. löschen
A5BC  F9 9E A0  SBC $A09E,Y Zeichen mit Befehlswort vergleichen
A5BF  F0 F5     BEQ $A5B6   Gefunden? Ja: nächstes Zeich.
A5C1  C9 80     CMP #$80    mit $80 (128) vergleichen
A5C3  D0 30     BNE $A5F5   Befehl nicht gefunden: $A5F5
A5C5  05 0B     ORA $0B     BASIC-Code gleich Zähler +$80
A5C7  A4 71     LDY $71     Zeiger auf cod. Zeile holen
A5C9  E8        INX
A5CA  C8        INY         Zeiger erhöhen
A5CB  99 FB 01  STA $01FB,Y BASIC-Code speichern
A5CE  B9 FB 01  LDA $01FB,Y und Statusregister setzen
A5D1  F0 36     BEQ $A609   =0 (Ende): dann fertig
A5D3  38        SEC         Carry setzen (Subtraktion)
A5D4  E9 3A     SBC #$3A    ':' Trennzeichen?
A5D6  F0 04     BEQ $A5DC   Ja: dann $A5DC
A5D8  C9 49     CMP #$49    DATA-Code ?
A5DA  D0 02     BNE $A5DE   Nein: Speichern überspringen
A5DC  85 0F     STA $0F     nach Hochkomma-Flag speichern
A5DE  38        SEC         Carry setzen
A5DF  E9 55     SBC #$55    REM-Code ?
A5E1  D0 9F     BNE $A582   Nein: zum Schleifenanfang
A5E3  85 08     STA $08     0 in Hochkomma-Flag
A5E5  BD 00 02  LDA $0200,X nächstes Zeichen holen
A5E8  F0 DF     BEQ $A5C9   =0 (Ende)? Ja: dann $A5C9
A5EA  C5 08     CMP $08     Als ASCII speichern?
A5EC  F0 DB     BEQ $A5C9   Nein: dann $A5C9
A5EE  C8        INY         Zeiger erhöhen
A5EF  99 FB 01  STA $01FB,Y Code abspeichern
```

```
A5F2  E8         INX          Zeiger erhöhen
A5F3  D0 F0      BNE $A5E5    Zum Schleifenanfang
A5F5  A6 7A      LDX $7A      Zeiger wieder auf Eingabep.
A5F7  E6 0B      INC $0B      Suchzähler erhöhen
A5F9  C8         INY          Zähler erhöhen
A5FA  B9 9D A0   LDA $A09D,Y nächsten Befehl suchen
A5FD  10 FA      BPL $A5F9    Gefunden? Nein: weitersuchen
A5FF  B9 9E A0   LDA $A09E,Y Ende der Tabelle?
A602  D0 B4      BNE $A5B8    Nein: dann weiter
A604  BD 00 02   LDA $0200,X nächstes Zeichen holen
A607  10 BE      BPL $A5C7    kleiner $80? Ja: $A5C7
A609  99 FD 01   STA $01FD,Y im Eingabepuffer speichern
A60C  C6 7B      DEC $7B      CHRGET-Zeiger zurücksetzen
A60E  A9 FF      LDA #$FF     Zeiger auf Eingabepuffer -1
A610  85 7A      STA $7A      setzen (LOW)
A612  60         RTS          Rücksprung

**************************** Startadresse einer
                              Programmzeile berechnen

Einsprung von $A4A4, $A6A7

A613  A5 2B      LDA $2B      Zeiger auf BASIC-
A615  A6 2C      LDX $2C      Programmstart laden

Einsprung von $A8C0

A617  A0 01      LDY #$01     Zähler setzen
A619  85 5F      STA $5F      BASIC-Programmstart als
A61B  86 60      STX $60      Zeiger nach $5F/60
A61D  B1 5F      LDA ($5F),Y Link-Adresse holen (HIGH)
A61F  F0 1F      BEQ $A640    gleich null: dann Ende
A621  C8         INY          Zähler 2 mal erhöhen ( LOW-
A622  C8         INY          Byte übergehen)
A623  A5 15      LDA $15      gesuchte Zeilennummer (HIGH)
A625  D1 5F      CMP ($5F),Y mit aktueller vergleichen
A627  90 18      BCC $A641    kleiner: dann nicht gefunden
A629  F0 03      BEQ $A62E    gleich: Nummer LOW prüfen
A62B  88         DEY          Zähler um 1 vermindern
A62C  D0 09      BNE $A637    unbedingter Sprung
```

```
A62E   A5 14      LDA $14      gesuchte Zeilennummer (LOW)
A630   88         DEY          Zeiger um 1 vermindern
A631   D1 5F      CMP ($5F),Y  Zeilennummer LOW vergleichen
A633   90 0C      BCC $A641    kleiner: Zeile nicht gefunden
A635   F0 0A      BEQ $A641    oder gleich: C=1 und RTS
A637   88         DEY          Y-Register auf 1 setzen
A638   B1 5F      LDA ($5F),Y  Adresse der nächsten Zeile
A63A   AA         TAX          in das X-Register laden
A63B   88         DEY          Register vermindern (auf 0)
A63C   B1 5F      LDA ($5F),Y  Link-Adresse holen (LOW)
A63E   B0 D7      BCS $A617    weiter suchen
A640   18         CLC          Carry löschen
A641   60         RTS          Rücksprung

****************************  BASIC-Befehl  NEW
A642   D0 FD      BNE $A641    Kein Trennzeichen: SYNTAX
                                ERROR

Einsprung von $E444

A644   A9 00      LDA #$00     Nullcode laden
A646   A8         TAY          und als Zähler ins Y-Reg.
A647   91 2B      STA ($2B),Y  Nullcode an Programmanfang
A649   C8         INY          Zähler erhöhen
A64A   91 2B      STA ($2B),Y  noch einen Nullcode dahinter
A64C   A5 2B      LDA $2B      Zeiger auf Programmst. (LOW)
A64E   18         CLC          Carry löschen
A64F   69 02      ADC #$02     Programmstart + 2 ergibt
A651   85 2D      STA $2D      neuen Variablenstart (LOW)
A653   A5 2C      LDA $2C      Zeiger auf Programmst. (HIGH)
A655   69 00      ADC #$00     + Übertrag ergibt neuen
A657   85 2E      STA $2E      Variablenstart (HIGH)

Einsprung von $A4ED, $A52A, $A87A

A659   20 8E A6   JSR $A68E   CHRGET, Routine neu setzen
A65C   A9 00      LDA #$00     Zero-Flag für CLR = 1 setzen
```

```
**************************** BASIC-Befehl  CLR
A65E   D0 2D      BNE $A68D   Kein Trennzeichen: SYNTAX
                              ERROR

Einsprung von $A87D

A660   20 E7 FF   JSR $FFE7   alle I/O Kanäle zurücksetzen

Einsprung von $E101

A663   A5 37      LDA $37     Zeiger auf BASIC-RAM-Ende
A665   A4 38      LDY $38     (LOW/HIGH) laden
A667   85 33      STA $33     String-Start auf BASIC-
A669   84 34      STY $34     RAM-Ende setzen
A66B   A5 2D      LDA $2D     Zeiger auf Variablen-
A66D   A4 2E      LDY $2E     start laden
A66F   85 2F      STA $2F     und in Array-Anfangs-
A671   84 30      STY $30     zeiger setzen
A673   85 31      STA $31     und in Zeiger auf Array-
A675   84 32      STY $32     Ende speichern

Einsprung von $E1BB

A677   20 1D A8   JSR $A81D   RESTORE-Befehl

Einsprung von $A462, $E382

A67A   A2 19      LDX #$19    Wert laden und String-
A67C   86 16      STX $16     Descriptor-Index zurücksetzen
A67E   68         PLA         2 Bytes vom Stapel in das
A67F   A8         TAY         Y-Register und den
A680   68         PLA         Akku holen
A681   A2 FA      LDX #$FA    Wert laden und damit
A683   9A         TXS         Stapelzeiger initialisieren
A684   48         PHA         2 Bytes aus dem Y-Register
A685   98         TYA         und dem Akku wieder auf
A686   48         PHA         den Stapel schieben
A687   A9 00      LDA #$00    Wert laden und damit
A689   85 3E      STA $3E     CONT sperren
```

```
A68B  85 10     STA $10     und in FN-Flag speichern
A68D  60        RTS         Rücksprung

**************************  Programmzeiger auf
                            BASIC-Start

Einsprung von $A659, $E185

A68E  18        CLC         Carry löschen (Addition)
A68F  A5 2B     LDA $2B     Zeiger auf Programmstart (LOW)
A691  69 FF     ADC #$FF    minus 1 ergibt
A693  85 7A     STA $7A     neuen CHRGET-Zeiger (LOW)
A695  A5 2C     LDA $2C     Programmstart (HIGH)
A697  69 FF     ADC #$FF    minus 1 ergibt
A699  85 7B     STA $7B     CHRGET-Zeiger (HIGH)
A69B  60        RTS         Rücksprung

**************************  BASIC Befehl  LIST
A69C  90 06     BCC $A6A4   Ziffer ? (Zeilennummer)
A69E  F0 04     BEQ $A6A4   nur LIST ?
A6A0  C9 AB     CMP #$AB    Code für '-'?
A6A2  D0 E9     BNE $A68D   anderer Code, dann SYNTAX ERR
A6A4  20 6B A9  JSR $A96B   Zeilennummer holen
A6A7  20 13 A6  JSR $A613   Startadresse berechnen
A6AA  20 79 00  JSR $0079   CHRGOT letztes Zeichen holen
A6AD  F0 0C     BEQ $A6BB   keine Zeilennummer
A6AF  C9 AB     CMP #$AB    Code für '-'?
A6B1  D0 8E     BNE $A641   nein: SYNTAX ERROR
A6B3  20 73 00  JSR $0073   CHRGET nächstes Zeichen holen
A6B6  20 6B A9  JSR $A96B   Zeilennummer holen
A6B9  D0 86     BNE $A641   kein Trennzeichen: SYNTAX ERR
A6BB  68        PLA         2 Bytes von Stapel holen
A6BC  68        PLA         (Rücksprungadresse übergehen)
A6BD  A5 14     LDA $14     zweite Zeilennummer laden
A6BF  05 15     ORA $15     gleich null ?
A6C1  D0 06     BNE $A6C9   Nein: $A6C9
A6C3  A9 FF     LDA #$FF    Wert laden und
A6C5  85 14     STA $14     zweite Zeilennummer Maximal-
A6C7  85 15     STA $15     wert $FFFF (65535)
A6C9  A0 01     LDY #$01    Zeiger setzen
```

```
A6CB  84 0F     STY $0F      und Quote-Modus abschalten
A6CD  B1 5F     LDA ($5F),Y Linkadresse HIGH holen
A6CF  F0 43     BEQ $A714    Ja: dann fertig
A6D1  20 2C A8  JSR $A82C    prüft auf Stop-Taste
A6D4  20 D7 AA  JSR $AAD7    'CR' ausgeben, neue Zeile
A6D7  C8        INY          Zeiger erhöhen
A6D8  B1 5F     LDA ($5F),Y Zeilenadresse holen (LOW)
A6DA  AA        TAX          und in das X-Reg. schieben
A6DB  C8        INY          Zeiger erhöhen
A6DC  B1 5F     LDA ($5F),Y Zeilenadresse holen (HIGH)
A6DE  C5 15     CMP $15      mit Endnummer vergleichen
A6E0  D0 04     BNE $A6E6    Gleich? Nein: $A6E6
A6E2  E4 14     CPX $14      LOW-Nummer vergleichen
A6E4  F0 02     BEQ $A6E8    Gleich? Ja: $A6E8
A6E6  B0 2C     BCS $A714    Größer: dann fertig
A6E8  84 49     STY $49      Y-Reg. zwischenspeichern
A6EA  20 CD BD  JSR $BDCD    Zeilennnummer ausgeben
A6ED  A9 20     LDA #$20     ' ' Leerzeichen
A6EF  A4 49     LDY $49      Y-Reg. wiederholen
A6F1  29 7F     AND #$7F     Bit 7 löschen
A6F3  20 47 AB  JSR $AB47    Zeichen ausgeben
A6F6  C9 22     CMP #$22     '"' Hochkomma ?
A6F8  D0 06     BNE $A700    Nein: $A700
A6FA  A5 0F     LDA $0F      Hochkomma-Flag laden,
A6FC  49 FF     EOR #$FF     umdrehen (NOT)
A6FE  85 0F     STA $0F      und wieder abspeichern
A700  C8        INY          Zeilenende nach 255 Zeichen ?
A701  F0 11     BEQ $A714    Nein: dann aufhören
A703  B1 5F     LDA ($5F),Y Zeichen holen
A705  D0 10     BNE $A717    kein Zeilenende, dann listen
A707  A8        TAY          Akku als Zeiger nach Y
A708  B1 5F     LDA ($5F),Y Startadresse der nächsten
A70A  AA        TAX          Zeile holen (LOW) und nach X
A70B  C8        INY          Zeiger erhöhen
A70C  B1 5F     LDA ($5F),Y Adresse der Zeile (HIGH)
A70E  86 5F     STX $5F      als Zeiger merken
A710  85 60     STA $60      (speichern nach $5F/60) und
A712  D0 B5     BNE $A6C9    weitermachen
A714  4C 86 E3  JMP $E386    zum BASIC-Warmstart
```

```
**************************** BASIC-Code in Klartext
                              umwandlen
A717   6C 06 03   JMP ($0306) JMP $A71A

Einsprung von $A717

A71A   10 D7      BPL A6F3    kein Interpretercode:ausgeben
A71C   C9 FF      CMP #$FF    Code für Pi?
A71E   F0 D3      BEQ A6F3    Ja: so ausgeben
A720   24 0F      BIT  0F     Hochkommamodus ?
A722   30 CF      BMI A6F3    dann Zeichen so ausgeben
A724   38         SEC         Carry setzen (Subtraktion)
A725   E9 7F      SBC #$7F    Offset abziehen
A727   AA         TAX         Code nach X
A728   84 49      STY $49     Zeichenzeiger merken
A72A   A0 FF      LDY #$FF    Zeiger auf Befehlstabelle
A72C   CA         DEX         erstes Befehlswort?
A72D   F0 08      BEQ $A737   Ja: ausgeben
A72F   C8         INY         Zeiger erhöhen
A730   B9 9E A0   LDA $A09E,Y Offset für X-tes Befehlswort
A733   10 FA      BPL $A72F   alle Zeichen bis zum letzen
A735   30 F5      BMI $A72C   überlesen (Bit 7 gesetzt)
A737   C8         INY         Zeiger erhöhen
A738   B9 9E A0   LDA $A09E,Y Befehlswort aus Tabelle holen
A73B   30 B2      BMI $A6EF   letzter Buchstabe: fertig
A73D   20 47 AB   JSR $AB47   Zeichen ausgeben
A740   D0 F5      BNE $A737   nächsten Buchstaben ausgeben

**************************** BASIC-Befehl  FOR
A742   A9 80      LDA #$80    Wert laden und
A744   85 10      STA $10     Integer sperren
A746   20 A5 A9   JSR $A9A5   LET, setzt FOR-Variable
A749   20 8A A3   JSR $A38A   sucht offene FOR-NEXT-Schlei.
A74C   D0 05      BNE $A753   nicht gefunden: $A753
A74E   8A         TXA         X-Reg. nach Akku
A74F   69 0F      ADC #$0F    Stapelzeiger erhöhen
A751   AA         TAX         Akku zurück nach X-Reg. und
A752   9A         TXS         in den Stapelzeiger
A753   68         PLA         Rücksprungadresse vom Stapel
A754   68         PLA         holen (LOW und HIGH)
```

```
A755  A9 09     LDA #$09    Wert für Prüfung laden
A757  20 FB A3  JSR $A3FB   prüft auf Platz im Stapel
A75A  20 06 A9  JSR $A906   sucht nächstes BAS.-Statement
A75D  18        CLC         Carry löschen (Addition)
A75E  98        TYA         CHRGET-Zeiger und Offset
A75F  65 7A     ADC $7A     = Startadresse der Schleife
A761  48        PHA         auf Stapel speichern
A762  A5 7B     LDA $7B     HIGH-Byte holen und
A764  69 00     ADC #$00    Übertrag addieren und
A766  48        PHA         auf den Stapel legen
A767  A5 3A     LDA $3A     Aktuelle
A769  48        PHA         Zeilennummer laden und auf
A76A  A5 39     LDA $39     den Stapel schieben
A76C  48        PHA         (LOW und HIGH-Byte)
A76D  A9 A4     LDA #$A4      'TO' - Code
A76F  20 FF AE  JSR $AEFF   prüft auf Code
A772  20 8D AD  JSR $AD8D   prüft ob numerische Variable
A775  20 8A AD  JSR $AD8A   numerischer Ausdruck nach FAC
A778  A5 66     LDA $66     Vorzeichenbyte von FAC holen
A77A  09 7F     ORA #$7F    Bit 0 bis 6 setzen
A77C  25 62     AND $62     mit $62 angleichen
A77E  85 62     STA $62     und abspeichern
A780  A9 8B     LDA #$8B    Rücksprungadresse laden
A782  A0 A7     LDY #$A7    (LOW und HIGH)
A784  85 22     STA $22     und zwischenspeichern
A786  84 23     STY $23     (LOW und HIGH)
A788  4C 43 AE  JMP $AE43   Schleifenendwert auf Stapel
A78B  A9 BC     LDA #$BC    Zeiger auf Konstante 1 setzen
A78D  A0 B9     LDY #$B9    (Ersatzwert für STEP)
A78F  20 A2 BB  JSR $BBA2   als Default-STEP-Wert in FAC
A792  20 79 00  JSR $0079   CHRGOT: letztes Zeichen holen
A795  C9 A9     CMP #$A9    'STEP' - Code?
A797  D0 06     BNE $A79F   kein STEP-Wert: $A79F
A799  20 73 00  JSR $0073   CHRGET nächstes Zeichen holen
A79C  20 8A AD  JSR $AD8A   numerischer Ausdruck nach FAC
A79F  20 2B BC  JSR $BC2B   holt Vorzeichenbyte
A7A2  20 38 AE  JSR $AE38   Vorz. und STEP-Wert auf Stack
A7A5  A5 4A     LDA $4A     Zeiger auf Variablenwert
A7A7  48        PHA         (LOW) auf den Stapel
A7A8  A5 49     LDA $49     Zeiger (HIGH)
```

```
A7AA   48         PHA          auf den Stapel
A7AB   A9 81      LDA #$81     und FOR-Code
A7AD   48         PHA          auf den Stapel legen

**************************** Interpreterschleife

Einsprung von $A7EA, $A89D, $AD75

A7AE   20 2C A8   JSR $A82C    prüft auf Stop-Taste
A7B1   A5 7A      LDA $7A      CHRGET Zeiger (LOW und HIGH)
A7B3   A4 7B      LDY $7B      laden
A7B5   C0 02      CPY #$02     Direkt-Modus?
A7B7   EA         NOP          No OPeration
A7B8   F0 04      BEQ $A7BE    ja: $A7BE
A7BA   85 3D      STA $3D      als Zeiger für CONT
A7BC   84 3E      STY $3E      merken
A7BE   A0 00      LDY #$00     Zeiger setzen
A7C0   B1 7A      LDA ($7A),Y laufendes Zeichen holen
A7C2   D0 43      BNE $A807    nicht Zeilenende?
A7C4   A0 02      LDY #$02     Zeiger neu setzen
A7C6   B1 7A      LDA ($7A),Y Programmende?
A7C8   18         CLC          Flag für END setzen
A7C9   D0 03      BNE $A7CE    Kein Programmende: $A7CE
A7CB   4C 4B A8   JMP $A84B    ja: dann END ausführen
A7CE   C8         INY          Zeiger erhöhen
A7CF   B1 7A      LDA ($7A),Y laufende Zeilennummer
A7D1   85 39      STA $39      (LOW) nach $39
A7D3   C8         INY          Zeiger auf nächstes Byte
A7D4   B1 7A      LDA ($7A),Y laufende Zeilennummer
A7D6   85 3A      STA $3A      (HIGH) nach $3A
A7D8   98         TYA          Zeiger nach Akku
A7D9   65 7A      ADC $7A      Programmzeiger auf
A7DB   85 7A      STA $7A      Programmzeile setzen
A7DD   90 02      BCC $A7E1    C=0: Erhöhung umgehen
A7DF   E6 7B      INC $7B      Programmzeiger (HIGH) erhöhen

Einsprung von $A499

A7E1   6C 08 03   JMP ($0308) JMP $A7E4 Statement ausführen
```

```
Einsprung von $A7E1

A7E4   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
A7E7   20 ED A7   JSR $A7ED   Statement ausführen
A7EA   4C AE A7   JMP $A7AE   zurück zur Interpreterschlei.

**************************** BASIC-Statement ausführen

Einsprung von $A7E7, $A948

A7ED   F0 3C      BEQ $A82B   Zeilenende, dann fertig

Einsprung von $A95C

A7EF   E9 80      SBC #$80    Token?
A7F1   90 11      BCC $A804   nein: dann zum LET-Befehl
A7F3   C9 23      CMP #$23    NEW?
A7F5   B0 17      BCS $A80E   Funktions-Token oder GO TO
A7F7   0A         ASL         BASIC-Befehl, Code mal 2
A7F8   A8         TAY         als Zeiger ins Y-Reg.
A7F9   B9 0D A0   LDA $A00D,Y Befehlsadresse (LOW und
A7FC   48         PHA         HIGH) aus Tabelle
A7FD   B9 0C A0   LDA $A00C,Y holen und als
A800   48         PHA         Rücksprungadresse auf Stapel
A801   4C 73 00   JMP $0073   Zeichen und Befehl ausführen
A804   4C A5 A9   JMP $A9A5   zum LET-Befehl

****************************
A807   C9 3A      CMP #$3A    ':' ist es Doppelpunkt?
A809   F0 D6      BEQ $A7E1   ja: $A7E1
A80B   4C 08 AF   JMP $AF08   sonst 'SYNTAX ERROR'

**************************** prüft auf 'GO' 'TO' Code
A80E   C9 4B      CMP #$4B    'GO'  (minus $80)
A810   D0 F9      BNE $A80B   nein: 'SYNTAX ERROR'
A812   20 73 00   JSR $0073   nächstes Zeichen holen
A815   A9 A4      LDA #$A4    'TO'
A817   20 FF AE   JSR $AEFF   prüft auf Code
A81A   4C A0 A8   JMP $A8A0   zum GOTO-Befehl
```

```
****************************  BASIC-Befehl  RESTORE

Einsprung von $A677

A81D   38        SEC         Carry setzen (Subtraktion)
A81E   A5 2B     LDA $2B     Programmstartzeiger (LOW)
A820   E9 01     SBC #$01    laden und davon 1 abziehen
A822   A4 2C     LDY $2C     und HIGH-Byte holen
A824   B0 01     BCS $A827
A826   88        DEY         LOW-Byte -1

Einsprung von $ACE7

A827   85 41     STA $41     als DATA-Zeiger
A829   84 42     STY $42     abspeichern
A82B   60        RTS         Rücksprung

****************************  prüft auf Stop-Taste

Einsprung von $A6D1, $A7AE

A82C   20 E1 FF  JSR $FFE1   Stop-Taste abfragen

****************************  BASIC-Befehl  STOP
A82F   B0 01     BCS $A832   C=1: Flag für STOP

****************************  BASIC-Befehl  END
A831   18        CLC         C=0  Flag für END
A832   D0 3C     BNE $A870   RUN/STOP nicht gedrückt: RTS
A834   A5 7A     LDA $7A     Programmzeiger laden
A836   A4 7B     LDY $7B     (LOW und HIGH-Byte)
A838   A6 3A     LDX $3A     Direkt-Modus?
A83A   E8        INX         (Zeilennummer -1)
A83B   F0 0C     BEQ $A849   ja: $A849
A83D   85 3D     STA $3D     als Zeiger für CONT setzen
A83F   84 3E     STY $3E     (LOW und HIGH)
A841   A5 39     LDA $39     Nummer der laufenden Zeile
A843   A4 3A     LDY $3A     holen (LOW und HIGH)
A845   85 3B     STA $3B     und als Zeilennummer für
A847   84 3C     STY $3C     CONT  merken
```

```
A849  68        PLA         Rücksprungadresse
A84A  68        PLA         vom Stapel entfernen

Einsprung von $A7CB

A84B  A9 81     LDA #$81    Zeiger auf Startadresse
A84D  A0 A3     LDY #$A3    BREAK setzen
A84F  90 03     BCC $A854   END Flag?
A851  4C 69 A4  JMP $A469   nein: 'BREAK IN XXX' ausgeben
A854  4C 86 E3  JMP $E386   zum BASIC-Warmstart

**************************  BASIC-Befehl  CONT
A857  D0 17     BNE $A870   Kein Trennzeichen: SYNTAX ERR
A859  A2 1A     LDX #$1A    Fehlernr. für 'CAN'T CONTINUE
A85B  A4 3E     LDY $3E     CONT gesperrt?
A85D  D0 03     BNE $A862   nein: $A862
A85F  4C 37 A4  JMP $A437   Fehlermeldung ausgeben
A862  A5 3D     LDA $3D     CONT-Zeiger (LOW) laden
A864  85 7A     STA $7A     und CONT-Zeiger als Programm-
A866  84 7B     STY $7B     zeiger abspeichern
A868  A5 3B     LDA $3B     und
A86A  A4 3C     LDY $3C     Zeilennummer wieder
A86C  85 39     STA $39     setzen
A86E  84 3A     STY $3A     (LOW- und HIGH-Byte)
A870  60        RTS         Rücksprung

**************************  BASIC-Befehl  RUN
A871  08        PHP         Statusregister retten
A872  A9 00     LDA #$00    Wert laden und
A874  20 90 FF  JSR $FF90   Flag für Programmodus setzen
A877  28        PLP         Statusregister zurückholen
A878  D0 03     BNE $A87D   weitere Zeichen (Zeilennr.)?
A87A  4C 59 A6  JMP $A659   Programmzeiger setzen, CLR

A87D  20 60 A6  JSR $A660   CLR-Befehl
A880  4C 97 A8  JMP $A897   GOTO-Befehl

**************************  BASIC-Befehl  GOSUB
A883  A9 03     LDA #$03    Wert für Prüfung
A885  20 FB A3  JSR $A3FB   prüft auf Platz im Stapel
```

```
A888  A5 7B     LDA $7B     Programmzeiger (LOW-
A88A  48        PHA         und HIGH-Byte) laden
A88B  A5 7A     LDA $7A     und auf den
A88D  48        PHA         Stapel retten
A88E  A5 3A     LDA $3A     Zeilennummer laden (HIGH)
A890  48        PHA         und auf den Stapel legen
A891  A5 39     LDA $39     Zeilennummer LOW laden
A893  48        PHA         und auf den Stapel legen
A894  A9 8D     LDA #$8D    'GOSUB'-Code laden
A896  48        PHA         und auf den Stapel legen

Einsprung von $A880

A897  20 79 00  JSR $0079   CHRGOT: letztes Zeichen holen
A89A  20 A0 A8  JSR $A8A0   GOTO-Befehl
A89D  4C AE A7  JMP $A7AE   zur Interpreterschleife

**************************** BASIC-Befehl  GOTO

Einsprung von $A81A, $A89A, $A945

A8A0  20 6B A9  JSR $A96B   Zeilennummer nach $14/$15
A8A3  20 09 A9  JSR $A909   nächsten Zeilenanfang suchen
A8A6  38        SEC         Carry löschen (Subtraktion)
A8A7  A5 39     LDA $39     aktuelle Zeilennummer (LOW)
A8A9  E5 14     SBC $14     kleiner als laufende Zeile?
A8AB  A5 3A     LDA $3A     aktuelle Zeilennummer (HIGH)
A8AD  E5 15     SBC $15     kleiner als laufende Zeile?
A8AF  B0 0B     BCS $A8BC   nein: $A8BC
A8B1  98        TYA         Differenz in Akku
A8B2  38        SEC         Carry setzen (Addition)
A8B3  65 7A     ADC $7A     Programmzeiger addieren
A8B5  A6 78     LDX $7B     sucht ab laufender Zeile
A8B7  90 07     BCC $A8C0   unbedingter
A8B9  E8        INX         Sprung
A8BA  B0 04     BCS $A8C0   zu $A8C0
A8BC  A5 2B     LDA $2B     sucht ab Programmstart
A8BE  A6 2C     LDX $2C
A8C0  20 17 A6  JSR $A617   sucht Programmzeile
A8C3  90 1E     BCC $A8E3   nicht gefunden: 'undef'd st.'
```

```
A8C5   A5 5F      LDA $5F     von der Startadresse (Zeile)
A8C7   E9 01      SBC #$01    eins subtrahieren und als
A8C9   85 7A      STA $7A     Programmzeiger (LOW)
A8CB   A5 60      LDA $60     HIGH-Byte der Zeile laden
A8CD   E9 00      SBC #$00    Übertrag berücksichtigen
A8CF   85 7B      STA $7B     und als Programmzeiger
A8D1   60         RTS         Rücksprung

****************************  BASIC-Befehl  RETURN
A8D2   D0 FD      BNE $A8D1   Kein Trennzeichen: SYNTAX ERR
A8D4   A9 FF      LDA #$FF    Wert laden und
A8D6   85 4A      STA $4A     FOR-NEXT-ZEIGER neu setzen
A8D8   20 8A A3   JSR $A38A   GOSUB-Datensatz suchen
A8DB   9A         TXS
A8DC   C9 8D      CMP #$8D    'GOSUB'-Code?
A8DE   F0 0B      BEQ $A8EB   ja: $A8E8
A8E0   A2 0C      LDX #$0C    Nr für 'return without gosub'
A8E2   2C         .BYTE $2C
A8E3   A2 11      LDX #$02    Nr für 'undef'd statement'
A8E5   4C 37 A4   JMP $A437   Fehlermeldung ausgeben
A8E8   4C 08 AF   JMP $AF08   'syntax error' ausgeben

A8EB   68         PLA         GOSUB-Code vom Stapel holen
A8EC   68         PLA         Zeilennummer (LOW) wieder-
A8ED   85 39      STA $39     holen und abspeichern
A8EF   68         PLA         Zeilennummer (HIGH) holen
A8F0   85 3A      STA $3A     und abspeichern
A8F2   68         PLA         Programmzeiger (LOW) wieder-
A8F3   85 7A      STA $7A     holen und abspeichern
A8F5   68         PLA         Programmzeiger (HIGH) holen
A8F6   85 7B      STA $7B     abspeichern

****************************  BASIC-Befehl  DATA

Einsprung von $ABE7, $B3DB

A8F8   20 06 A9   JSR $A906   nächstes Statement suchen
```

```
Einsprung von $ABF6, $ACD1

A8FB   98        TYA         Offset
A8FC   18        CLC         Carry löschen (Addition)
A8FD   65 7A     ADC $7A     Programmzeiger addieren
A8FF   85 7A     STA $7A     und wieder abspeichern
A901   90 02     BCC $A905   Verminderung übergehen
A903   E6 7B     INC $7B     Programmzeiger vermindern
A905   60        RTS         Rücksprung

**************************** Offset des nächsten
                             Trennzeichens finden

Einsprung von $A75A, $A8F8, $ABF3, $ACB8

A906   A2 3A     LDX #$3A    ':' Doppelpunkt
A908   2C        .BYTE $2C

Einsprung von $A8A3, $A93B

A909   A2 00     LDX #$00    $0  Zeilenende
A90B   86 07     STX $07     als Suchzeichen
A90D   A0 00     LDY #$00    Zähler
A90F   84 08     STY $08     initialisieren
A911   A5 08     LDA $08     Speicherzelle $7
A913   A6 07     LDX $07     gesuchtes Zeichen
A915   85 07     STA $07     mit $8
A917   86 08     STX $08     vertauschen
A919   B1 7A     LDA ($7A),Y Zeichen holen
A91B   F0 E8     BEQ $A905   Zeilenende, dann fertig
A91D   C5 08     CMP $08     = Suchzeichen?
A91F   F0 E4     BEQ $A905   ja: $A905
A921   C8        INY         Zeiger erhöhen
A922   C9 22     CMP #$22    '"' Hochkomma?
A924   D0 F3     BNE $A919   nein: $A919
A926   F0 E9     BEQ $A911   sonst $7 und $8 vertauschen

**************************** BASIC-Befehl  IF
A928   20 9E AD  JSR $AD9E   FRMEVL  Ausdruck berechnen
A92B   20 79 00  JSR $0079   CHRGOT  letztes Zeichen
```

```
A92E   C9 89      CMP #$89    'GOTO'-Code?
A930   F0 05      BEQ $A937   ja: $A937
A932   A9 A7      LDA #$A7    'THEN'-Code
A934   20 FF AE   JSR $AEFF   prüft auf Code
A937   A5 61      LDA $61     Ergebnis des IF-Ausdrucks
A939   D0 05      BNE $A940   Ausdruck wahr?

****************************  BASIC-Befehl  REM
A93B   20 09 A9   JSR $A909   nein, Zeilenanfang suchen
A93E   F0 BB      BEQ $A8FB   Programmz. auf nächste Zeile

A940   20 79 00   JSR $0079   CHRGOT: letztes Zeichen holen
A943   B0 03      BCS $A948   keine Ziffer?
A945   4C A0 A8   JMP $A8A0   zum GOTO-Befehl
A948   4C ED A7   JMP $A7ED   Befehl dekodieren, ausführen

****************************  BASIC-Befehl  ON
A94B   20 9E B7   JSR $B79E   Byte-Wert (0 bis 255) holen
A94E   48         PHA         Code merken
A94F   C9 8D      CMP #$8D    'GOSUB'-Code?
A951   F0 04      BEQ $A957   ja: $A957
A953   C9 89      CMP #$89    'GOTO'-Code?
A955   D0 91      BNE $A8E8   nein: dann 'SYNTAX ERROR'
A957   C6 65      DEC $65     Zähler vermindern
A959   D0 04      BNE $A95F   noch nicht null?
A95B   68         PLA         ja: Code zurückholen
A95C   4C EF A7   JMP $A7EF   und Befehl ausführen

A95F   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
A962   20 6B A9   JSR $A96B   Zeilennummer holen
A965   C9 2C      CMP #$2C    ',' Komma?
A967   F0 EE      BEQ $A957   ja: dann weiter
A969   68         PLA         kein Sprung: Code zurückholen
A96A   60         RTS         Rücksprung
```

```
**************************** Zeilennummer nach $14/$15

Einsprung von $A49C, $A6A4, $A6B6, A8A0, $A962

A96B   A2 00      LDX #$00    Wert laden und
A96D   86 14      STX $14     Vorsetzen
A96F   86 15      STX $15     (für Zeilennummer gleich 0)

Einsprung von $A9A2

A971   B0 F7      BCS $A96A   keine Ziffer, dann fertig
A973   E9 2F      SBC #$2F    '0'-1 abziehen, gibt Hexwert
A975   85 07      STA $07     merken
A977   A5 15      LDA $15     HIGH-Byte holen
A979   85 22      STA $22     zwischenspeichern
A97B   C9 19      CMP #$19    Zahl bereits größer 6400?
A97D   B0 D4      BCS $A953   dann 'SYNTAX ERROR'
A97F   A5 14      LDA $14     Zahl * 10 (= *2*2+Zahl*2)
A981   0A         ASL         Wert und Zwischenwert je
A982   26 22      ROL $22     2 mal um 1 Bit nach
A984   0A         ASL         links rollen
A985   26 22      ROL $22     (entspricht 2 * 2)
A987   65 14      ADC $14     plus ursprünglicher Wert
A989   85 14      STA $14     und abspeichern
A98B   A5 22      LDA $22     Zwischenwert zu
A98D   65 15      ADC $15     zweitem Wert addieren
A98F   85 15      STA $15     und wieder abspeichern
A991   06 14      ASL $14     Speicherzelle $14 und
A993   26 15      ROL $15     $15 verdoppeln
A995   A5 14      LDA $14     Wert wieder laden
A997   65 07      ADC $07     und Einerziffer addieren
A999   85 14      STA $14     wieder speichern
A99B   90 02      BCC $A99F   Carry gesetzt? (Übertrag)
A99D   E6 15      INC $15     Übertrag addieren
A99F   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
A9A2   4C 71 A9   JMP $A971   und weiter machen
```

```
**************************** BASIC-Befehl  LET

Einsprung von $A746, $A804

A9A5   20 8B B0   JSR $B08B   sucht Variable hinter LET
A9A8   85 49      STA $49     und Variablenadresse
A9AA   84 4A      STY $4A     merken (LOW- und HIGH-Byte)
A9AC   A9 B2      LDA #$B2    '=' - Code
A9AE   20 FF AE   JSR $AEFF   prüft auf Code
A9B1   A5 0E      LDA $0E     Integer-Flag
A9B3   48         PHA         auf Stapel retten
A9B4   A5 0D      LDA $0D     und Typ-Flag
A9B6   48         PHA         (String/numerisch) retten
A9B7   20 9E AD   JSR $AD9E   FRMEVL: Ausdruck holen
A9BA   68         PLA         Typ-Flag wiederholen
A9BB   2A         ROL  A      und Bit 7 ins Carry schieben
A9BC   20 90 AD   JSR $AD90   auf richtigen Typ prüfen
A9BF   D0 18      BNE $A9D9   String? ja: $A9D9
A9C1   68         PLA         Integer-Flag zurückholen

Einsprung von $AC8E

A9C2   10 12      BPL $A9D6   INTEGER? ja: $A9D6

**************************** Wertzuweisung INTEGER
A9C4   20 1B BC   JSR $BC1B   FAC runden
A9C7   20 BF B1   JSR $B1BF   und nach INTEGER wandlen
A9CA   A0 00      LDY #$00    Zeiger setzen
A9CC   A5 64      LDA $64     HIGH-Byte holen und
A9CE   91 49      STA ($49),Y Wert in Variable bringen
A9D0   C8         INY         Zeiger erhöhen
A9D1   A5 65      LDA $65     LOW-Byte holen und
A9D3   91 49      STA ($49),Y Wert in Variable bringen
A9D5   60         RTS         Rücksprung

**************************** Wertzuweisung REAL
A9D6   4C D0 BB   JMP $BBD0   FAC nach Variable bringen
```

```
**************************  Wertzuweisung String
A9D9   68         PLA       Akku vom Stapel holen

Einsprung von $AC83

A9DA   A4 4A      LDY $4A   Variablenadresse (HIGH) holen
A9DC   C0 BF      CPY #$BF  ist Variable TI$?
A9DE   D0 4C      BNE $AA2C nein: $AA2C
A9E0   20 A6 B6   JSR $B6A6 FRESTR
A9E3   C9 06      CMP #$06  Stringlänge gleich 6
A9E5   D0 3D      BNE $AA24 nein: 'illegal quantity'
A9E7   A0 00      LDY #$00  Wert holen
A9E9   84 61      STY $61   und damit FAC
A9EB   84 66      STY $66   initialisieren
A9ED   84 71      STY $71   (Akku, Vorzeichen und Zeiger)
A9EF   20 1D AA   JSR $AA1D prüft nächstes Z. auf Ziffer
A9F2   20 E2 BA   JSR $BAE2 FAC = FAC * 10
A9F5   E6 71      INC $71   Stellenzähler erhöhen
A9F7   A4 71      LDY $71   und ins Y-Reg. bringen
A9F9   20 1D AA   JSR $AA1D prüft nächstes Z. auf Ziffer
A9FC   20 0C BC   JSR $BC0C FAC nach ARG kopieren
A9FF   AA         TAX       FAC gleich 0?
AA00   F0 05      BEQ $AA07 ja: $AA07
AA02   E8         INX       Exponent von FAC erhöhen
AA03   8A         TXA       (FAC *2) und in den Akku
AA04   20 ED BA   JSR $BAED FAC = FAC + ARG
AA07   A4 71      LDY $71   Stellenzähler
AA09   C8         INY       erhöhen
AA0A   C0 06      CPY #$06  schon 6 Stellen?
AA0C   D0 DF      BNE $A9ED nein: nächstes Zeichen
AA0E   20 E2 BA   JSR $BAE2 FAC = FAC * 10
AA11   20 9B BC   JSR $BC9B FAC rechtsbündig machen
AA14   A6 64      LDX $64   Werte für
AA16   A4 63      LDY $63   eingegebene Uhrzeit
AA18   A5 65      LDA $65   holen und
AA1A   4C DB FF   JMP $FFDB Time setzen
```

```
**************************** Zeichen auf Ziffer prüfen

Einsprung von $A9EF, $A9F9

AA1D   B1 22      LDA ($22),Y Zeichen holen (aus String)
AA1F   20 80 00   JSR $0080   auf Ziffer prüfen
AA22   90 03      BCC $AA27   Ziffer: $AA27
AA24   4C 48 B2   JMP $B248   sonst: 'illegal quantity'
AA27   E9 2F      SBC #$2F    von ASCII nach HEX umwandeln
AA29   4C 7E BD   JMP $BD7E   in FAC und ARG übertragen

**************************** Wertzuweisung an normalen
                              String
AA2C   A0 02      LDY #$02    Zeiger setzen
AA2E   B1 64      LDA ($64),Y Stringadresse HIGH mit
AA30   C5 34      CMP $34     Stringanfangsadr. vergleichen
AA32   90 17      BCC $AA4B   kleiner: String im Programm
AA34   D0 07      BNE $AA3D   größer: $AA3D
AA36   88         DEY         Zeiger vermindern
AA37   B1 64      LDA ($64),Y Stringadresse (LOW) holen
AA39   C5 33      CMP $33     und vergleichen
AA3B   90 0E      BCC $AA4B   kleiner: String im Programm
AA3D   A4 65      LDY $65     Zeiger auf Stringdescriptor
AA3F   C4 2E      CPY $2E     mit Variablenstart vergl.
AA41   90 08      BCC $AA4B   kleiner: $AA4B
AA43   D0 0D      BNE $AA52   größer: $AA52
AA45   A5 64      LDA $64     Stringdiscriptorzeiger (LOW)
AA47   C5 2D      CMP $2D     mit Variablenstart vergl.
AA49   B0 07      BCS $AA52   größer: $AA52
AA4B   A5 64      LDA $64     Zeiger in Akku und Y-Reg.
AA4D   A4 65      LDY $65     auf Stringdescriptor setzen
AA4F   4C 68 AA   JMP $AA68   bis $AA68 überspringen
AA52   A0 00      LDY #$00    Zeiger setzen
AA54   B1 64      LDA ($64),Y Länge des Strings holen
AA56   20 75 B4   JSR $B475   prüft Platz, setzt Stringz.
AA59   A5 50      LDA $50     Zeiger auf Stringdescriptor
AA5B   A4 51      LDY $51     holen (LOW- und HIGH-Byte)
AA5D   85 6F      STA $6F     und
AA5F   84 70      STY $70     speichern
AA61   20 7A B6   JSR $B67A   String in Bereich übertragen
```

```
AA64   A9 61      LDA #$61    Werte laden
AA66   A0 00      LDY #$00    und damit

Einsprung von $AAeF

AA68   85 50      STA $50     Stringdiscriptor
AA6A   84 51      STY $51     neu setzen
AA6C   20 DB B6   JSR $B6DB   Descriptor löschen
AA6F   A0 00      LDY #$00    Zeiger setzen
AA71   B1 50      LDA ($50),Y Länge des Descriptors holen
AA73   91 49      STA ($49),Y und abspeichern
AA75   C8         INY         Zeiger erhöhen
AA76   B1 50      LDA ($50),Y Adresse (LOW) holen
AA78   91 49      STA ($49),Y und speichern
AA7A   C8         INY         Zeiger erhöhen
AA7B   B1 50      LDA ($50),Y und Adresse (HIGH)
AA7D   91 49      STA ($49),Y in Variable bringen
AA7F   60         RTS         Rücksprung

**************************** BASIC-Befehl  PRINT#
AA80   20 86 AA   JSR $AA86   CMD-Befehl
AA83   4C B5 AB   JMP $ABB5   und CLRCH

**************************** BASIC-Befehl  CMD

Einsprung von $AA80

AA86   20 9E B7   JSR $B79E   holt Byte-Ausdruck
AA89   F0 05      BEQ $AA90   Trennzeichen: $AA90
AA8B   A9 2C      LDA #$2C    ',', Wert laden
AA8D   20 FF AE   JSR $AEFF   prüft auf Komma
AA90   08         PHP         Statusregister merken
AA91   86 13      STX $13     Nr. des Ausgabegeräts merken
AA93   20 18 E1   JSR $E118   CKOUT, Ausgabegerät setzen
AA96   28         PLP         Statusregister wiederholen
AA97   4C A0 AA   JMP $AAA0   zum PRINT-Befehl
AA9A   20 21 AB   JSR $AB21   String drucken
AA9D   20 79 00   JSR $0079   CHRGOT  letztes Zeichen
```

```
**************************** BASIC-Befehl  PRINT

Einsprung von $AA97

AAA0   F0 35      BEQ $AAD7   Trennzeichen: $AAD7

Einsprung von $AB16

AAA2   F0 43      BEQ $AAE7   Trennz. (TAB, SPC): RTS
AAA4   C9 A3      CMP #$A3    'TAB('-Code?
AAA6   F0 50      BEQ $AAF8   ja: $AAF8
AAA8   C9 A6      CMP #$A6    'SPC('-Code?
AAAA   18         CLC         Flag für SPC setzen
AAAB   F0 4B      BEQ $AAF8   SPC-Code: $AAF8
AAAD   C9 2C      CMP #$2C    ','-Code? (Komma)
AAAF   F0 37      BEQ $AAE8   ja: $AAE8
AAB1   C9 3B      CMP #$3B    ';'-Code? (Semikolon)
AAB3   F0 5E      BEQ $AB13   ja: nächstes Zeichen, weiter
AAB5   20 9E AD   JSR $AD9E   FRMEVL: Term holen
AAB8   24 0D      BIT $0D     Typflag
AABA   30 DE      BMI $AA9A   String?
AABC   20 DD BD   JSR $BDDD   FAC in ASCII-String wandeln
AABF   20 87 B4   JSR $B487   Stringparameter holen
AAC2   20 21 AB   JSR $AB21   String drucken
AAC5   20 3B AB   JSR $AB3B   Cursor right bzw. Leerzeichen
AAC8   D0 D3      BNE $AA9D   weiter machen

Einsprung von $A576

AACA   A9 00      LDA #$00    Eingabepuffer
AACC   9D 00 02   STA $0200,X mit $0 abschließen
AACF   A2 FF      LDX #$FF    Zeiger auf
AAD1   A0 01      LDY #$01    Eingabepuffer ab $0200 setzen
AAD3   A5 13      LDA $13     Nummer des Ausgabegeräts
AAD5   D0 10      BNE $AAE7   Tastatur? nein: RTS

Einsprung von $A44E, $A6D4

AAD7   A9 0D      LDA #$0D    'CR' carriage return
AAD9   20 47 AB   JSR $AB47   ausgeben
```

```
AADC   24 13      BIT $13     logische Filenummer
AADE   10 05      BPL $AAE5   kleiner 128?
AAE0   A9 0A      LDA #$0A    'LF' line feed
AAE2   20 47 AB   JSR $AB47   ausgeben

Einsprung von $AB35

AAE5   49 FF      EOR #$FF    NOT
AAE7   60         RTS         Rücksprung

AAE8   38         SEC         Zehner-Tabulator mit Komma
AAE9   20 F0 FF   JSR $FFF0   Cursorposition holen
AAEC   98         TYA         Spalte ins Y-Reg.
AAED   38         SEC         Carry setzen (Subtr.)
AAEE   E9 0A      SBC #$0A    10 abziehen
AAF0   B0 FC      BCS $AAEE   nicht negativ?
AAF2   49 FF      EOR #$FF    invertieren
AAF4   69 01      ADC #$01    +1 (Zweierkomplement)
AAF6   D0 16      BNE $AB0E   unbedingter Sprung

**************************** TAB(  (C=1)  und SPC( (C=0)
AAF8   08         PHP         Flags merken
AAF9   38         SEC         Carry setzen
AAFA   20 F0 FF   JSR $FFF0   Cursorposition holen
AAFD   84 09      STY $09     und Spalte merken
AAFF   20 9B B7   JSR $B79B   Byte-Wert holen
AB02   C9 29      CMP #$29    ')' Klammer zu?
AB04   D0 59      BNE $AB5F   nein: 'SYNTAX ERROR'
AB06   28         PLP         Flags wiederherstellen
AB07   90 06      BCC $AB0F   zu SPC(
AB09   8A         TXA         TAB-Wert in Akku
AB0A   E5 09      SBC $09     mit Cursorspalte vergleichen
AB0C   90 05      BCC $AB13   kleiner Cursor-Position: RTS
AB0E   AA         TAX         Schritte bis zum Tabulator
AB0F   E8         INX         aus Zähler initialisieren
AB10   CA         DEX         um 1 vermindern
AB11   D0 06      BNE $AB19   =0? nein: Cursor right
AB13   20 73 00   JSR $0073   nächstes Zeichen holen
AB16   4C A2 AA   JMP $AAA2   und weitermachen
```

```
AB19   20 3B AB   JSR $AB3B   Cursor right bzw. Leerzeichen
AB1C   D0 F2      BNE $AB10   zum Schleifenanfang

**************************** String ausgeben

Einsprung von $A469, $A478, $AB6F, $ACF8, $BDDA, $E191
$E42D, $E441, $FB8B

AB1E   20 87 B4   JSR $B487   Stringparameter holen

Einsprung von $AA9A, AAC2, $ABCB

AB21   20 A6 B6   JSR $B6A6   FRESTR
AB24   AA         TAX         Stringlänge
AB25   A0 00      LDY #$00    Zeiger für Stringausgabe
AB27   E8         INX         erhöhen

Einsprung von $AB38

AB28   CA         DEX         vermindern
AB29   F0 BC      BEQ $AAE7   String zu Ende?
AB2B   B1 22      LDA ($22),Y Zeichen des Strings
AB2D   20 47 AB   JSR $AB47   ausgeben
AB30   C8         INY         Zeiger erhöhen
AB31   C9 0D      CMP #$0D    'CR' carriage return?
AB33   D0 F3      BNE $AB28   nein: weiter
AB35   20 E5 AA   JSR $AAE5   Fehler ! Test auf LF-Ausgabe
AB38   4C 28 AB   JMP $AB28   und weitermachen
**************************** Ausgabe eines Leerzeichens
                              bzw. Cursor right

Einsprung von $AAC5, $AB19, $AC00

AB3B   A5 13      LDA $13     Ausgabe in File?
AB3D   F0 03      BEQ $AB42   Bildschirm: dann Cursor right
AB3F   A9 20      LDA #$20    ' ' Leerzeichencode laden
AB41   2C         .BYTE $2C
AB42   A9 1D      LDA #$1D    Cursor right Code laden
AB44   2C         .BYTE $2C
```

```
Einsprung von $A451, $ABFD, $AC47

AB45   A9 3F      LDA #$3F    '?' Fragezeichencode laden

Einsprung von $A45B, $A6F3, $A73D, $AAD9, $AAE2, $AB2D

AB47   20 0C E1   JSR $E10C   Code ausgeben
AB4A   29 FF      AND #$FF    Flags setzen
AB4C   60         RTS         Rücksprung

****************************  Fehlerbehandlung bei Eingabe

Einsprung von $AC9A

AB4D   A5 11      LDA $11     Flag für INPUT / GET / READ
AB4F   F0 11      BEQ $AB62   INPUT: $AB62
AB51   30 04      BMI $AB57   READ: $AB57
AB53   A0 FF      LDY #$FF    GET:
AB55   D0 04      BNE $AB5B   unbedingter Sprung

****************************  Fehler bei READ
AB57   A5 3F      LDA $3F     DATA-Zeilennummer
AB59   A4 40      LDY $40     holen (LOW- und HIGH-Byte)

****************************  Fehler bei GET
AB5B   85 39      STA $39     gleiche Zeilennummer
AB5D   84 3A      STY $3A     des Fehlers
AB5F   4C 08 AF   JMP $AF08   'SYNTAX ERROR'

****************************  Fehler bei INPUT
AB62   A5 13      LDA $13     Nummer des Eingabegeräts
AB64   F0 05      BEQ $AB6B   Tastatur: 'REDO FROM START'
AB66   A2 18      LDX #$18    Nummer für 'FILE DATA'
AB68   4C 37 A4   JMP $A437   Fehlermeldung ausgeben
AB6B   A9 0C      LDA #$0C    Zeiger in Akku und Y-Reg.
AB6D   A0 AD      LDY #$AD    auf '?REDO FROM START'
AB6F   20 1E AB   JSR $AB1E   String ausgeben
AB72   A5 3D      LDA $3D     Werte holen und
AB74   A4 3E      LDY $3E     Programmzeiger
AB76   85 7A      STA $7A     zurücksetzen
```

```
AB78  84 7B     STY $7B     auf INPUT-Befehl
AB7A  60        RTS         Rücksprung

****************************  BASIC-Befehl  GET
AB7B  20 A6 B3  JSR $B3A6   Testet auf Direkt-Modus
AB7E  C9 23     CMP #$23    folgt '#'?
AB80  D0 10     BNE $AB92   nein: $AB92
AB82  20 73 00  JSR $0073   CHRGET nächstes Zeichen holen
AB85  20 9E B7  JSR $B79E   Byte-Wert holen
AB88  A9 2C     LDA #$2C    ',' Komma
AB8A  20 FF AE  JSR $AEFF   prüft auf Code
AB8D  86 13     STX $13     Filenummer
AB8F  20 1E E1  JSR $E11E   CHKIN, Eingabe vorbereiten
AB92  A2 01     LDX #$01    Zeiger auf
AB94  A0 02     LDY #$02    Pufferende = $201 ein Zeichen
AB96  A9 00     LDA #$00    Wert laden und
AB98  8D 01 02  STA $0201   Puffer mit $0 abschließen
AB9B  A9 40     LDA #$40    GET-Flag
AB9D  20 0F AC  JSR $AC0F   Wertzuweisung an Variable
ABA0  A6 13     LDX $13     Eingabegerät
ABA2  D0 13     BNE $ABB7   nicht Tastatur, dann CLRCH
ABA4  60        RTS         Rücksprung

****************************  BASIC-Befehl  INPUT#
ABA5  20 9E B7  JSR $B79E   holt Byte-Wert
ABA8  A9 2C     LDA #$2C    ',' Code für Komma
ABAA  20 FF AE  JSR $AEFF   prüft auf Komma
ABAD  86 13     STX $13     Eingabegerät
ABAF  20 1E E1  JSR $E11E   CHKIN, Eingabe vorbereiten
ABB2  20 CE AB  JSR $ABCE   INPUT ohne Dialogstring

Einsprung von $AA83, $ABE4

ABB5  A5 13     LDA $13     Eingabegerät im Akku
ABB7  20 CC FF  JSR $FFCC   setzt Eingabegerät zurück
ABBA  A2 00     LDX #$00    Wert laden und
ABBC  86 13     STX $13     Eingabegerät wieder Tastatur
ABBE  60        RTS         Rücksprung
```

```
**************************** BASIC-Befehl  INPUT
ABBF   C9 22      CMP #$22    '"'  Hochkomma?
ABC1   D0 0B      BNE $ABCE   nein: $ABDE
ABC3   20 BD AE   JSR $AEBD   Dialogstring holen
ABC6   A9 3B      LDA #$3B    ';' Semikolon
ABC8   20 FF AE   JSR $AEFF   prüft auf Code
ABCB   20 21 AB   JSR $AB21   String ausgeben

Einsprung von $ABB2

ABCE   20 A6 B3   JSR $B3A6   prüft auf Direkt-Modus
ABD1   A9 2C      LDA #$2C    ',' Komma
ABD3   8D FF 01   STA $01FF   an Pufferstart
ABD6   20 F9 AB   JSR $ABF9   Fragezeichen ausgeben
ABD9   A5 13      LDA $13     Nummer des Eingabegeräts
ABDB   F0 0D      BEQ $ABEA   Tastatur? ja: $ABEA
ABDD   20 B7 FF   JSR $FFB7   Status holen
ABE0   29 02      AND #$02    Bit 1 isolieren (Tineout R.)
ABE2   F0 06      BEQ $ABEA   Time-out?
ABE4   20 B5 AB   JSR $ABB5   ja: CLRCH,Tastatur aktivieren
ABE7   4C F8 A8   JMP $A8F8   nächstes Statement ausführen

ABEA   AD 00 02   LDA $0200   erstes Zeichen holen
ABED   D0 1E      BNE $AC0D   Ende?
ABEF   A5 13      LDA $13     ja: Eingabegerät
ABF1   D0 E3      BNE $ABD6   nicht Tastatur: $ABD6
ABF3   20 06 A9   JSR $A906   Offset (Statement) suchen
ABF6   4C FB A8   JMP $A8FB   Programmzeiger auf Statement

Einsprung von $ABD6, $AC4A

ABF9   A5 13      LDA $13     Eingabegerät holen
ABFB   D0 06      BNE $AC03   nicht Tastatur: $AC03
ABFD   20 45 AB   JSR $AB45   '?' ausgeben
AC00   20 3B AB   JSR $AB3B   ' ' Leerzeichen ausgeben
AC03   4C 60 A5   JMP $A560   Eingabezeile holen

**************************** BASIC-Befehl  READ
AC06   A6 41      LDX $41     DATA-Zeiger nach
AC08   A4 42      LDY $42     $41/42 holen
```

```
AC0A   A9 98      LDA #$98    READ-Flag
AC0C   2C         .BYTE $2C
AC0D   A9 00      LDA #$00    Flagwert laden

Einsprung von $AB9D

AC0F   85 11      STA $11     und INPUT-Zeiger setzen
AC11   86 43      STX $43     INPUT-Zeiger auf
AC13   84 44      STY $44     Eingabequelle setzen

Einsprung von $ACB5

AC15   20 8B B0   JSR $B08B   sucht Variable
AC18   85 49      STA $49     Variablenadresse
AC1A   84 4A      STY $4A     speichern
AC1C   A5 7A      LDA $7A     LOW- und HIGH-Byte des
AC1E   A4 7B      LDY $7B     Programmzeigers
AC20   85 4B      STA $4B     in $4B/$4C
AC22   84 4C      STY $4C     zwischenspeichern
AC24   A6 43      LDX $43     INPUT-Zeiger
AC26   A4 44      LDY $44     (LOW und HIGH)
AC28   86 7A      STX $7A     als Programmzeiger
AC2A   84 7B      STY $7B     abspeichern
AC2C   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen
AC2F   D0 20      BNE $AC51   Endzeichen? nein: $AC51
AC31   24 11      BIT $11     Eingabeflag
AC33   50 0C      BVC $AC41   kein GET: $AC41
AC35   20 24 E1   JSR $E124   GETIN
AC38   8D 00 02   STA $0200   Zeichen in Puffer schreiben
AC3B   A2 FF      LDX #$FF    Zeiger auf
AC3D   A0 01      LDY #$01    Puffer setzen
AC3F   D0 0C      BNE $AC4D   unbedingter Sprung
AC41   30 75      BMI $ACB8   READ: $ACB8
AC43   A5 13      LDA $13     Eingabegerät holen
AC45   D0 03      BNE $AC4A   nicht Tastatur: $AC4A
AC47   20 45 AB   JSR $AB45   Fragezeichen ausgeben
AC4A   20 F9 AB   JSR $ABF9   zweites Fragezeichen ausgeben
AC4D   86 7A      STX $7A     Programmzeiger setzen
AC4F   84 7B      STY $7B     (LOW und HIGH)
```

```
Einsprung von $ACDC

AC51  20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
AC54  24 0D      BIT $0D     Typ-Flag
AC56  10 31      BPL $AC89   kein String: $AC89
AC58  24 11      BIT $11     Eingabeflag
AC5A  50 09      BVC $AC65   kein GET: $AC65
AC5C  E8         INX         Programmzeiger erhöhen
AC5D  86 7A      STX $7A     und neu setzen ($0200)
AC5F  A9 00      LDA #$00    Wert laden und
AC61  85 07      STA $07     Trennzeichen setzen
AC63  F0 0C      BEQ $AC71   unbedingter Sprung
AC65  85 07      STA $07     nächstes Zeichen
AC67  C9 22      CMP #$22    '"'  Hochkomma?
AC69  F0 07      BEQ $AC72   ja: $AC72
AC6B  A9 3A      LDA #$3A    ':' Doppelpunktcode laden
AC6D  85 07      STA $07     und abspeichern
AC6F  A9 2C      LDA #$2C    ',' Kommacode (Endzeichen
AC71  18         CLC         für Stringübertragung)
AC72  85 08      STA $08     abspeichern
AC74  A5 7A      LDA $7A     Programmzeiger laden
AC76  A4 7B      LDY $7B     (LOW und HIGH)
AC78  69 00      ADC #$00    und Übertrag addieren
AC7A  90 01      BCC $AC7D   C = 0: $AC7D
AC7C  C8         INY         bei '"' um 1 erhöhen
AC7D  20 8D B4   JSR $B48D   String übernehmen
AC80  20 E2 B7   JSR $B7E2   Programmzeiger hinter String
AC83  20 DA A9   JSR $A9DA   String an Variable zuweisen
AC86  4C 91 AC   JMP $AC91   weiter machen
AC89  20 F3 BC   JSR $BCF3   Ziffernstring in FAC holen
AC8C  A5 0E      LDA $0E     INTEGER/REAL-Flag
AC8E  20 C2 A9   JSR $A9C2   FAC an numerische Variable

Einsprung von $AC86

AC91  20 79 00   JSR $0079   CHRGOT: letztes Zeichen holen
AC94  F0 07      BEQ $AC9D   Ende?
AC96  C9 2C      CMP #$2C    ',' Code?
AC98  F0 03      BEQ $AC9D   ja: $AC9D
AC9A  4C 4D AB   JMP $AB4D   zur Fehlerbehandlung
```

```
AC9D  A5 7A     LDA $7A      Programmzeiger
AC9F  A4 7B     LDY $7B      holen und
ACA1  85 43     STA $43      in DATA-Zeiger
ACA3  84 44     STY $44      abspeichern
ACA5  A5 4B     LDA $4B      ursprüngliche
ACA7  A4 4C     LDY $4C      Programmzeiger
ACA9  85 7A     STA $7A      wieder zurückholen
ACAB  84 7B     STY $7B      und speichern
ACAD  20 79 00  JSR $0079    CHRGOT: letztes Zeichen holen
ACB0  F0 2D     BEQ $ACDF    Trennzeichen: $ACDF
ACB2  20 FD AE  JSR $AEFD    CKCOM: prüft auf Komma
ACB5  4C 15 AC  JMP $AC15    weiter
ACB8  20 06 A9  JSR $A906    nächstes Statement suchen
ACBB  C8        INY          Offset erhöhen
ACBC  AA        TAX          Zeilenende?
ACBD  D0 12     BNE $ACD1    nein: $ACD1
ACBF  A2 0D     LDX #$0D     'OUT OF DATA' Code
ACC1  C8        INY          Zeiger erhöhen
ACC2  B1 7A     LDA ($7A),Y  Programmende?
ACC4  F0 6C     BEQ $AD32    ja: 'OUT OF DATA', X = 0
ACC6  C8        INY          Zeiger erhöhen
ACC7  B1 7A     LDA ($7A),Y  Zeilennummer (LOW) holen
ACC9  85 3F     STA $3F      und abspeichern
ACCB  C8        INY          Zeiger erhöhen
ACCC  B1 7A     LDA ($7A),Y  Zeilenummer (HIGH)
ACCE  C8        INY          Zeiger erhöhen
ACCF  85 40     STA $40      Zeilennummer speichern
ACD1  20 FB A8  JSR $A8FB    Programmz. auf Statement
ACD4  20 79 00  JSR $0079    CHRGOT letztes Zeichen holen
ACD7  AA        TAX          und ins X-Reg.
ACD8  E0 83     CPX #$83     'DATA' Code?
ACDA  D0 DC     BNE $ACB8    nein: weitersuchen
ACDC  4C 51 AC  JMP $AC51    Daten lesen
ACDF  A5 43     LDA $43      LOW- und HIGH-Byte des
ACE1  A4 44     LDY $44      Input-Zeigers
ACE3  A6 11     LDX $11      Eingabe-Flag
ACE5  10 03     BPL $ACEA    kein DATA: $ACEA
ACE7  4C 27 A8  JMP $A827    DATA-Zeiger setzen
ACEA  A0 00     LDY #$00     Zeiger setzen
ACEC  B1 43     LDA ($43),Y  nächstes Zeichen holen
```

```
ACEE   F0 0B      BEQ $ACFB   Endzeichen: $ACFB
ACF0   A5 13      LDA $13     Eingabe über Tastatur?
ACF2   D0 07      BNE $ACFB   nein: $ACFB
ACF4   A9 FC      LDA #$FC    Zeiger auf
ACF6   A0 AC      LDY #$AC    '?extra ignored' setzen
ACF8   4C 1E AB   JMP $AB1E   String ausgeben
ACFB   60         RTS         Rücksprung

ACFC   3F 45 58 54 52 41 20 49  '?extra ignored'
AD04   47 4E 4F 52 45 44 0D 00
AD0C   3F 52 45 44 4F 20 46 52  '?redo from start'
AD14   4F 20 53 54 41 52 54 0D
AD1C   00

****************************  BASIC-Befehl  NEXT
AD1D   D0 04      BNE $AD24   folgt Variablenname? ja:$AD24
AD20   A0 00      LDY #$00    Variablenzeiger = 0
AD22   F0 03      BEQ $AD27   unbedingter Sprung

Einsprung von $AD87

AD24   20 8B B0   JSR $B08B   sucht Variable
AD27   85 49      STA $49     Adresse der
AD29   84 4A      STY $4A     Variablen speichern
AD2B   20 8A A3   JSR $A38A   sucht FOR-NEXT-Schleife
AD2E   F0 05      BEQ $AD35   gefunden: $AD35
AD30   A2 0A      LDX #$0A    Nummer für 'next without for'
AD32   4C 37 A4   JMP $A437   Fehlermeldung ausgeben
AD35   9A         TXS         X-Reg. retten
AD36   8A         TXA         X-Register nach Akku
AD37   18         CLC         Carry löschen (Addition)
AD38   69 04      ADC #$04    Zeiger auf Exponenten des
AD3A   48         PHA         STEP-Wert + 4 und retten
AD3B   69 06      ADC #$06    Zeiger auf Exponent des TO-
AD3D   85 24      STA $24     Wert und retten
AD3F   68         PLA         Akku wieder vom Stapel holen
AD40   A0 01      LDY #$01    Zeiger für Konstante setzen
AD42   20 A2 BB   JSR $BBA2   Variable vom Stapel nach FAC
AD45   BA         TSX         Stapelzeiger als Zeiger h.
AD46   BD 09 01   LDA $0109,X Vorzeichenbyte holen und
```

```
AD49  85 66     STA $66     für FAC speichern
AD4B  A5 49     LDA $49     Variablenadresse für
AD4D  A4 4A     LDY $4A     FOR-NEXT holen
AD4F  20 67 B8  JSR $B867   addiert STEP-Wert zu FAC
AD52  20 D0 BB  JSR $BBD0   FAC nach Variable bringen
AD55  A0 01     LDY #$01    Zeiger auf Konstante setzen
AD57  20 5D BC  JSR $BC5D   FAC mit Schleifenendwert vergleichen
AD5A  BA        TSX         Stapelzeiger als Zeiger h.
AD5B  38        SEC         Carry setzen (Subtraktion)
AD5C  FD 09 01  SBC $0109,X Stapelwert größer?
AD5F  F0 17     BEQ $AD78   ja: Schleife verlassen
AD61  BD 0F 01  LDA $010F,X Zeilennummer des Schleifen-
AD64  85 39     STA $39     anfangs holen (LOW- und
AD66  BD 10 01  LDA $0110,X HIGH-Byte) und als aktuelle
AD69  85 3A     STA $3A     BASIC-Zeilennummer speichern
AD6B  BD 12 01  LDA $0112,X Schleifenanfang holen (LOW-
AD6E  85 7A     STA $7A     und HIGH-Byte) und
AD70  BD 11 01  LDA $0111,X als neuen Programmzeiger
AD73  85 7B     STA $7B     abspeichern
AD75  4C AE A7  JMP $A7AE   zur Interpreterschleife
AD78  8A        TXA         Zeiger in Akku holen
AD79  69 11     ADC #$11    (Werte der Schleife aus
AD7B  AA        TAX         Stapel entfernen)
AD7C  9A        TXS         neuen Stapelzeiger setzen
AD7D  20 79 00  JSR $0079   CHRGOT letztes Zeichen holen
AD80  C9 2C     CMP #$2C    ',' Komma?
AD82  D0 F1     BNE $AD75   nein: dann fertig
AD84  20 73 00  JSR $0073   CHRGET nächstes Zeichen holen
AD87  20 24 AD  JSR $AD24   nächste  NEXT-Variable

**************************** FRMNUM  Ausdruck holen und
                            auf numerisch prüfen

Einsprung von $A775, $A79C, $B438, $B79E, B7EB, $E12A

AD8A  20 9E AD  JSR $AD9E   FRMEVL  Term holen
```

```
****************************  prüft auf numerisch

Einsprung von $A772, $ADF6, $AE61, $AFE3, B1B8, $B3C3
$B3F1, $B400, $B465

AD8D   18        CLC         Flag für Test auf numerisch
AD8E   24        .BYTE $24

****************************  prüft auf String

Einsprung von $AFBA, $B646, $B6A3

AD8F   38        SEC         Flag für Test auf String

Einsprung von $A9BC, $B016

AD90   24 0D     BIT $0D     Typflag testen
AD92   30 03     BMI $AD97   gesetzt: $AD97
AD94   B0 03     BCS $AD99   C=1: 'TYPE MISMATCH'
AD96   60        RTS         Rücksprung
AD97   B0 FD     BCS $AD96   C=1: RTS
AD99   A2 16     LDX #$16    Nummer für 'TYPE MISMATCH'
AD9B   4C 37 A4  JMP $A437   Fehlermeldung ausgeben

****************************  FRMEVL  auswerten eines
                              beliebigen Ausdrucks

Einsprung von $A928, $A9B7, $AAB5, $AD8A, $AEF4, $AFB4
$B1B5, $E257

AD9E   A6 7A     LDX $7A     Programmzeiger (LOW) = 0?
ADA0   D0 02     BNE $ADA4   ja: HIGH-B. nicht vermindern
ADA2   C6 7B     DEC $7B     HIGH-Byte vermindern
ADA4   C6 7A     DEC $7A     LOW-Byte vermindern
ADA6   A2 00     LDX #$00    Prioritätswert laden
ADA8   24        .BYTE $24
```

```
Einsprung von $AE2D

ADA9  48        PHA         Operatormaske retten
ADAA  8A        TXA         Prioritätswert in Akku
ADAB  48        PHA         schieben und retten
ADAC  A9 01     LDA #$01    2 Bytes
ADAE  20 FB A3  JSR $A3FB   prüft auf Platz im Stapel
ADB1  20 83 AE  JSR $AE83   Nächstes Element holen
ADB4  A9 00     LDA #$00    Wert laden und
ADB6  85 4D     STA $4D     Maske für Vergleichsoperator

Einsprung von $B677

ADB8  20 79 00  JSR $0079   CHRGOT letztes Zeichen holen

Einsprung von $ADD4

ADBB  38        SEC         Carry setzen (Subtraktion)
ADBC  E9 B1     SBC #$B1    $B1 von Operatorcode subtr.
ADBE  90 17     BCC $ADD7   C=0: $ADD7
ADC0  C9 03     CMP #$03    mit $3 vergleichen
ADC2  B0 13     BCS $ADD7   =3: $ADD7
ADC4  C9 01     CMP #$01
ADC6  2A        ROL         Maske für kleiner
ADC7  49 01     EOR #$01    gleich und größer
ADC9  45 4D     EOR $4D     für Bits 0,1 und 2
ADCB  C5 4D     CMP $4D     in $4D erstellen
ADCD  90 61     BCC $AE30   (Wenn Codes von 177
ADCF  85 4D     STA $4D     bis 179 folgen)
ADD1  20 73 00  JSR $0073   CHRGET nächstes Zeichen holen
ADD4  4C BB AD  JMP $ADBB   nächstes Zeichen auswerten
ADD7  A6 4D     LDX $4D     Operatormaske holen
ADD9  D0 2C     BNE $AE07   gleich 0? nein: $AE07
ADDB  B0 7B     BCS $AE58   Code größer oder gleich 180?
ADDD  69 07     ADC #$07    Code kleiner 170?
ADDF  90 77     BCC $AE58   ja: $AE58
ADE1  65 0D     ADC $0D     Stringaddition?
ADE3  D0 03     BNE $ADE8   nein: Verkettung umgehen
ADE5  4C 3D B6  JMP $B63D   Stringverkettung
ADE8  69 FF     ADC #$FF    Code-$AA (wiederherstellen)
```

```
ADEA   85 22      STA $22      und speichern
ADEC   0A         ASL          verdoppeln
ADED   65 22      ADC $22      + Wert (also mal 3)
ADEF   A8         TAY          als Zeiger ins Y-Register
ADF0   68         PLA          bisheriger Prioritätswert
ADF1   D9 80 A0   CMP $A080,Y  mit Prioritätsw. vergleichen
ADF4   B0 67      BCS $AE5D    größer: $AE5D
ADF6   20 8D AD   JSR $AD8D    prüft auf numerisch
ADF9   48         PHA          Prioritätswert retten

Einsprung von $AF11

ADFA   20 20 AE   JSR $AE20    Operatoradr. und Operanden r.
ADFD   68         PLA
ADFE   A4 4B      LDY $4B      Operator?
AE00   10 17      BPL $AE19    ja: $AE19
AE02   AA         TAX          weitere Operation?
AE03   F0 56      BEQ $AE5B    nein: RTS
AE05   D0 5F      BNE $AE66    ARG vom Stapel holen
AE07   46 0D      LSR $0D      Stringflag löschen
AE09   8A         TXA          Operatormaske nach
AE0A   2A         ROL          links schieben
AE0B   A6 7A      LDX $7A      Programmzeiger holen (LOW)
AE0D   D0 02      BNE $AE11    =0: HIGH-Byte vermindern
AE0F   C6 7B      DEC $7B      HIGH-Byte vermindern
AE11   C6 7A      DEC $7A      LOW-Byte vermindern
AE13   A0 1B      LDY #$1B     Offset des Hierarchieflags
AE15   85 4D      STA $4D      Flag setzen
AE17   D0 D7      BNE $ADF0    unbedingter Sprung
AE19   D9 80 A0   CMP $A080,Y  mit Hierarchieflag vergl.
AE1C   B0 48      BCS $AE66    größer: $AE66
AE1E   90 D9      BCC $ADF9    sonst weiter

Einsprung von $ADFA

AE20   B9 82 A0   LDA $A082,Y  Operationsadresse (HIGH)
AE23   48         PHA          auf Stapel retten
AE24   B9 81 A0   LDA $A081,Y  Operationsadresse (LOW)
AE27   48         PHA          auf Stapel retten
AE28   20 33 AE   JSR $AE33    Operanden auf Stapel retten
```

```
AE2B   A5 4D      LDA $4D      Operatormaske laden
AE2D   4C A9 AD   JMP $ADA9    zum Schleifenanfang
AE30   4C 08 AF   JMP $AF08    gibt 'SYNTAX ERROR'

Einsprung von $AE28

AE33   A5 66      LDA $66      Vorzeichen von FAC
AE35   BE 80 A0   LDX $A080,Y  Hierarchieflag

Einsprung von $A7A2

AE38   A8         TAY          Vorzeichen ins Y-Reg.
AE39   68         PLA          Rücksprungadresse holen
AE3A   85 22      STA $22      und merken
AE3C   E6 22      INC $22      Rücksprungadresse erhöhen
AE3E   68         PLA          nächstes Adressbyte holen
AE3F   85 23      STA $23      und speichern
AE41   98         TYA          Vorzeichen wieder in Akku
AE42   48         PHA          und auf Stapel legen

Einsprung von $A788

AE43   20 1B BC   JSR $BC1B    FAC runden
AE46   A5 65      LDA $65      FAC auf Stapel legen
AE48   48         PHA          1. Byte retten
AE49   A5 64      LDA $64      2.Byte holen
AE4B   48         PHA          und retten
AE4C   A5 63      LDA $63      3. Byte holen
AE4E   48         PHA          und retten
AE4F   A5 62      LDA $62      4.Byte holen
AE51   48         PHA          und retten
AE52   A5 61      LDA $61      5.Byte holen
AE54   48         PHA          und retten
AE55   6C 22 00   JMP ($0022)  Sprung auf Operation
AE58   A0 FF      LDY #$FF     Flagwert für Operator
AE5A   68         PLA          Prioritätsflag retten
AE5B   F0 23      BEQ $AE80    =0? ja: $AE80
AE5D   C9 64      CMP #$64     =$64?
AE5F   F0 03      BEQ $AE64    ja: $AE64
AE61   20 8D AD   JSR $AD8D    prüft auf numerisch
```

```
AE64   84 4B      STY $4B     Flag für Operator
AE66   68         PLA         Akku vom Stapel holen
AE67   4A         LSR         halbieren
AE68   85 12      STA $12     und abspeichern
AE6A   68         PLA         ARG von Stapel holen
AE6B   85 69      STA $69     1. Byte speichern
AE6D   68         PLA         2. Byte holen
AE6E   85 6A      STA $6A     und speichern
AE70   68         PLA         3. Byte holen
AE71   85 6B      STA $6B     und speichern
AE73   68         PLA         4. Byte holen
AE74   85 6C      STA $6C     und speichern
AE76   68         PLA         5. Byte holen
AE77   85 6D      STA $6D     und speichern
AE79   68         PLA         6. Byte (Vorzeichen holen
AE7A   85 6E      STA $6E     und speichern
AE7C   45 66      EOR $66     Vorzeichen von ARG und FAC
AE7E   85 6F      STA $6F     verknüpfen und speichern
AE80   A5 61      LDA $61     Exponentbyte von FAC laden
AE82   60         RTS         Rücksprung

**************************** Nächstes Element eines
                               Ausdrucks holen

Einsprung von $ADB1, $B643

AE83   6C 0A 03   JMP ($030A) JMP $AE86

Einsprung von $AE83

AE86   A9 00      LDA #$00    Wert laden und damit
AE88   85 0D      STA $0D     Typflag auf numerisch setzen
AE8A   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
AE8D   B0 03      BCS $AE92   Ziffer? nein: $AE92
AE8F   4C F3 BC   JMP $BCF3   Variable nach FAC holen
AE92   20 13 B1   JSR $B113   Buchstabe?
AE95   90 03      BCC $AE9A   nein: JMP umgehen
AE97   4C 28 AF   JMP $AF28   Variable holen
AE9A   C9 FF      CMP #$FF    BASIC-Code für Pi?
AE9C   D0 0F      BNE $AEAD   nein: $AEAD
```

```
AE9E   A9 A8      LDA #$A8    Zeiger auf Konstante Pi
AEA0   A0 AE      LDY #$AE    (LOW und HIGH-Byte)
AEA2   20 A2 BB   JSR $BBA2   Konstante in FAC holen
AEA5   4C 73 00   JMP $0073   CHRGET nächstes Zeichen holen

****************************
AEA8   82 49 0F DA A1         Konstante Pi  3.14159265

****************************
AEAD   C9 2E      CMP #$2E    '.' Dezimalpunkt?
AEAF   F0 DE      BEQ $AE8F   ja: $AE8F
AEB1   C9 AB      CMP #$AB    '-'?
AEB3   F0 58      BEQ $AF0D   zum Vorzeichenwechsel
AEB5   C9 AA      CMP #$AA    '+'?
AEB7   F0 D1      BEQ $AE8A   ja: $Ae8A
AEB9   C9 22      CMP #$22    '"'?
AEBB   D0 0F      BNE $AECC   nein: $AECC
AEBD   A5 7A      LDA $7A     LOW- und HIGH-Byte des
AEBF   A4 7B      LDY $7B     Programmzeigers holen
AEC1   69 00      ADC #$00    und Übertrag addieren
AEC3   90 01      BCC $AEC6   C=0: $AEC6
AEC5   C8         INY         HIGH-Byte erhöhen
AEC6   20 87 B4   JSR $B487   String übertragen
AEC9   4C E2 B7   JMP $B7E2   Programmz. auf Stringende +1
AECC   C9 A8      CMP #$A8    'NOT'-Code?
AECE   D0 13      BNE $AEE3   nein: $AEE3
AED0   A0 18      LDY #$18    Offset des H.Flags in Tabelle
AED2   D0 3B      BNE $AF0F   unbedingter Sprung

****************************  BASIC-Befehl  NOT
AED4   20 BF B1   JSR $B1BF   FAC nach INTEGER wandeln
AED7   A5 65      LDA $65     HIGH-Byte holen
AED9   49 FF      EOR #$FF    alle Bits umdrehen
AEDB   A8         TAY         und ins Y-Reg.
AEDC   A5 64      LDA $64     LOW-Byte holen
AEDE   49 FF      EOR #$FF    alle Bits invertieren
AEE0   4C 91 B3   JMP $B391   nach Fließkomma wandeln
```

```
;*****************************
AEE3   C9 A5      CMP #$A5    'FN'-Code?
AEE5   D0 03      BNE $AEEA   nein: $AEEA
AEE7   4C F4 B3   JMP $B3F4   FN ausführen

;*****************************
AEEA   C9 B4      CMP #$B4    'SGN'-Code
AEEC   90 03      BCC $AEF1   kleiner (keine Stringfunkt.)?
AEEE   4C A7 AF   JMP $AFA7   holt String ,ersten Parameter

;*****************************  holt Term in Klammern

Einsprung von $B3FD

AEF1   20 FA AE   JSR $AEFA   prüft auf Klammer auf
AEF4   20 9E AD   JSR $AD9E   FRMEVL  holt Term

;*****************************  prüft auf Zeichen im B.-Text

Einsprung von $B20B, $B3C6, $B761

AEF7   A9 29      LDA #$29    ')'   Klammer zu
AEF9   2C         .BYTE $2C

Einsprung von $AEF1, $AFB1, $B3B9

AEFA   A9 28      LDA #$28    '('   Klammer auf
AEFC   2C         .BYTE $2C

Einsprung von $ACB2, $AFB7, $B07E, $B742, B7F1, $E20E

AEFD   A9 2C      LDA #$2C    ','   Komma

Einsprung von $A76F, $A817, $A934, $A9AE, $AA8D, AB8A
$ABAA, $ABC8, $B3CB, $B3E3

AEFF   A0 00      LDY #$00    Zeiger setzen
AF01   D1 7A      CMP ($7A),Y mit laufendem Zeichen vergl.
AF03   D0 03      BNE $AF08   keine Übereinstimmung?
AF05   4C 73 00   JMP $0073   CHRGET nächstes Zeichen holen
```

```
Einsprung von $A80B, $A8E8, $AB5F, $AE30, $B09C, $B138
$B446, $E216

AF08   A2 0B      LDX #$0B    Nummer für 'SYNTAX ERROR'
AF0A   4C 37 A4   JMP $A437   Fehlermeldung ausgeben

AF0D   A0 15      LDY #$15    Offset Hierachie-Code für VZW
AF0F   68         PLA         nächsten 2 Bytes vom
AF10   68         PLA         Stapel entfernen
AF11   4C FA AD   JMP $ADFA   zur Auswertung

**************************** prüft auf Variable
                              innerhalb des BASICs

Einsprung von $AF3B, $AF6E

AF14   38         SEC         Carry setzen (Subtr.)
AF15   A5 64      LDA $64     Descriptor holen
AF17   E9 00      SBC #$00    liegt Descriptor ($64/$65)
AF19   A5 65      LDA $65     zwischen $A000 und $E32A?
AF1B   E9 A0      SBC #$A0
AF1D   90 08      BCC $AF27   ja: dann C=1, sonst RTS
AF1F   A9 A2      LDA #$A2    1. Wert laden
AF21   E5 64      SBC $64     1. Descriptorbyte abziehen
AF23   A9 E3      LDA #$E3    2. Wert laden
AF25   E5 65      SBC $65     und Descriptorwert abziehen
AF27   60         RTS         Rücksprung

**************************** Variable holen

Einsprung von $AE97

AF28   20 8B B0   JSR $B08B   Variable suchen
AF2B   85 64      STA $64     Zeiger auf Variable
AF2D   84 65      STY $65     bzw. Stringdescriptor
AF2F   A6 45      LDX $45     als
AF31   A4 46      LDY $46     Variablenname speichern
AF33   A5 0D      LDA $0D     Typflag holen
AF35   F0 26      BEQ $AF5D   numerisch?
```

```
AF37  A9 00     LDA #$00    Wert laden und
AF39  85 70     STA $70     in Rundungsbyte für FAC
AF3B  20 14 AF  JSR $AF14   Descriptor im Interpreter?
AF3E  90 1C     BCC $AF5C   nein
AF40  E0 54     CPX #$54    'T'? (von TI$)
AF42  D0 18     BNE $AF5C   nein: $AF5C
AF44  C0 C9     CPY #$C9    'I$'? (von TI$)
AF46  D0 14     BNE $AF5C   nein: $AF5C
AF48  20 84 AF  JSR $AF84   Zeit nach FAC holen
AF4B  84 5E     STY $5E     Flag für Exponentialdarst. =0
AF4D  88        DEY         vermindern (=$FF)
AF4E  84 71     STY $71     Zeiger für Stringstartadresse
AF50  A0 06     LDY #$06    Länge 6 für TI$
AF52  84 5D     STY $5D     speichern
AF54  A0 24     LDY #$24    Zeiger auf Stellenwert
AF56  20 68 BE  JSR $BE68   erzeugt String TI$
AF59  4C 6F B4  JMP $B46F   bringt String in Str.bereich
AF5C  60        RTS         Rücksprung

AF5D  24 0E     BIT $0E     INTEGER/ REAL Flag
AF5F  10 0D     BPL $AF6E   REAL? ja: $AF6E

**************************  Integervariable holen
AF61  A0 00     LDY #$00    Zeiger setzen
AF63  B1 64     LDA ($64),Y Intgerzahl holen (1. Byte)
AF65  AA        TAX         ins X-Reg.
AF66  C8        INY         Zeiger erhöhen
AF67  B1 64     LDA ($64),Y 2. Byte holen
AF69  A8        TAY         ins Y-Register
AF6A  8A        TXA         1. Byte in Akku holen
AF6B  4C 91 B3  JMP $B391   und nach Fließkomma wandeln

**************************  REAL-Variable holen
AF6E  20 14 AF  JSR $AF14   Descriptor im Interpreter?
AF71  90 2D     BCC $AFA0   nein
AF73  E0 54     CPX #$54    'T'? (von TI)
AF75  D0 1B     BNE $AF92   nein: $AF92
AF77  C0 49     CPY #$49    'I'? (von TI)
AF79  D0 25     BNE $AFA0   nein: $AFA0
AF7B  20 84 AF  JSR $AF84   TIME in FAC holen
```

```
AF7E   98         TYA          Akku =0 setzen
AF7F   A2 A0      LDX #$A0     Exponentbyte für FAC
AF81   4C 4F BC   JMP $BC4F    FAC linksbündig machen

****************************** Zeit holen

Einsprung von $AF48, $AF7B

AF84   20 DE FF   JSR $FFDE    TIME holen
AF87   86 64      STX $64      1. Byte nach FAC
AF89   84 63      STY $63      2. Byte nach FAC
AF8B   85 65      STA $65      3. Byte nach FAC
AF8D   A0 00      LDY #$00     Wert laden (0) und
AF8F   84 62      STY $62      als 4. Byte nach FAC
AF91   60         RTS          Rücksprung

AF92   E0 53      CPX #$53     'S'?
AF94   D0 0A      BNE $AFA0    nein: $AFA0
AF96   C0 54      CPY #$54     'T'?
AF98   D0 06      BNE $AFA0    nein: $AFA0
AF9A   20 B7 FF   JSR $FFB7    Status holen
AF9D   4C 3C BC   JMP $BC3C    Byte in Fließkommaformat

****************************** REAL-Variable holen
AFA0   A5 64      LDA $64      LOW- und HIGH-Byte der
AFA2   A4 65      LDY $65      Variablenadresse
AFA4   4C A2 BB   JMP $BBA2    Variable in FAC holen

****************************** Funktionsberechnung

Einsprung von $AEEE

AFA7   0A         ASL          Funktionscode mal 2
AFA8   48         PHA          auf den Stapel retten
AFA9   AA         TAX          und ins X-Register
AFAA   20 73 00   JSR $0073    CHRGET nächstes Zeichen
AFAD   E0 8F      CPX #$8F     numerische Funktion?
AFAF   90 20      BCC $AFD1    ja: $AFD1
```

```
**************************** Stringfunktion, String und
                              ersten Parameter
AFB1   20 FA AE   JSR $AEFA   prüft auf Klammer auf
AFB4   20 9E AD   JSR $AD9E   FRMEVL holen beliebigen Term
AFB7   20 FD AE   JSR $AEFD   prüft auf Komma
AFBA   20 8F AD   JSR $AD8F   prüft auf String
AFBD   68         PLA         Funktionstoken left$, r$, m$
AFBE   AA         TAX         Akku nach X holen
AFBF   A5 65      LDA $65     Adresse des
AFC1   48         PHA         Stringdescriptors
AFC2   A5 64      LDA $64     holen und auf den Stapel
AFC4   48         PHA         retten (LOW und HIGH)
AFC5   8A         TXA         Akku wiederholen
AFC6   48         PHA         Token auf den Stapel retten
AFC7   20 9E B7   JSR $B79E   holt Byte-Wert (2. Parameter)
AFCA   68         PLA         Token zurückholen
AFCB   A8         TAY         und ins Y-Reg.
AFCC   8A         TXA         2. Bytewert in den Akku laden
AFCD   48         PHA         und auf den Stapel retten
AFCE   4C D6 AF   JMP $AFD6   Routine ausführen

**************************** numerische Funktion auswerten
AFD1   20 F1 AE   JSR $AEF1   holt Term in Klammern
AFD4   68         PLA         BASIC-Code für Funktion holen
AFD5   A8         TAY         und als Zeiger ins Y-Reg.

Einsprung von $AFCE

AFD6   B9 EA 9F   LDA $9FEA,Y Vektor für Funktionsbe-
AFD9   85 55      STA $55     rechnung holen und speichern
AFDB   B9 EB 9F   LDA $9FEB,Y 2.Byte holen
AFDE   85 56      STA $56     und speichern
AFE0   20 54 00   JSR $0054   Funktion ausführen
AFE3   4C 8D AD   JMP $AD8D   prüft auf numerisch

**************************** BASIC-Befehl  OR
AFE6   A0 FF      LDY #$FF    Flag für OR
AFE8   2C         .BYTE $2C
```

```
**************************** BASIC-Befehl  AND
AFE9   A0 00      LDY #$00    Flag für AND
AFEB   84 0B      STY $0B     Flag setzen
AFED   20 BF B1   JSR $B1BF   FAC nach INTEGER wandeln
AFF0   A5 64      LDA $64     ersten Wert holen
AFF2   45 0B      EOR $0B     mit Flag verknüpfen
AFF4   85 07      STA $07     und speichern
AFF6   A5 65      LDA $65     zweiten Wert holen
AFF8   45 0B      EOR $0B     mit Flag verknüpfen
AFFA   85 08      STA $08     und speichern
AFFC   20 FC BB   JSR $BBFC   ARG nach FAC
AFFF   20 BF B1   JSR $B1BF   FAC nach Integer
B002   A5 65      LDA $65     zweites Byte holen
B004   45 0B      EOR $0B     mit Flag verknüpfen
B006   25 08      AND $08     logische AND-Verknüpfung
B008   45 0B      EOR $0B     mit Flag verknüpfen
B00A   A8         TAY         ins Y-Reg. retten
B00B   A5 64      LDA $64     erstes Byte holen
B00D   45 0B      EOR $0B     mit Flag verknüpfen
B00F   25 07      AND $07     logische AND-Verknüpfung
B011   45 0B      EOR $0B     mit Flag verknüpfen
B013   4C 91 B3   JMP $B391   wieder in Fließkomma wandeln

**************************** Vergleich
B016   20 90 AD   JSR $AD90   prüft auf identischen Typ
B019   B0 13      BCS $B02E   String: dann weiter
B01B   A5 6E      LDA $6E     Wert holen
B01D   09 7F      ORA #$7F    ARG in Speicherformat
B01F   25 6A      AND $6A     wandeln und
B021   85 6A      STA $6A     wieder abspeichern
B023   A9 69      LDA #$69    Adresse von ARG
B025   A0 00      LDY #$00    (LOW- und HIGH-Byte)
B027   20 5B BC   JSR $BC5B   Vergleich ARG mit FAC
B02A   AA         TAX
B02B   4C 61 B0   JMP $B061   Ergebnis in FAC holen

**************************** Stringvergleich
B02E   A9 00      LDA #$00    Wert laden und damit
B030   85 0D      STA $0D     Stringflag löschen
B032   C6 4D      DEC $4D     Operatormaske - 1
```

```
B034  20 A6 B6   JSR $B6A6   FRESTR
B037  85 61      STA $61     Stringlänge holen
B039  86 62      STX $62     LOW- und HIGH-Byte der
B03B  84 63      STY $63     Stringadresse speichern
B03D  A5 6C      LDA $6C     LOW- und HIGH-Byte des
B03F  A4 6D      LDY $6D     Zeigers auf zweiten String
B041  20 AA B6   JSR $B6AA   FRESTR
B044  86 6C      STX $6C     Adresse des
B046  84 6D      STY $6D     2. Strings
B048  AA         TAX         Länge des 2.Strings merken
B049  38         SEC         Carry setzen (Subtraktion)
B04A  E5 61      SBC $61     Längen vergleichen
B04C  F0 08      BEQ $B056   gleich: $B056
B04E  A9 01      LDA #$01    Wert für: 1.String länger
B050  90 04      BCC $B056   2.String kürzer
B052  A6 61      LDX $61     Länge des 1.Strings
B054  A9 FF      LDA #$FF    Wert für: 1.String kürzer
B056  85 66      STA $66     Flag für gleichen String,
B058  A0 FF      LDY #$FF    wenn beide Strings identisch aber
B05A  E8         INX         ungleich lang sind
B05B  C8         INY         Zeiger erhöhen
B05C  CA         DEX         Stringende?
B05D  D0 07      BNE $B066   nein: weiter
B05F  A6 66      LDX $66     Vorzeichenbyte holen

Einsprung von $B02B

B061  30 0F      BMI $B072   negativ: $B072
B063  18         CLC         Carry löschen
B064  90 0C      BCC $B072   unbedingter Sprung
B066  B1 6C      LDA ($6C),Y Vergleich der Strings
B068  D1 62      CMP ($62),Y zeichenweise
B06A  F0 EF      BEQ $B05B   gleiche Zeichen: weiter
B06C  A2 FF      LDX #$FF    Wert laden
B06E  B0 02      BCS $B072   und Vergleich beenden
B070  A2 01      LDX #$01    Wert laden
B072  E8         INX         und um 1 erhöhen
B073  8A         TXA         Wert in den Akku
B074  2A         ROL         linksverschieben, Bit 1, 2=$1
B075  25 12      AND $12     mit Vorzeichen verknüpfen
```

```
B077   F0 02      BEQ $B07B   =0: $B07B
B079   A9 FF      LDA #$FF
B07B   4C 3C BC   JMP $BC3C   Ergebnis nach FAC holen

B07E   20 FD AE   JSR $AEFD   CHKCOM prüft auf Komma

**************************** BASIC-Befehl  DIM
B081   AA         TAX        nächstes Zeichen
B082   20 90 B0   JSR $B090   Variable dimensionieren
B085   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen
B088   D0 F4      BNE $B07E   nicht Ende: zur nächsten Var.
B08A   60         RTS        Rücksprung

**************************** Variable holen

Einsprung von $A9A5, $AC15, $AD24, $AF28, $B3C0

B08B   A2 00      LDX #$00    Flag für nicht dimensionieren
B08D   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen

Einsprung von $B082

B090   86 0C      STX $0C     DIM-Flag setzen

Einsprung von $B3EA

B092   85 45      STA $45     Variablenname
B094   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen
B097   20 13 B1   JSR $B113   prüft auf Buchstabe
B09A   B0 03      BCS $B09F   ja: $B09F
B09C   4C 08 AF   JMP $AF08   'SYNTAX ERROR'
B09F   A2 00      LDX #$00    Wert laden und damit
B0A1   86 0D      STX $0D     Stringflag löschen
B0A3   86 0E      STX $0E     Integerflag löschen
B0A5   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
B0A8   90 05      BCC $B0AF   Ziffer?
B0AA   20 13 B1   JSR $B113   prüft auf Buchstabe
B0AD   90 0B      BCC $B0BA   nein: $B0BA
B0AF   AA         TAX        zweiter Buchstabe des Names
B0B0   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
```

```
B0B3  90 FB      BCC $B0B0   Ziffer?
B0B5  20 13 B1   JSR $B113   prüft auf Buchstabe
B0B8  B0 F6      BCS $B0B0   ja: weitere Zeichen überlesen
B0BA  C9 24      CMP #$24    '$' Code?
B0BC  D0 06      BNE $B0C4   nein: $B0C4
B0BE  A9 FF      LDA #$FF    Wert laden und
B0C0  85 0D      STA $0D     Stringflag setzen
B0C2  D0 10      BNE $B0D4   Sprung
B0C4  C9 25      CMP #$25    '%' Code?
B0C6  D0 13      BNE $B0DB   nein: $B0DB
B0C8  A5 10      LDA $10     Integer erlaubt?
B0CA  D0 D0      BNE $B09C   nein: 'SYNTAX ERROR'
B0CC  A9 80      LDA #$80    Wert für Integer laden
B0CE  85 0E      STA $0E     und Integerflag setzen
B0D0  05 45      ORA $45     Bit 7 im 1.Zeichen setzen und
B0D2  85 45      STA $45     speichern (Bit7=1: Integer)
B0D4  8A         TXA         X nach Akku speichern
B0D5  09 80      ORA #$80    Bit 7 im 2.Buchstaben setzen
B0D7  AA         TAX         X-Reg. zurückholen
B0D8  20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
B0DB  86 46      STX $46     zweiten Buchstaben speichern
B0DD  38         SEC         Feldvariablen erlaubt?
B0DE  05 10      ORA $10     wenn nicht, Bit7 setzen
B0E0  E9 28      SBC #$28    '('-Wert abziehen
B0E2  D0 03      BNE $B0E7   nicht Klammer auf?
B0E4  4C D1 B1   JMP $B1D1   dimensionierte Variable holen
B0E7  A0 00      LDY #$00    Wert laden und
B0E9  84 10      STY $10     FN-Flag = 0 setzen
B0EB  A5 2D      LDA $2D     Zeiger auf Variablenanfang
B0ED  A6 2E      LDX $2E     holen (LOW und HIGH)
B0EF  86 60      STX $60     und zum
B0F1  85 5F      STA $5F     Suchen merken
B0F3  E4 30      CPX $30     Suchzeiger = Variablenanfang
B0F5  D0 04      BNE $B0FB   nein: $B0FB
B0F7  C5 2F      CMP $2F     Ende der Variablen erreicht?
B0F9  F0 22      BEQ $B11D   ja: nicht gefunden, anlegen
B0FB  A5 45      LDA $45     ersten Buchstaben des Namens

B0FD  D1 5F      CMP ($5F),Y mit Tabelle vergleichen
B0FF  D0 08      BNE $B109   nein: weitersuchen
```

```
B101   A5 46      LDA $46      zweiten Buchstaben
B103   C8         INY          Zeiger erhöhen
B104   D1 5F      CMP ($5F),Y  vergleichen
B106   F0 7D      BEQ $B185    gleich: gefunden
B108   88         DEY          Zeiger vermindern
B109   18         CLC          Carry setzen (Addition)
B10A   A5 5F      LDA $5F      Zeiger um 7
B10C   69 07      ADC #$07     erhöhen  (2+5 Byte REAL Var.)
B10E   90 E1      BCC $B0F1    (Länge eines V.-Eintrags)
B110   E8         INX          Übertrag addieren
B111   D0 DC      BNE $B0EF    weiter suchen

**************************** prüft auf Buchstabe

Einsprung von $AE92, $B097, $B0AA, $B0B5

B113   C9 41      CMP #$41     'A'-Code? (Buchstabencode)
B115   90 05      BCC $B11C    nein: $B11C sonst C = 0
B117   E9 5B      SBC #$5B     'Z'+1
B119   38         SEC          ja: dann C = 1
B11A   E9 A5      SBC #$A5     nein: dann C = 0
B11C   60         RTS          Rücksprung

**************************** Variable anlegen
B11D   68         PLA
B11E   48         PHA          Aufrufadresse prüfen
B11F   C9 2A      CMP #$2A     Aufruf von FRMEVL?
B121   D0 05      BNE $B128    nein: dann neu anlegen
B123   A9 13      LDA #$13     Zeiger auf Konstante 0
B125   A0 BF      LDY #$BF     (LOW und HIGH)
B127   60         RTS          Rücksprung
B128   A5 45      LDA $45      LOW- und HIGH-Byte
B12A   A4 46      LDY $46      des Variablennames
B12C   C9 54      CMP #$54     'T'-Code?
B12E   D0 0B      BNE $B13B    nein: $B13B
B130   C0 C9      CPY #$C9     'I$'-Code?
B132   F0 EF      BEQ $B123    ja: TI$
B134   C0 49      CPY #$49     'I'-Code?
B136   D0 03      BNE $B13B    nein: $B13B
B138   4C 08 AF   JMP $AF08    'SYNTAX ERROR'
```

```
B13B  C9 53     CMP #$53    'S'-Code?
B13D  D0 04     BNE $B143   nein: $B143
B13F  C0 54     CPY #$54    'T'-Code?
B141  F0 F5     BEQ $B138   ST, dann 'SYNTAX ERROR'
B143  A5 2F     LDA $2F     LOW- und HIGH-Byte des
B145  A4 30     LDY $30     Zeigers auf Arraytabelle
B147  85 5F     STA $5F     laden und
B149  84 60     STY $60     merken
B14B  A5 31     LDA $31     LOW- und HIGH-Byte des
B14D  A4 32     LDY $32     Zeigers auf Ende der
B14F  85 5A     STA $5A     Arraytabelle
B151  84 5B     STY $5B     merken
B153  18        CLC         Carry für Addition setzen
B154  69 07     ADC #$07    um 7 verschieben für Anlage
B156  90 01     BCC $B159   einer neuen Variablen
B158  C8        INY         Übertrag addieren
B159  85 58     STA $58     LOW- und HIGH-Byte des
B15B  84 59     STY $59     neuen Blockendes speichern
B15D  20 B8 A3  JSR $A3B8   Block verschieben
B160  A5 58     LDA $58     Werte
B162  A4 59     LDY $59     wiederholen
B164  C8        INY         und damit
B165  85 2F     STA $2F     Zeiger auf Arraytabelle
B167  84 30     STY $30     neu setzen
B169  A0 00     LDY #$00    Zeiger setzen
B16B  A5 45     LDA $45     erster Buchstabe des Namens
B16D  91 5F     STA ($5F),Y und speichern
B16F  C8        INY         Zeiger erhöhen,
B170  A5 46     LDA $46     zweiten Buchstaben holen
B172  91 5F     STA ($5F),Y und abspeichern
B174  A9 00     LDA #$00    Nullwert laden
B176  C8        INY         Zeiger erhöhen
B177  91 5F     STA ($5F),Y nächsten 5 Werte
B179  C8        INY         der Variable auf 0 setzen
B17A  91 5F     STA ($5F),Y 2. Byte speichern
B17C  C8        INY         Zeiger erhöhen
B17D  91 5F     STA ($5F),Y 3. Byte speichern
B17F  C8        INY         Zeiger erhöhen
B180  91 5F     STA ($5F),Y 4. Byte speichern
B182  C8        INY         Zeiger erhöhen
```

```
B183   91 5F      STA ($5F),Y 5. Byte speichern
B185   A5 5F      LDA $5F     Zeiger auf Variablenwert
B187   18         CLC         Carry löschen (Addition)
B188   69 02      ADC #$02    zwei für Namen addieren
B18A   A4 60      LDY $60     in Zeiger auf Variable
B18C   90 01      BCC $B18F
B18E   C8         INY         Zeiger auf erstes Byte
B18F   85 47      STA $47     als Variablenzeiger
B191   84 48      STY $48     nach $47/48 speichern
B193   60         RTS         Rücksprung

****************************  berechnet Zeiger auf erstes
                              Arrayelement

Einsprung von $B253, $B261

B194   A5 0B      LDA $0B     Anzahl der Dimensionen
B196   0A         ASL         mal 2
B197   69 05      ADC #$05    plus 5
B199   65 5F      ADC $5F     zu $5F und
B19B   A4 60      LDY $60     $60 addieren
B19D   90 01      BCC $B1A0   Erhöhung umgehen
B19F   C8         INY         Übertrag addieren
B1A0   85 58      STA $58     Ergebnis-Zeiger nach
B1A2   84 59      STY $59     $58/59 speichern
B1A4   60         RTS         Rücksprung

****************************
B1A5   90 80 00 00 00         Konstante  -32768

****************************  Umwandlung FAC nach Integer
B1AA   20 BF B1   JSR $B1BF   FAC nach Integer wandeln
B1AD   A5 64      LDA $64     LOW-Byte
B1AF   A4 65      LDY $65     HIGH-Byte
B1B1   60         RTS         Rücksprung

****************************  Ausdruck holen und
                              nach Integer
```

```
Einsprung von $B1E3

B1B7  20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
B1BA  20 9E AD   JSR $AD9E   FRMEVL, Ausdruck auswerten

Einsprung von $B7A1

B1BD  20 8D AD   JSR $AD8D   prüft auf numerisch
B1C0  A5 66      LDA $66     Vorzeichen?
B1C2  30 0D      BMI $B1CC   negativ: dann 'ILLEGAL QUANT'

Einsprung von $A9C7, $AED4, $AFED, $AFFF, $B1AA

B1C4  A5 61      LDA $61     Exponent
B1C6  C9 90      CMP #$90    Betrag größer 32768?
B1C8  90 09      BCC $B1CE   nein: $B1CE
B1CA  A9 A5      LDA #$A5    Zeiger auf
B1CC  A0 B1      LDY #$B1    Konstante -32768 setzen
B1CE  20 5B BC   JSR $BC5B   Vergleich FAC mit Konstante
B1D1  D0 7A      BNE $B248   ungleich: 'ILLEGAL QUANT'
B1D3  4C 9B BC   JMP $BC9B   wandelt Fließkomma in Integer

****************************  dimensionierte Variable holen

Einsprung von $B0E4

B1D6  A5 0C      LDA $0C     DIM Flag
B1D8  05 0E      ORA $0E     Integer Flag
B1DA  48         PHA         auf Stapel retten
B1DB  A5 0D      LDA $0D     String Flag
B1DD  48         PHA         auf Stapel retten
B1DE  A0 00      LDY #$00    Anzahl der Indizes
B1E0  98         TYA         in Akku und
B1E1  48         PHA         auf Stapel retten
B1E2  A5 46      LDA $46     2. Buchstabe des Variablenn.
B1E4  48         PHA         und retten
B1E5  A5 45      LDA $45     1. Buchstabe der Variablenn.
B1E7  48         PHA         retten
B1E8  20 B2 B1   JSR $B1B2   Index holen und nach Integer
B1EB  68         PLA         die zwei
```

```
B1E7  85 45     STA $45      Bytes des
B1E9  68        PLA          Variablennamens zurückholen
B1EA  85 46     STA $46      und wieder abspeichern
B1EC  68        PLA          Anzahl der Indizes
B1ED  A8        TAY          holen und ins Y-Reg.
B1EE  BA        TSX          Stapelzeiger als Zeiger setzen
B1EF  BD 02 01  LDA $0102,X  Variablenflags
B1F2  48        PHA          aus dem Stapel kopieren
B1F3  BD 01 01  LDA $0101,X  und oben auf den
B1F6  48        PHA          Stapel legen
B1F7  A5 64     LDA $64      anstelle der
B1F9  9D 02 01  STA $0102,X  Variablenflags
B1FC  A5 65     LDA $65      Index LOW und HIGH in
B1FE  9D 01 01  STA $0101,X  den Stapel kopieren
B201  C8        INY          Anzahl der Indizes erhöhen
B202  20 79 00  JSR $0079    CHRGOT letztes Zeichen holen
B205  C9 2C     CMP #$2C     ',' Komma?
B207  F0 D2     BEQ $B1DB    ja: dann nächsten Index
B209  84 0B     STY $0B      Anzahl der Indizes speichern
B20B  20 F7 AE  JSR $AEF7    prüft auf Klammer zu
B20E  68        PLA          Flags vom
B20F  85 0D     STA $0D      Stapel
B211  68        PLA          zurückholen
B212  85 0E     STA $0E      und abspeichern
B214  29 7F     AND #$7F     Integerflag herstellen
B216  85 0C     STA $0C      und abspeichern
B218  A6 2F     LDX $2F      LOW- und HIGH-Byte des
B21A  A5 30     LDA $30      Zeigers auf Arraytabelle
B21C  86 5F     STX $5F      holen und
B21E  85 60     STA $60      Zeiger merken
B220  C5 32     CMP $32      Ende erreicht?
B222  D0 04     BNE $B228    nein: weiter
B224  E4 31     CPX $31      mit Tabellenende vergleichen
B226  F0 39     BEQ $B261    ja: nicht gefunden, anlegen
B228  A0 00     LDY #$00     Zeiger setzen
B22A  B1 5F     LDA ($5F),Y  Namen aus Tabelle holen
B22C  C8        INY          Zeiger erhöhen
B22D  C5 45     CMP $45      mit ges. Namen vergleichen
B22F  D0 06     BNE $B237    ungleich: $B237
B231  A5 46     LDA $46      Vergleich mit
```

```
B255  D1 5F     CMP ($5F),Y zweitem Buchstaben
B257  F0 16     BEQ $B24D   gefunden: $B24D
B259  C8        INY         Zeiger erhöhen
B25A  B1 5F     LDA ($5F),Y Suchzeiger zur
B25C  18        CLC         Feldlänge
B25D  65 5F     ADC $5F     Addieren
B25F  AA        TAX         ergibt Zeiger auf
B260  C8        INY         nächstes Array
B261  B1 5F     LDA ($5F),Y gleiches System
B263  65 60     ADC $60     mit zweitem Byte
B265  90 D7     BCC $B21C   und weiter suchen

Einsprung von $B308

B267  A2 12     LDX #$12    Nummer für 'bad subscript'
B269  2C        .BYTE $2C

Einsprung von $AA24, $B798, $B9F1

B26A  A2 0E     LDX #$0E    Nummer für 'illegal quanti.'
B26C  4C 37 A4  JMP $A437   Fehlermeldung ausgeben

B26F  A2 13     LDX #$13    Nummer für 'redim'd array'
B271  A5 0C     LDA $0C     DIM-Flag null?
B273  D0 F7     BNE $B26C   nein: dann Fehlermeldung
B275  20 94 B1  JSR $B194   Zeiger auf 1.Arrayelement
B278  A5 0B     LDA $0B     Zahl der gefundenen Dimensio.
B27A  A0 04     LDY #$04    Zeiger setzen
B27C  D1 5F     CMP ($5F),Y mit Dimensionen des Arrays
                            vergleichen
B27E  D0 E7     BNE $B267   ungleich:  'bad subscript'
B280  4C EA B2  JMP $B2EA   sucht gewünschtes Element

**************************  Arrayvariable anlegen
B283  20 94 B1  JSR $B194   Länge des Arraykopfs
B286  20 08 A4  JSR $A408   prüft auf genügend Platz
B289  A0 00     LDY #$00    Zeiger für Polynom-
B28B  84 72     STY $72     auswertung neu setzen
B28D  A2 05     LDX #$05    Wert für Variablenlänge(REAL)
B28F  A5 45     LDA $45     erster Buchstabe des Namens
```

```
B26F  91 5F     STA ($5F),Y in Arraytabelle
B271  10 01     BPL $B274  kein Integer?
B273  CA        DEX        bei Integerzahl
B274  C8        INY        Bytes vermindern
B275  A5 46     LDA $46    zweiter Buchstabe
B277  91 5F     STA ($5F),Y in Tabelle schreiben
B279  10 02     BPL $B27D  kein String oder Integer?
B27B  CA        DEX        entgültige
B27C  CA        DEX        Variablenlänge herstellen
B27D  86 71     STX $71    und speichern (2, 3 oder 5)
B27F  A5 0B     LDA $0B    Anzahl der Dimensionen holen
B281  C8        INY        Zeiger
B282  C8        INY        um 3
B283  C8        INY        erhöhen
B284  91 5F     STA ($5F),Y im Arrayheader speichern
B286  A2 0B     LDX #$0B   11, Defaultwert für
B288  A9 00     LDA #$00   Dimensionierung
B28A  24 0C     BIT $0C    Aufruf durch DIM-Befehl?
B28C  50 08     BVC $B296  nein: $B296
B28E  68        PLA        Dimension vom Stapel holen
B28F  18        CLC        Carry löschen (Addition)
B290  69 01     ADC #$01   eins addieren
B292  AA        TAX        und ins X-Reg.
B293  68        PLA        2.Wert holen
B294  69 00     ADC #$00   Übertrag addieren
B296  C8        INY        Zeiger erhöhen
B297  91 5F     STA ($5F),Y und speichern
B299  C8        INY        Zeiger erhöhen
B29A  8A        TXA        1.Wert wieder in den Akku
B29B  91 5F     STA ($5F),Y und ebenfalls speichern
B29D  20 4C B3  JSR $B34C  Platz für Dimensionen berech.
B2A0  86 71     STX $71    LOW- und HIGH-Byte des
B2A2  85 72     STA $72    Variablenende-Zeigers merken
B2A4  A4 22     LDY $22    Zeiger auf Arrayheader
B2A6  C6 0B     DEC $0B    weitere Dimensionen?
B2A8  D0 DC     BNE $B286  ja: $B286 (Schleifenbeginn)
B2AA  65 59     ADC $59    Feldlänge plus Startadresse
B2AC  B0 5D     BCS $B30B  Überlauf: 'OUT OF MEMORY'
B2AE  85 59     STA $59    Wert wieder speichern
B2B0  A8        TAY        und ins Y-Reg. bringen
```

```
B2B1   8A          TXA          Variablenendzeiger in Akku
B2B2   65 58       ADC $58      2.Zeichen addieren
B2B4   90 03       BCC $B2B9    Überlauf: Platz prüfen
B2B6   C8          INY          Endadresse erhöhen
B2B7   F0 52       BEQ $B30B    Überlauf: 'OUT OF MEMORY'
B2B9   20 08 A4    JSR $A408    prüft auf Speicherplatz
B2BC   85 31       STA $31      Zeiger auf Ende
B2BE   84 32       STY $32      der Arraytabelle setzen
B2C0   A9 00       LDA #$00     Array mit Nullen füllen
B2C2   E6 72       INC $72
B2C4   A4 71       LDY $71      1.Schleifenende?
B2C6   F0 05       BEQ $B2CD    ja: $B2CD
B2C8   88          DEY          Zeiger vermindern
B2C9   91 58       STA ($58),Y  Nullwert setzen
B2CB   D0 FB       BNE $B2C8    fertig: $B2C8
B2CD   C6 59       DEC $59
B2CF   C6 72       DEC $72
B2D1   D0 F5       BNE $B2C8
B2D3   E6 59       INC $59
B2D5   38          SEC          Carry setzen (Subtr.)
B2D6   A5 31       LDA $31      Zeiger auf Feldende
B2D8   E5 5F       SBC $5F      - Zeiger auf Arrayheader
B2DA   A0 02       LDY #$02     Zeiger setzen
B2DC   91 5F       STA ($5F),Y  Arraylänge LOW
B2DE   A5 32       LDA $32      Zeiger auf Feldende
B2E0   C8          INY          Zeiger erhöhen
B2E1   E5 60       SBC $60      - Zeiger auf Arrayheader
B2E3   91 5F       STA ($5F),Y  Arraylänge HIGH
B2E5   A5 0C       LDA $0C      Aufruf vom DIM-Befehl?
B2E7   D0 62       BNE $B34B    ja: RTS

****************************   Arrayelement suchen
B2E9   C8          INY          Zeiger erhöhen

Einsprung von $B25E

B2EA   B1 5F       LDA ($5F),Y  Zahl der Dimensionen
B2EC   85 0B       STA $0B      speichern
B2EE   A9 00       LDA #$00     Nullwert laden und
B2F0   85 71       STA $71      Zeiger auf Polynom-
```

```
B2F2  85 72     STA $72     auswertung löschen
B2F4  C8        INY         Zeiger erhöhen
B2F5  68        PLA         1. Indexwert vom Stapel
B2F6  AA        TAX         holen und ins X-Reg. bringen
B2F7  85 64     STA $64     Wert speichern
B2F9  68        PLA         2. Indexwert holen
B2FA  85 65     STA $65     und speichern
B2FC  D1 5F     CMP ($5F),Y mit Wert im Array vergleichen
B2FE  90 0E     BCC $B30E   kleiner?
B300  D0 06     BNE $B308   größer: 'bad subscript'
B302  C8        INY         Zeiger erhöhen
B303  8A        TXA         1.Wert zurückholen
B304  D1 5F     CMP ($5F),Y LOW-Byte vergleichen
B306  90 07     BCC $B30F   kleiner: dann weiter
B308  4C 45 B2  JMP $B245   'bad subscript'
B30B  4C 35 A4  JMP $A435   'out of memory'

**************************  Berechnung der Adresse
                            eines Arrayelements
B30E  C8        INY         Zeiger erhöhen
B30F  A5 72     LDA $72     Zeiger auf Polynomausw.(HIGH)
B311  05 71     ORA $71     Zeiger auf Polynomausw.(LOW)
B313  18        CLC         Carry löschen
B314  F0 0A     BEQ $B320   Multiplikation umgehen
B316  20 4C B3  JSR $B34C   Multiplikation
B319  8A        TXA         (X/Y)=($71/72)*(($5F/60),Y)
B31A  65 64     ADC $64
B31C  AA        TAX         Akku zurück ins X-Reg.
B31D  98        TYA
B31E  A4 22     LDY $22     Zeiger in Arrayheader
B320  65 65     ADC $65
B322  86 71     STX $71
B324  C6 0B     DEC $0B     Anzahl der Dimensionen
B326  D0 CA     BNE $B2F2   mit nächstem Index weiter
B328  85 72     STA $72
B32A  A2 05     LDX #$05    Variablenlänge (5, REAL)
B32C  A5 45     LDA $45     erster Buchstabe des Namens
B32E  10 01     BPL $B331   Integer? nein: $B331
B330  CA        DEX         Länge vermindern
B331  A5 46     LDA $46     zweiter Buchstabe des Namens
```

```
B333   10 02      BPL $B337    FLP? ja: $B337
B335   CA         DEX          Länge 2 mal
B336   CA         DEX          vermindern
B337   86 28      STX $28      Länge der Variablen 2,3 oder5
B339   A9 00      LDA #$00     Wert laden und damit
B33B   20 55 B3   JSR $B355    Offset im Array berechnen
B33E   8A         TXA          zur Adresse des ersten
B33F   65 58      ADC $58      Elements addieren
B341   85 47      STA $47      ergibt Variablenadresse
B343   98         TYA          2.Byte in Akku holen
B344   65 59      ADC $59      addieren, ergibt
B346   85 48      STA $48      HIGH-Byte der Adresse
B348   A8         TAY          ins Y-Reg. bringen und
B349   A5 47      LDA $47      1.Byte wieder in Akku holen
B34B   60         RTS          Rücksprung

****************************  Hilfsroutine für
                              Arrayberechnung

Einsprung von $B29D, $B316

B34C   84 22      STY $22      Register merken
B34E   B1 5F      LDA ($5F),Y  1.Wert holen
B350   85 28      STA $28      und abspeichern
B352   88         DEY          Zeiger vermindern
B353   B1 5F      LDA ($5F),Y  2.Wert holen

Einsprung von $B33B

B355   85 29      STA $29      und abspeichern
B357   A9 10      LDA #$10     Wert laden und damit
B359   85 5D      STA $5D      Verschiebezähler setzen
B35B   A2 00      LDX #$00     LOW- und HIGH-Byte des Er-
B35D   A0 00      LDY #$00     gebnisregisters auf 0 setzen
B35F   8A         TXA          LOW-Byte in Akku holen und
B360   0A         ASL          um 1 Bit nach links schieben
B361   AA         TAX          Byte zurück ins X-Reg.
B362   98         TYA          HIGH-Byte in den Akku holen,
B363   2A         ROL  A       um 1 Bit nach links schieben
B364   A8         TAY          und zurückbringen
```

```
B365   B0 A4      BCS $B30B   Überlauf: 'out of memory'
B367   06 71      ASL $71     nächstes Bit aus
B369   26 72      ROL $72     $71/72 herausholen
B36B   90 0B      BCC $B378   =0? ja: Addition umgehen
B36D   18         CLC         Carry setzen (Addition)
B36E   8A         TXA         LOW-Byte holen
B36F   65 28      ADC $28     1.Wert addieren
B371   AA         TAX         LOW-Byte zurückbringen
B372   98         TYA         HIGH-Byte holen
B373   65 29      ADC $29     2.Wert addieren
B375   A8         TAY         HIGH-Byte zurückholen
B376   B0 93      BCS $B30B   Überlauf: 'out of memory'
B378   C6 5D      DEC $5D     nächstes Bit holen
B37A   D0 E3      BNE $B35F   alle 16 Bits? nein: weiter

B37C   60         RTS         Rücksprung

**************************** BASIC-Funktion  FRE
B37D   A5 0D      LDA $0D     Typflag
B37F   F0 03      BEQ $B384   kein String
B381   20 A6 B6   JSR $B6A6   FRESTR
B384   20 26 B5   JSR $B526   Garbage Collection
B387   38         SEC         Carry setzen (Subtr.)
B388   A5 33      LDA $33     Stringanfang (LOW)
B38A   E5 31      SBC $31     - Variablenende (LOW)
B38C   A8         TAY         ergibt freien Speicher
B38D   A5 34      LDA $34     Stringanfang (HIGH)
B38F   E5 32      SBC $32     - Variablenende (HIGH)

Einsprung von $AEE0, $AF6B, $B013

B391   A2 00      LDX #$00    Wert laden und
B393   86 0D      STX $0D     Flag auf numerisch setzen
B395   85 62      STA $62     LOW- und HIGH-Byte des
B397   84 63      STY $63     Ergebnisses merken
B399   A2 90      LDX #$90    und nach
B39B   4C 44 BC   JMP $BC44   Fließkomma wandlen
```

```
****************************** BASIC-Funktion  POS
B39E   38         SEC          C=1  Cursorposition holen
B39F   20 F0 FF   JSR $FFF0    Cursorposition holen

Einsprung von $B77F, $B795, $B821

B3A2   A9 00      LDA #$00     Z=1
B3A4   F0 EB      BEQ $B391    unbedingter Sprung

****************************** Test auf Direkt-Modus

Einsprung von $AB7B, $ABCE, $B3B6

B3A6   A6 3A      LDX $3A      Flag laden (Direktm. = $FF)
B3A8   E8         INX          testen
B3A9   D0 A0      BNE $B34B    nein: dann RTS
B3AB   A2 15      LDX #$15     Nummer für 'illegal direct'
B3AD   2C         .BYTE $2C
B3AE   A2 1B      LDX #$1B     Nummer für 'undef'd function'
B3B0   4C 37 A4   JMP $A437    Fehlermeldung ausgeben

****************************** BASIC-Befehl  DEF FN
B3B3   20 E1 B3   JSR $B3E1    prüft FN-Syntax
B3B6   20 A6 B3   JSR $B3A6    testet auf Direkt-Modus
B3B9   20 FA AE   JSR $AEFA    prüft auf 'Klammer auf'
B3BC   A9 80      LDA #$80     Wert laden
B3BE   85 10      STA $10      sperrt INTEGER-Variable
B3C0   20 8B B0   JSR $B08B    sucht Variable
B3C3   20 8D AD   JSR $AD8D    prüft auf numerisch
B3C6   20 F7 AE   JSR $AEF7    prüft auf 'Klammer zu'
B3C9   A9 B2      LDA #$B2     '=' BASIC-Code
B3CB   20 FF AE   JSR $AEFF    prüft auf '='
B3CE   48         PHA          erstes Zeichen auf Stapel
B3CF   A5 48      LDA $48      LOW- und HIGH-Byte der
B3D1   48         PHA          FN-Variablen-Adresse
B3D2   A5 47      LDA $47      auf den Stapel
B3D4   48         PHA          legen
B3D5   A5 7B      LDA $7B      LOW- und HIGH-Byte
B3D7   48         PHA          des Programmzeigers
B3D8   A5 7A      LDA $7A      auf den Stapel
```

```
B3DA   48         PHA          legen
B3DB   20 F8 A8   JSR $A8F8    Programmzeiger auf Statement
B3DE   4C 4F B4   JMP $B44F    FN-Variable vom Stapel holen

****************************** prüft FN-Syntax

Einsprung von $B3B3, $B3F4

B3E1   A9 A5      LDA #$A5     FN-Code
B3E3   20 FF AE   JSR $AEFF    prüft auf FN-Code
B3E6   09 80      ORA #$80     Wert laden
B3E8   85 10      STA $10      sperrt INTEGER-Variable
B3EA   20 92 B0   JSR $B092    sucht Variable
B3ED   85 4E      STA $4E      LOW- und HIGH-Byte
B3EF   84 4F      STY $4F      FN-Variablenzeiger setzen
B3F1   4C 8D AD   JMP $AD8D    prüft auf numerisch

****************************** BASIC-Funktion  FN

Einsprung von $AEE7

B3F4   20 E1 B3   JSR $B3E1    prüft FN-Syntax
B3F7   A5 4F      LDA $4F      LOW- und HIGH-Byte des
B3F9   48         PHA          FN-Variablenzeigers
B3FA   A5 4E      LDA $4E      auf den Stapel
B3FC   48         PHA          legen
B3FD   20 F1 AE   JSR $AEF1    holt Term in Klammern
B400   20 8D AD   JSR $AD8D    prüft auf numerisch
B403   68         PLA          LOW- und HIGH-Byte
B404   85 4E      STA $4E      des
B406   68         PLA          FN-Variablenzeigers wieder-
B407   85 4F      STA $4F      holen und speichern
B409   A0 02      LDY #$02     Zeiger setzen
B40B   B1 4E      LDA ($4E),Y  Zeiger (LOW) auf FN-Variable

B40D   85 47      STA $47      in Variablenadresszeiger
B40F   AA         TAX          und ins X-Reg.
B410   C8         INY          Zeiger erhöhen
B411   B1 4E      LDA ($4E),Y  Zeiger (HIGH) laden
B413   F0 99      BEQ $B3AE    gibt 'undef'd function'
```

```
B415   85 48      STA $48      in Variablenadresse
B417   C8         INY          Zeiger erhöhen
B418   B1 47      LDA ($47),Y  FN-Variablenwert holen
B41A   48         PHA          und auf Stapel retten
B41B   88         DEY          Zeiger vermindern
B41C   10 FA      BPL $B418    fertig? nein: nächster Wert
B41E   A4 48      LDY $48
B420   20 D4 BB   JSR $BBD4    FAC in FN-Variable übertragen
B423   A5 7B      LDA $7B      Programmzeiger (LOW)
B425   48         PHA          auf Stapel
B426   A5 7A      LDA $7A      Programmzeiger (HIGH)
B428   48         PHA          auf Stapel
B429   B1 4E      LDA ($4E),Y  LOW und HIGH-Byte
B42B   85 7A      STA $7A      des
B42D   C8         INY          Programmzeigers auf
B42E   B1 4E      LDA ($4E),Y  FN-Ausdruck
B430   85 7B      STA $7B      speichern
B432   A5 48      LDA $48      Zeiger auf FN-Variable
B434   48         PHA          holen und
B435   A5 47      LDA $47      auf den Stapel
B437   48         PHA          retten
B438   20 8A AD   JSR $AD8A    numerischen Ausdruck holen
B43B   68         PLA          LOW- und HIGH-Byte
B43C   85 4E      STA $4E      des Zeigers auf FN-
B43E   68         PLA          Variable vom Stapel holen
B43F   85 4F      STA $4F      und in FN-Zeiger speichern
B441   20 79 00   JSR $0079    CHRGOT letztes Zeichen holen
B444   F0 03      BEQ $B449    keine weiteren Zeichen?
B446   4C 08 AF   JMP $AF08    gibt 'SYNTAX ERROR'

B449   68         PLA          LOW- und HIGH-Byte
B44A   85 7A      STA $7A      des
B44C   68         PLA          Programmzeigers
B44D   85 7B      STA $7B      zurückholen

Einsprung von $B3DE

B44F   A0 00      LDY #$00     Zeiger setzen
B451   68         PLA          FN-Variable vom Stapel
B452   91 4E      STA ($4E),Y  zurückholen
```

```
B4BD   D0 0B      BNE $B4CA   nein: $B4CA
B4BF   98         TYA         Länge in Akku
B4C0   20 75 B4   JSR $B475   Stringzeiger berechnen
B4C3   A6 6F      LDX $6F     LOW- und HIGH-Byte der
B4C5   A4 70      LDY $70     Startadresse holen
B4C7   20 88 B6   JSR $B688   String in Bereich kopieren

****************************  Stringzeiger in
                              Descriptorstapel bringen

Einsprung von $B674, $B6FD, $B729

B4CA   A6 16      LDX $16     Stringdescriptor-Zeiger
B4CC   E0 22      CPX #$22    Stringstapel voll?
B4CE   D0 05      BNE $B4D5   nein: $B4D5
B4D0   A2 19      LDX #$19    Nr für 'formula too complex'
B4D2   4C 37 A4   JMP $A437   Fehlermeldung ausgeben
B4D5   A5 61      LDA $61     Stringlänge holen und
B4D7   95 00      STA $00,X   Stringstapel speichern
B4D9   A5 62      LDA $62     LOW- und HIGH-Byte der
B4DB   95 01      STA $01,X   Adresse holen
B4DD   A5 63      LDA $63     und in
B4DF   95 02      STA $02,X   Stringstapel bringen
B4E1   A0 00      LDY #$00    Nullwert laden
B4E3   86 64      STX $64     und Zeiger
B4E5   84 65      STY $65     jetzt auf Descriptor setzen
B4E7   84 70      STY $70     Zeiger für Polynomauswertung
B4E9   88         DEY         Register vermindern
B4EA   84 0D      STY $0D     Stringflag setzen $FF
B4EC   86 17      STX $17     Index des letzten
B4EE   E8         INX         Stringdescriptors
B4EF   E8         INX         um drei erhöhen
B4F0   E8         INX         und als
B4F1   86 16      STX $16     neuen Index merken
B4F3   60         RTS         Rücksprung
```

```
********************************* Platz für String reservieren,
                                  Länge in A

Einsprung von $B47D

B4F4   46 0F      LSR $0F     Flag für Garbage Collection
                              zurücksetzen
B4F6   48         PHA         Stringlänge
B4F7   49 FF      EOR #$FF    Alle Bits umdrehen
B4F9   38         SEC         mit HIGH-Byte des
B4FA   65 33      ADC $33     Stringanfangs-Zeigers addieren
B4FC   A4 34      LDY $34     LOW-Byte ins Y-Reg.
B4FE   B0 01      BCS $B501   Carry gesetzt ? dann weiter
B500   88         DEY         ansonsten LOW-Byte erniedrigen
B501   C4 32      CPY $32     Zu wenig Platz, dann
B503   90 11      BCC $B516   Garbage Collection durchführen
B505   D0 04      BNE $B50B   alles ok ?
B507   C5 31      CMP $31     Ende der Arrays, dann
B509   90 0B      BCC $B516   Garbage Collect durchführen
B50B   85 33      STA $33     ansonsten
B50D   84 34      STY $34     alle
B50F   85 35      STA $35     Zeiger
B511   84 36      STY $36     neu
B513   AA         TAX         setzen
B514   68         PLA         Stringlänge zurückholen
B515   60         RTS         Rücksprung

B516   A2 10      LDX #$10    Nummer für 'OUT OF MEMORY'
B518   A5 0F      LDA $0F      Flag für Garbage Collection
B51A   30 B6      BMI $B4D2   durchgeführt? 'OUT OF MEMORY'
B51C   20 26 B5   JSR $B526   Garbage Collection
B51F   A9 80      LDA #$80    Flag setzen
B521   85 0F      STA $0F     und speichern
B523   68         PLA         Stringlänge
B524   D0 D0      BNE $B4F6   String nochmals einbauen
```

```
B454   68         PLA          und abspeichern
B455   C8         INY          Zeiger erhöhen
B456   91 4E      STA ($4E),Y 2.Wert abspeichern
B458   68         PLA          3.Wert vom Stapel holen
B459   C8         INY          Zeiger erhöhen
B45A   91 4E      STA ($4E),Y und abspeichern
B45C   68         PLA          4.Wert vom Stapel holen
B45D   C8         INY          Zeiger erhöhen
B45E   91 4E      STA ($4E),Y und abspeichern
B460   68         PLA          5.Wert vom Stapel holen
B461   C8         INY          Zeiger erhöhen
B462   91 4E      STA ($4E),Y und abspeichern
B464   60         RTS          Rücksprung

****************************  BASIC-Funktion  STR$
B465   20 8D AD   JSR $AD8D    prüft auf numerisch
B468   A0 00      LDY #$00     Wert laden und
B46A   20 DF BD   JSR $BDDF    FAC nach ASCII umwandeln
B46D   68         PLA          Rücksprungadresse vom
B46E   68         PLA          Stapel entfernen

Einsprung von $AF59

B46F   A9 FF      LDA #$FF     LOW-Byte
B471   A0 00      LDY #$00     Startadresse des Strings=$FF
B473   F0 12      BEQ $B487

****************************  Stringzeiger berechnen

Einsprung von $AA56, $B4C0, $B65D

B475   A6 64      LDX $64      Zeiger in
B477   A4 65      LDY $65      $64/65 in
B479   86 50      STX $50      Zeiger auf Stringdescriptor
B47B   84 51      STY $51      speichern

Einsprung von $B6F3, $B70F

B47D   20 F4 B4   JSR $B4F4    Platz für String, Länge in A
B480   86 62      STX $62      Adresse LOW
```

```
B482   84 63      STY $63     Adresse HIGH
B484   85 61      STA $61     Länge
B486   60         RTS

**************************** String holen, Zeiger in A/Y

Einsprung von $AABF, $AB1E, $AEC6

B487   A2 22      LDX #$22    '"'-Code
B489   86 07      STX $07     nach Suchzeichen
B48B   86 08      STX $08     und Hochkommaflag

Einsprung von $AC7D

B48D   85 6F      STA $6F     Startadresse des Strings
B48F   84 70      STY $70     nach $6F/70
B491   85 62      STA $62     und $62/63
B493   84 63      STY $63     speichern
B495   A0 FF      LDY #$FF    Zeiger setzen
B497   C8         INY         Zeiger erhöhen
B498   B1 6F      LDA ($6F),Y nächstes Zeichen des Strings
B49A   F0 0C      BEQ $B4A8   Endekennzeichen?
B49C   C5 07      CMP $07     Suchzeichen?
B49E   F0 04      BEQ $B4A4   ja: $B4A4
B4A0   C5 08      CMP $08     = Zeichen in Hochkommaflag
B4A2   D0 F3      BNE $B497   nein: $B497
B4A4   C9 22      CMP #$22    '"'-Code?
B4A6   F0 01      BEQ $B4A9   ja: $B4A9
B4A8   18         CLC         Carry löschen (Addition)
B4A9   84 61      STY $61     Länge des Str. speichern und
B4AB   98         TYA         in Akku holen
B4AC   65 6F      ADC $6F     und zur Startadresse addieren
B4AE   85 71      STA $71     ergibt Endadresse LOW + 1
B4B0   A6 70      LDX $70     Übertrag
B4B2   90 01      BCC $B4B5   Addition umgehen
B4B4   E8         INX         Übertrag addieren
B4B5   86 72      STX $72     Endadresse HIGH + 1
B4B7   A5 70      LDA $70     Startadresse HIGH
B4B9   F0 04      BEQ $B4BF   null?
B4BB   C9 02      CMP #$02    zwei?
```

```
**************************** Garbage Collection

Einsprung von $A41C, $B384, $B51C

B526   A6 37      LDX $37     LOW-Byte Basic-RAM-Zeiger
B528   A5 38      LDA $38     HIGH-Byte Basic-RAM-Zeiger

Einsprung von $B63A

B52A   86 33      STX $33     in Stringzeiger
B52C   85 34      STA $34     speichern
B52E   A0 00      LDY #$00    LOW- und HIGH-Byte
B530   84 4F      STY $4F     der FN Zeiger
B532   84 4E      STY $4E     auf Null setzen
B534   A5 31      LDA $31     LOW- und HIGH-Byte der
B536   A6 32      LDX $32     Array-Zeiger laden
B538   85 5F      STA $5F     und in die Arithmetikregister
B53A   86 60      STX $60     speichern
B53C   A9 19      LDA #$19    Startadresse
B53E   A2 00      LDX #$00    der Descriptorentabelle
B540   85 22      STA $22     als Suchzeiger nach
B542   86 23      STX $23     $22 und $23 bringen
B544   C5 16      CMP $16     identisch mit String-Zeiger?
B546   F0 05      BEQ $B54D   wenn ja, dann weiter
B548   20 C7 B5   JSR $B5C7   Stringposition feststellen
B54B   F0 F7      BEQ $B544   unbedingter Sprung
B54D   A9 07      LDA #$07    Schrittweite für die Suche
B54F   85 53      STA $53     in Variablentabelle
B551   A5 2D      LDA $2D     Tabellenzeiger
B553   A6 2E      LDX $2E     laden
B555   85 22      STA $22     und als Suchzeiger nach
B557   86 23      STX $23     $22 und $23 bringen
B559   E4 30      CPX $30     Am Ende der Tabelle angelangt
B55B   D0 04      BNE $B561   wenn nicht, dann zu $B561
B55D   C5 2F      CMP $2F     ansonsten Sprung zur
B55F   F0 05      BEQ $B566   Array-Behandlung
B561   20 BD B5   JSR $B5BD   Stringposition feststellen
B564   F0 F3      BEQ $B559   unbedingter Sprung
B566   85 58      STA $58     Zeiger in die
B568   86 59      STX $59     Array-Tabelle speichern
```

```
B56A  A9 03     LDA #$03    Schrittweite für Suche
B56C  85 53     STA $53     innerhalb des Arrays festlegen
B56E  A5 58     LDA $58     Am Ende
B570  A6 59     LDX $59     der
B572  E4 32     CPX $32     Arraytabelle angelangt, dann
B574  D0 07     BNE $B57D   Sprung zu $B57D
B576  C5 31     CMP $31     Vergleich mit HIGH-Byte
B578  D0 03     BNE $B57D   Sprung zu $B57D
B57A  4C 06 B6  JMP $B606   ansonsten Transfer
B57D  85 22     STA $22     Zeiger auf Array-Header
B57F  86 23     STX $23     stellen
B581  A0 00     LDY #$00    Zähler auf Null setzen
B583  B1 22     LDA ($22),Y Variablenname erstes Zeichen
B585  AA        TAX         ins X-Reg übertragen
B586  C8        INY         Zähler erhöhen
B587  B1 22     LDA ($22),Y Variablenname zweites Zeichen
B589  08        PHP         Statusregister retten
B58A  C8        INY         Zähler erhöhen
B58B  B1 22     LDA ($22),Y Die Länge
B58D  65 58     ADC $58     des Arrays
B58F  85 58     STA $58     zu
B591  C8        INY         Zeiger
B592  B1 22     LDA ($22),Y auf
B594  65 59     ADC $59     Arraytabelle
B596  85 59     STA $59     addieren
B598  28        PLP         Statusregister wiederholen
B599  10 D3     BPL $B56E   keine Stringvariable ?
B59B  8A        TXA         dann weitersuchen
B59C  30 D0     BMI $B56E   Stringvariable, nein, weiter
B59E  C8        INY         Zähler erhöhen
B59F  B1 22     LDA ($22),Y Dimensionenanzahl holen
B5A1  A0 00     LDY #$00    Zähler wieder Null
B5A3  0A        ASL  A      mal 2
B5A4  69 05     ADC #$05    plus 5
B5A6  65 22     ADC $22     zum Zeiger addieren
B5A8  85 22     STA $22     und speichern
B5AA  90 02     BCC $B5AE   wenn ungleich, dann zu $B5AE
B5AC  E6 23     INC $23     Zeiger erhöhen
B5AE  A6 23     LDX $23     und in Array schieben
B5B0  E4 59     CPX $59     auf nächstes Feld vergleichen
```

```
B5B2   D0 04      BNE $B5B8   wenn ungleich, dann zu $B5B8
B5B4   C5 58      CMP $58     wenn gleich, dann
B5B6   F0 BA      BEQ $B572   zu $B572
B5B8   20 C7 B5   JSR $B5C7   Stringposition feststellen
B5BB   F0 F3      BEQ $B5B0   unbedingter Sprung

**************************** prüft Beseitigungsmöglichkeit

Einsprung von $B561

B5BD   B1 22      LDA ($22),Y Variablenname erstes Zeichen
B5BF   30 35      BMI $B5F6   Integer o. Funktion ?
B5C1   C8         INY         Zähler erhöhen
B5C2   B1 22      LDA ($22),Y Variablenname zweites Zeichen
B5C4   10 30      BPL $B5F6   wenn Real, dann $B5F6
B5C6   C8         INY         Zähler erhöhen

Einsprung von $B548, $B5B8

B5C7   B1 22      LDA ($22),Y holt Stringlänge
B5C9   F0 2B      BEQ $B5F6   wenn Stringlänge=0,dann $B5F6
B5CB   C8         INY         Zähler erhöhen
B5CC   B1 22      LDA ($22),Y holt Startadresse des Strings
B5CE   AA         TAX         schiebt ins X-Reg
B5CF   C8         INY         Zähler erhöhen
B5D0   B1 22      LDA ($22),Y holt Sringzeiger
B5D2   C5 34      CMP $34     Vergleich mit $34
B5D4   90 06      BCC $B5DC   wenn gleich, dann $B5DC
B5D6   D0 1E      BNE $B5F6   wenn größer, dann $B5F6
B5D8   E4 33      CPX $33     mit $33 vergleichen
B5DA   B0 1A      BCS $B5F6   wenn gleich, dann $B5F6
B5DC   C5 60      CMP $60     Vergleich mit $60
B5DE   90 16      BCC $B5F6   wenn gleich, dann $B5F6
B5E0   D0 04      BNE $B5E6   wenn größer, dann $B5E6
B5E2   E4 5F      CPX $5F     Vergleich mit $5F
B5E4   90 10      BCC $B5F6   wenn gleich, dann $B5F6
B5E6   86 5F      STX $5F     Startadresse des
B5E8   85 60      STA $60     Strings speichern
B5EA   A5 22      LDA $22     Stringdescriptor
B5EC   A6 23      LDX $23     laden
```

```
B5EE   85 4E      STA $4E      und
B5F0   86 4F      STX $4F      speichern
B5F2   A5 53      LDA $53      Tabellen Schrittweite laden
B5F4   85 55      STA $55      und speichern
B5F6   A5 53      LDA $53      und zum
B5F8   18         CLC          Suchzeiger
B5F9   65 22      ADC $22      addieren
B5FB   85 22      STA $22      und wieder
B5FD   90 02      BCC $B601    speichern
B5FF   E6 23      INC $23      Zeiger erhöhen
B601   A6 23      LDX $23      und laden
B603   A0 00      LDY #$00     Zähler löschen
B605   60         RTS          Rücksprung

**************************** Strings zusammenfügen

Einsprung von $B57A

B606   A5 4F      LDA $4F      String zwischen Tabellenende
B608   05 4E      ORA $4E      und dem oberen RAM-Bereich
B60A   F0 F5      BEQ $B601    gefunden ? nein, dann RTS
B60C   A5 55      LDA $55      Arraysuchlauf, dann $55=03
B60E   29 04      AND #$04     ansonsten $55=07
B610   4A         LSR  A       wenn Einzelvariable, dann
B611   A8         TAY          Y-Reg =2 und 0 bei Array
B612   85 55      STA $55      Wert sichern
B614   B1 4E      LDA ($4E),Y  Stringlänge holen
B616   65 5F      ADC $5F      zum LOW-Byte der Stringanfangs-
B618   85 5A      STA $5A      adresse Add., =Endadresse +1
B61A   A5 60      LDA $60      auf gleiche
B61C   69 00      ADC #$00     Weise das
B61E   85 5B      STA $5B      HIGH-Byte berechnen
B620   A5 33      LDA $33      Zielbereich
B622   A6 34      LDX $34      für den
B624   85 58      STA $58      Transfer
B626   86 59      STX $59      holen
B628   20 BF A3   JSR $A3BF    Strings verschieben
B62B   A4 55      LDY $55      LOW-Byte
B62D   C8         INY          der
B62E   A5 58      LDA $58      Anfangsadresse in
```

```
B630   91 4E      STA ($4E),Y Descriptor speichern
B632   AA         TAX         HIGH-Byte
B633   E6 59      INC $59     der Anfangsadresse

B635   A5 59      LDA $59     in
B637   C8         INY         Descriptor
B638   91 4E      STA ($4E),Y bringen
B63A   4C 2A B5   JMP $B52A   nicht alles ?, dann weiter

****************************** Stringverknüpfung '+'

Einsprung von $ADE5

B63D   A5 65      LDA $65     HIGH-Byte des Descriptors vom
B63F   48         PHA         ersten String auf Stack
B640   A5 64      LDA $64     LOW-Byte
B642   48         PHA         in Stack
B643   20 83 AE   JSR $AE83   zweiten String holen
B646   20 8F AD   JSR $AD8F   prüft auf Stringvariable
B649   68         PLA         Descriptorzeiger des ersten
B64A   85 6F      STA $6F     Strings wiederholen
B64C   68         PLA         und
B64D   85 70      STA $70     speichern
B64F   A0 00      LDY #$00    Zähler auf Null
B651   B1 6F      LDA ($6F),Y Länge des ersten Strings
B653   18         CLC         plus Länge
B654   71 64      ADC ($64),Y des zweiten Strings
B656   90 05      BCC $B65D   kleiner als 256
B658   A2 17      LDX #$17    Nummer für 'STRING TOO LONG'
B65A   4C 37 A4   JMP $A437   Fehlermeldung ausgeben
B65D   20 75 B4   JSR $B475   Platz für verknüpften String
B660   20 7A B6   JSR $B67A   ersten String übertragen
B663   A5 50      LDA $50     Zeiger auf
B665   A4 51      LDY $51     zweiten Stringdescriptor
B667   20 AA B6   JSR $B6AA   FRESTR
B66A   20 8C B6   JSR $B68C   2. String an 1. anhängen
B66D   A5 6F      LDA $6F     Descriptorzeiger des
B66F   A4 70      LDY $70     zweiten Strings
B671   20 AA B6   JSR $B6AA   FRESTR
```

```
B674   20 CA B4   JSR $B4CA   Descriptor in Stringstack
B677   4C B8 AD   JMP $ADB8   zurück zur Formelauswertung

****************************** String in reserv. Bereich

Einsprung von $AA61, $B660

B67A   A0 00      LDY #$00    Zähler auf Null
B67C   B1 6F      LDA ($6F),Y Stringlänge holen
B67E   48         PHA         und merken
B67F   C8         INY         Zähler erhöhen
B680   B1 6F      LDA ($6F),Y LOW-Byte der Stringadresse
B682   AA         TAX         ins X-Reg
B683   C8         INY         Zähler erhöhen
B684   B1 6F      LDA ($6F),Y HIGH-Byte der Stringadresse
B686   A8         TAY         ins Y-Reg und
B687   68         PLA         Stack

Einsprung von $B4C7

B688   86 22      STX $22     Zeiger auf
B68A   84 23      STY $23     String speichern

Einsprung von $B66A, $B726

B68C   A8         TAY         Länge null ?
B68D   F0 0A      BEQ $B699   dann fertig
B68F   48         PHA         wieder in Stack
B690   88         DEY         Zähler erniedrigen
B691   B1 22      LDA ($22),Y String
B693   91 35      STA ($35),Y in den
B695   98         TYA         Stringbereich
B696   D0 F8      BNE $B690   übertragen
B698   68         PLA         Den
B699   18         CLC         Zeiger
B69A   65 35      ADC $35     um
B69C   85 35      STA $35     die
B69E   90 02      BCC $B6A2   Stringlänge
B6A0   E6 36      INC $36     erhöhen
B6A2   60         RTS         Rücksprung
```

```
****************************** Stringverwaltung  FRESTR

Einsprung von $B782, $E25A

B6A3   20 8F AD   JSR $AD8F   prüft auf Stringvariable

Einsprung von $A9E0, $AB21, $B034, $B381

B6A6   A5 64      LDA $64     Zeiger auf
B6A8   A4 65      LDY $65     Stringdescriptor

Einsprung von $B041, $B667, $B671, $B716

B6AA   85 22      STA $22     nach
B6AC   84 23      STY $23     $22 und $23 bringen
B6AE   20 DB B6   JSR $B6DB   Descriptor vom Stringstack
B6B1   08         PHP         Statusregister retten
B6B2   A0 00      LDY #$00    Zähler auf Null
B6B4   B1 22      LDA ($22),Y Stringlänge holen
B6B6   48         PHA         und in Stack schieben
B6B7   C8         INY         Zähler erhöhen
B6B8   B1 22      LDA ($22),Y LOW-Byte der Anfangsadresse
B6BA   AA         TAX         ins X-Reg schieben
B6BB   C8         INY         Zähler erhöhen
B6BC   B1 22      LDA ($22),Y HIGH-Byte der Anfangsadresse
B6BE   A8         TAY         ins Y-Reg schieben
B6BF   68         PLA         Stringlänge wieder aus Stack
B6C0   28         PLP         Statusreg. wieder aus Stack
B6C1   D0 13      BNE $B6D6   Neustring=Altstring nein? RTS
B6C3   C4 34      CPY $34     Stringadresse identisch mit
B6C5   D0 0F      BNE $B6D6   Zeiger auf Stringende?
B6C7   E4 33      CPX $33     nein, dann
B6C9   D0 0B      BNE $B6D6   zu $B6D6
B6CB   48         PHA         String-Anfangszeiger
B6CC   18         CLC         auf Länge
B6CD   65 33      ADC $33     des
B6CF   85 33      STA $33     Strings
B6D1   90 02      BCC $B6D5   hinaufsetzen
B6D3   E6 34      INC $34     Stringlänge
```

```
B6D5   68        PLA          holen
B6D6   86 22     STX $22      LOW-Byte der Startadresse
B6D8   84 23     STY $23      HIGH-Byte der Startadresse
B6DA   60        RTS          Rücksprung

****************************** Stringzeiger aus
                                Descriptorstack entfernen

Einsprung von $AA6C, $B6AE

B6DB   C4 18     CPY $18      Zeiger auf Stringdescriptor
B6DD   D0 0C     BNE $B6EB    identisch mit $18, nicht? RTS
B6DF   C5 17     CMP $17      identisch mit 17
B6E1   D0 08     BNE $B6EB    wenn nicht, dann RTS
B6E3   85 16     STA $16      Zeiger nach $16 speichern
B6E5   E9 03     SBC #$03     Von Adresse $17
B6E7   85 17     STA $17      3 abziehen
B6E9   A0 00     LDY #$00     Zähler auf Null
B6EB   60        RTS          Rücksprung

****************************** BASIC-Funktion CHR$
B6EC   20 A1 B7  JSR $B7A1    holt Byte-Wert  (0 bis 255)
B6EF   8A        TXA          Kode in Akku
B6F0   48        PHA          Akkuinhalt in Stack
B6F1   A9 01     LDA #$01     Länge des Strings gleich 1
B6F3   20 7D B4  JSR $B47D    Platz für String freimachen
B6F6   68        PLA          ASCII-Kode zurückholen
B6F7   A0 00     LDY #$00     Zähler auf Null
B6F9   91 62     STA ($62),Y  als Stringzeichen speichern
B6FB   68        PLA          Rücksprungadresse aus
B6FC   68        PLA          Stack entfernen
B6FD   4C CA B4  JMP $B4CA    Descriptor in Stringstack

****************************** BASIC-Funktion  LEFT$
B700   20 61 B7  JSR $B761    Stringadresse & Länge
                              aus Stack holen
B703   D1 50     CMP ($50),Y  Länge mit LEFT$-Parameter
                              vergleichen
B705   98        TYA          LEFT$-Parameter
```

```
Einsprung von $B734

B706   90 04      BCC $B70C   kleiner als Stringlänge ?
B708   B1 50      LDA ($50),Y Stringlänge holen
B70A   AA         TAX         und ins X-Reg schieben
B70B   98         TYA         Stringlänge und
B70C   48         PHA         Parameter für LEFT$
B70D   8A         TXA         in Stack
B70E   48         PHA         schieben
B70F   20 7D B4   JSR $B47D   Platz für neuen String
                              reservieren
B712   A5 50      LDA $50     Zeiger auf Stringdescriptor
B714   A4 51      LDY $51     laden
B716   20 AA B6   JSR $B6AA   FRESTR
B719   68         PLA         Länge des neuen Strings aus
B71A   A8         TAY         Stack holen und ins X-Reg
B71B   68         PLA         alte
B71C   18         CLC         Stringadresse
B71D   65 22      ADC $22     entsprechend
B71F   85 22      STA $22     erhöhen
B721   90 02      BCC $B725   und speichern
B723   E6 23      INC $23     HIGH-Byte erhöhen
B725   98         TYA         neue Stringlänge holen
B726   20 8C B6   JSR $B68C   neuen String in
                              Stringbereich übertragen
B729   4C CA B4   JMP $B4CA   Descriptor in Stringstack
                              bringen

****************************** BASIC-Funktion RIGHT$
B72C   20 61 B7   JSR $B761   Stringparameter und Länge
                              vom Stack holen
B72F   18         CLC         von Stringlänge
B730   F1 50      SBC ($50),Y abziehen
B732   49 FF      EOR #$FF    Nummer des ersten Elements
                              im alten String
B734   4C 06 B7   JMP $B706   weiter wie LEFT$

****************************** BASIC-Funktion  MID$
B737   A9 FF      LDA #$FF    Ersatzwert für den zweiten
B739   85 65      STA $65     Zahlenparameter
```

```
B73B  20 79 00   JSR $0079    CHRGOT  letzten Zeichen holen
B73E  C9 29      CMP #$29     ')' Klammer zu
B740  F0 06      BEQ $B748    wenn ja, dann kein zweiter
                              Parameter, weiter bei $B748
B742  20 FD AE   JSR $AEFD    prüft auf Komma
B745  20 9E B7   JSR $B79E    holt Byte-Wert des zweiten
                              Parameters
B748  20 61 B7   JSR $B761    Stringparameter und
                              Startposition holen

B74B  F0 4B      BEQ $B798    1. Parameter null, 'ILLEGAL
                              QUANTITY'
B74D  CA         DEX          erste Elementposition
B74E  8A         TXA          innerhalb
B74F  48         PHA          des alten Strings
B750  18         CLC          im Stack ablegen
B751  A2 00      LDX #$00     Zähler setzen
B753  F1 50      SBC ($50),Y  alte Stringlänge kleiner als
                              erster Parameter ?
B755  B0 B6      BCS $B70D    wenn ja, dann zu LEFT$
B757  49 FF      EOR #$FF     Berechnen der neuen Länge
B759  C5 65      CMP $65      wenn kleiner als zweiter
B75B  90 B1      BCC $B70E    Parameter, dann zu LEFT $
B75D  A5 65      LDA $65      Zweitparameter als 'rechte'
                              Stringbegrenzung
B75F  B0 AD      BCS $B70E    unbedingter Sprung

****************************** Stringparameter numerischer Wert
                              vom Stack holen

Einsprung von $B700, $B72C, $B748

B761  20 F7 AE   JSR $AEF7    prüft auf Klammer zu
B764  68         PLA          LOW-Byte der
B765  A8         TAY          Aufrufadresse merken
B766  68         PLA          HIGH-Byte der
B767  85 55      STA $55      Aufrufadresse merken
B769  68         PLA          LOW-und HIGH-Byte der
B76A  68         PLA          Aufrufadresse merken
B76B  68         PLA          1. Parameter holen
```

```
B76C   AA        TAX         und ins X-Reg
B76D   68        PLA         LOW- und HIGH-Byte
B76E   85 50     STA $50     des
B770   68        PLA         Stringdescriptors
B771   85 51     STA $51     nach
B773   A5 55     LDA $55     $51 und $52 speichern
B775   48        PHA         Aufrufadresse
B776   98        TYA         wieder auf
B777   48        PHA         Stack
B778   A0 00     LDY #$00    Zähler auf Null
B77A   8A        TXA         Länge, zweiter Parameter
B77B   60        RTS         Rücksprung

****************************** BASIC-Funktion  LEN
B77C   20 82 B7  JSR $B782   FRESTR, Stringlänge holen
B77F   4C A2 B3  JMP $B3A2   Byte-Wert nach
                             Fließkommaformat wandeln

****************************** Stringparameter holen

Einsprung von $B77C, $B78B, $B7AD

B782   20 A3 B6  JSR $B6A3   FRESTR, String holen, Länge
                             in A
B785   A2 00     LDX #$00    Typeflag
B787   86 0D     STX $0D     auf numerisch setzen
B789   A8        TAY         Länge in Y
B78A   60        RTS         Rücksprung

****************************** BASIC-Funktion  ASC
B78B   20 82 B7  JSR $B782   String holen, Zeiger in
                             $22/$23, Länge in Y
B78E   F0 08     BEQ $B798   Länge gleich null,
                             'ILLEGAL QUANTITY'
B790   A0 00     LDY #$00    Zähler auf Null
B792   B1 22     LDA ($22),Y erstes Zeichen holen
B794   A8        TAY         ASCII-Kode
B795   4C A2 B3  JMP $B3A2   nach Fließkomma wandeln
B798   4C 48 B2  JMP $B248   'ILLEGAL QUANTITY'
```

```
****************************** holt Byte-Wert nach X

Einsprung von $AAFF

B79B   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen

Einsprung von $A94V, $AA86, $AB85, $ABA5, $AFC7, $B745,
              $B7F4, $E203, $E221

B79E   20 8A AD   JSR $AD8A   FRMNUM numerischen Wert
                              nach FAC holen

Einsprung von $B6EC

B7A1   20 B8 B1   JSR $B1B8   prüft auf Bereich und
                              wandelt nach Integer
B7A4   A6 64      LDX $64     HIGH-Byte
B7A6   D0 F0      BNE $B798   ungleich null, dann
                              'ILLEGAL QUANTITY'
B7A8   A6 65      LDX $65     LOW-Byte des geholten
                              Ausdrucks ins X-Reg
B7AA   4C 79 00   JMP $0079   CHRGOT letztes Zeichen holen

****************************** BASIC-Funktion  VAL
B7AD   20 82 B7   JSR $B782   Stringadresse und Länge holen
B7B0   D0 03      BNE $B7B5   Stringlänge ungleich Null ?
B7B2   4C F7 B8   JMP $B8F7   Null in FAC
B7B5   A6 7A      LDX $7A     Programmzeiger
B7B7   A4 7B      LDY $7B     holen
B7B9   86 71      STX $71     und
B7BB   84 72      STY $72     speichern
B7BD   A6 22      LDX $22     Stringanfangsadresse
B7BF   86 7A      STX $7A     in Stringzeiger bringen
B7C1   18         CLC         LOW-Byte des
B7C2   65 22      ADC $22     ersten Zeichens
B7C4   85 24      STA $24     nach dem String speichern
B7C6   A6 23      LDX $23     HIGH-Byte
B7C8   86 7B      STX $7B     des ersten
B7CA   90 01      BCC $B7CD   Zeichens
B7CC   E8         INX         nach dem String
```

```
B7CD   86 25      STX $25      speichern
B7CF   A0 00      LDY #$00     Zähler auf Null
B7D1   B1 24      LDA ($24),Y  erstes Byte nach String
B7D3   48         PHA          auf Stack
B7D4   98         TYA          speichern
B7D5   91 24      STA ($24),Y  und durch null ersetzen
B7D7   20 79 00   JSR $0079    CHRGOT letztes Zeichen holen
B7DA   20 F3 BC   JSR $BCF3    String in Fließkommazahl
                               umwandeln
B7DD   68         PLA          Zeichen nach String
B7DE   A0 00      LDY #$00     Zähler auf Null
B7E0   91 24      STA ($24),Y  wieder zurücksetzen

Einsprung von $AC80, $AEC9

B7E2   A6 71      LDX $71      Die
B7E4   A4 72      LDY $72      Programmzeiger
B7E6   86 7A      STX $7A      wieder
B7E8   84 7B      STY $7B      zurückholen
B7EA   60         RTS          Rücksprung

****************************** GETADR und GETBYT holt
                               16-Bit und 8-Bit-Wert

Einsprung von $B824, $B82D

B7EB   20 8A AD   JSR $AD8A    FRMNUM holt numerischen Wert
B7EE   20 F7 B7   JSR $B7F7    FAC in Adressformat wandlen
                               $14/$15

Einsprung von $B839

B7F1   20 FD AE   JSR $AEFD    CHKCOM prüft auf Komma
B7F4   4C 9E B7   JMP $B79E    holt Byte-Wert nach X
```

```
********************************* GETADR  FAC in positive
                                   16-Bit-Zahl wandeln

Einsprung von $B7EE, $B813, $E12D

B7F7   A5 66      LDA $66      Vorzeichen
B7F9   30 9D      BMI $B798    negativ, dann
                               'ILLEGAL QUANTITY'
B7FB   A5 61      LDA $61      Exponent
B7FD   C9 91      CMP #$91     Zahl mit 65536 vergleichen
B7FF   B0 97      BCS $B798    größer, dann
                               'ILLEGAL QUANTITY'
B801   20 9B BC   JSR $BC9B    FAC in Adressformat wandeln
B804   A5 64      LDA $64      Wert
B806   A4 65      LDY $65      holen
B808   84 14      STY $14      und nach $14/$15
B80A   85 15      STA $15      speichern
B80C   60         RTS          Rücksprung

********************************* BASIC-Funktion  PEEK
B80D   A5 15      LDA $15      $15 und $16
B80F   48         PHA          in
B810   A5 14      LDA $14      Stack
B812   48         PHA          sichern
B813   20 F7 B7   JSR $B7F7    FAC nach Adressformat
                               wandeln
B816   A0 00      LDY #$00     Zähler auf Null
B818   B1 14      LDA ($14),Y  Peek-Wert holen
B81A   A8         TAY          nach Y-Reg
B81B   68         PLA          $15 und $16
B81C   85 14      STA $14      wieder
B81E   68         PLA          vom Stack
B81F   85 15      STA $15      zurückholen
B821   4C A2 B3   JMP $B3A2    Y nach Fließkommaformat

********************************* BASIC-Befehl  POKE
B824   20 EB B7   JSR $B7EB    Poke-Adresse und Wert holen
B827   8A         TXA          Poke-Wert in Akku
B828   A0 00      LDY #$00     Zähler auf Null
```

```
B82A   91 14      STA ($14),Y und in Speicher schreiben
B82C   60         RTS         Rücksprung

****************************** BASIC-Befehl  WAIT
B82D   20 EB B7   JSR $B7EB   Adresse und Wert holen
B830   86 49      STX $49     zweiter Parameter nach $49
B832   A2 00      LDX #$00    Default für dritten Parameter
B834   20 79 00   JSR $0079   CHRGOT letztes Zeichen
B837   F0 03      BEQ $B83C   kein dritter Parameter ?
B839   20 F1 B7   JSR $B7F1   prüft auf Komma und holt
                              Parameter
B83C   86 4A      STX $4A     dritter Parameter nach $4A
B83E   A0 00      LDY #$00    Zähler auf Null
B840   B1 14      LDA ($14),Y Wait-Adresse
B842   45 4A      EOR $4A     logisch
B844   25 49      AND $49     verknüpfen
B846   F0 F8      BEQ $B840   weiter warten
B848   60         RTS         Rücksprung

****************************** Arithmetik-Routinen

****************************** FAC = FAC + 0.5

Einsprung von $BE2F, $E290

B849   A9 11      LDA #$11    Zeiger auf
B84B   A0 BF      LDY #$BF    Konstante 0.5
B84D   4C 67 B8   JMP $B867   FAC = FAC + Konstante (A/Y)

****************************** Minus  FAC = Konstante
                              (A/Y) - FAC
B850   20 8C BA   JSR $BA8C   Konstante (A/Y) nach ARG

****************************** Minus  FAC = ARG - FAC
B853   A5 66      LDA $66     Die
B855   49 FF      EOR #$FF    Vorzeichen
B857   85 66      STA $66     umdrehen
B859   45 6E      EOR $6E     mit Vorzeichen von FAC
B85B   85 6F      STA $6F     verknüpfen
```

```
B85D   A5 61      LDA $61     Exponent von FAC
B85F   4C 6A B8   JMP $B86A   FAC = FAC + ARG

********************************
B862   20 99 B9   JSR $B999   Exponenten von FAC und ARG
B865   90 3C      BCC $B8A3   angleichen

******************************** Plus  FAC = Konstante (A/Y) +
                                FAC

Einsprung von $AD$F, $B84D, $BA01, $BA1D ,$E081 , $E0D0,
              $E268, $E2A4

B867   20 8C BA   JSR $BA8C   Konstante (A/Y) nach ARG

******************************** Plus  FAC = FAC + ARG

Einsprung von $B85F, $BD8E

B86A   D0 03      BNE $B86F   FAC ungleich null ?
B86C   4C FC BB   JMP $BBFC   nein, dann FAC = ARG
B86F   A6 70      LDX $70     Rundungsbyte für FAC
B871   86 56      STX $56     in $56 speichern
B873   A2 69      LDX #$69    Offset-Zeiger für ARG laden
B875   A5 69      LDA $69     Exponent von ARG laden

Einsprung von $BAF1

B877   A8         TAY         in Y-Reg schieben
B878   F0 CE      BEQ $B848   wenn ARG=0, dann RTS
B87A   38         SEC         Exponent von
B87B   E5 61      SBC $61     FAC subtrahieren
B87D   F0 24      BEQ $B8A3   wenn Exponent gleich, dann zu
                              $B8A3
B87F   90 12      BCC $B893   wenn Exponent von FAC größer,
                              dann zu $B893
B881   84 61      STY $61     FAC-Exponent durch
                              ARG-Vorzeichen ersetzen
B883   A4 6E      LDY $6E     FAC-Vorzeichen durch
B885   84 66      STY $66     ARG-Vorzeichen ersetzen
```

```
B887  49 FF     EOR #$FF     Vorzeichen wechseln
B889  69 00     ADC #$00     Carry ist schon 1
B88B  A0 00     LDY #$00     Rundungsstelle
B88D  84 56     STY $56      löschen
B88F  A2 61     LDX #$61     Offset-Zeiger für FAC laden
B891  D0 04     BNE $B897    unbedingter Sprung
B893  A0 00     LDY #$00     FAC-Rundungsstelle
B895  84 70     STY $70      löschen
B897  C9 F9     CMP #$F9     wenn Exponentdifferenz
B899  30 C7     BMI $B862    größer als 7, dann zu $B862
B89B  A8        TAY          Akku löschen
B89C  A5 70     LDA $70      FAC-Rundungsstelle
B89E  56 01     LSR $01,X    laden
B8A0  20 B0 B9  JSR $B9B0    Mantisse verschieben
B8A3  24 6F     BIT $6F      wenn FAC- und ARG-Vorzeichen
B8A5  10 57     BPL $B8FE    identisch, dann zu $B8FE
B8A7  A0 61     LDY #$61     Offset-Zeiger für FAC laden
B8A9  E0 69     CPX #$69     wenn Offset-Zeiger für ARG
B8AB  F0 02     BEQ $B8AF    initialisiert, dann zu $B8AF
B8AD  A0 69     LDY #$69     Offset-Zeiger laden
B8AF  38        SEC          Carryflag für Subtraktion
                             setzen
B8B0  49 FF     EOR #$FF     Alle Bits umdrehen
B8B2  65 56     ADC $56      Rundungsstelle addieren
B8B4  85 70     STA $70      und speichern
B8B6  B9 04 00  LDA $0004,Y  viertes Byte
B8B9  F5 04     SBC $04,X    subtrahieren und in
B8BB  85 65     STA $65      FAC speichern
B8BD  B9 03 00  LDA $0003,Y  drittes Byte
B8C0  F5 03     SBC $03,X    subtrahieren und in
B8C2  85 64     STA $64      FAC speichern
B8C4  B9 02 00  LDA $0002,Y  zweites Byte
B8C7  F5 02     SBC $02,X    subtrahieren und in
B8C9  85 63     STA $63      FAC speichern
B8CB  B9 01 00  LDA $0001,Y  erstes Byte
B8CE  F5 01     SBC $01,X    subtrahieren und in
B8D0  85 62     STA $62      FAC speichern
```

```
Einsprung von $BC55, $BCE6

B8D2   B0 03      BCS $B8D7   wenn Übertrag negativ, dann
                              weiter
B8D4   20 47 B9   JSR $B947   Mantisse von FAC invertieren

Einsprung von $BB9F, $E0EF

B8D7   A0 00      LDY #$00    Y-Reg und
B8D9   98         TYA         Akku löschen
B8DA   18         CLC         Carry löschen
B8DB   A6 62      LDX $62     wenn $62=0 dann,
B8DD   D0 4A      BNE $B929   zu $B929
B8DF   A6 63      LDX $63     Das
B8E1   86 62      STX $62     gesamte
B8E3   A6 64      LDX $64     FAC
B8E5   86 63      STX $63     wieder
B8E7   A6 65      LDX $65     norma-
B8E9   86 64      STX $64     lisieren
B8EB   A6 70      LDX $70     Rundungsstelle
B8ED   86 65      STX $65     wieder
B8EF   84 70      STY $70     löschen
B8F1   69 08      ADC #$08    Zähler um 8 Bits verschieben
B8F3   C9 20      CMP #$20    wenn 32 Bits verschoben,
B8F5   D0 E4      BNE $B8DB   dann weiter

Einsprung von $B7B2, $BADC

B8F7   A9 00      LDA #$00    Mantisse =0

Einsprung von $BF81

B8F9   85 61      STA $61     FAC =0

Einsprung von $BACC

B8FB   85 66      STA $66     Exponent =0
B8FD   60         RTS         Rücksprung
```

```
B8FE  65 56      ADC $56     Rundungsstelle addieren
B900  85 70      STA $70     und speichern
B902  A5 65      LDA $65     FAC
B904  65 6D      ADC $6D     und ARG
B906  85 65      STA $65     addieren
B908  A5 64      LDA $64     FAC
B90A  65 6C      ADC $6C     und ARG
B90C  85 64      STA $64     addieren
B90E  A5 63      LDA $63     FAC
B910  65 6B      ADC $6B     und ARG
B912  85 63      STA $63     addieren
B914  A5 62      LDA $62     FAC
B916  65 6A      ADC $6A     und ARG
B918  85 62      STA $62     addieren
B91A  4C 36 B9   JMP $B936   Überlaufbit in Mantisse
                             zurückshiften
B91D  69 01      ADC #$01    Zähler erhöhen
B91F  06 70      ASL $70     FAC solange
B921  26 65      ROL $65     nach links
B923  26 64      ROL $64     verschieben bis das
B925  26 63      ROL $63     Bit 7
B927  26 62      ROL $62     gesetzt ist
B929  10 F2      BPL $B91D   nicht gesetzt ? dann nochmal
B92B  38         SEC         wenn Binärexponent kleiner
B92C  E5 61      SBC $61     als die Anzahl der
B92E  B0 C7      BCS $B8F7   Verschiebungen, dann wird die
                             Zahl als Null behandelt
B930  49 FF      EOR #$FF    Exponent um
B932  69 01      ADC #$01    Verschiebungsanzahl
B934  85 61      STA $61     vermindern

Einsprung von $B91A

B936  90 0E      BCC $B946   Carry gesetzt, nein dann RTS

Einsprung von $BC28

B938  E6 61      INC $61     Exponent erhöhen
B93A  F0 42      BEQ $B97E   wenn überlauf in Exponent,
                             dann 'OVERFLOW ERROR'
```

```
B93C   66 62      ROR $62     Überlaufbit in Carry schieben
B93E   66 63      ROR $63     Das Carry-Flag
B940   66 64      ROR $64     erhält die
B942   66 65      ROR $65     Position des
B944   66 70      ROR $70     höchstwertigen Bits
B946   60         RTS         Rücksprung

****************************** Mantisse von FAC invertieren

Einsprung von $B8D4

B947   A5 66      LDA $66     FAC Vorzeichen
B949   49 FF      EOR #$FF    invertieren
B94B   85 66      STA $66     und speichern

Einsprung von $BCAB

B94D   A5 62      LDA $62     FAC
B94F   49 FF      EOR #$FF    invertieren
B951   85 62      STA $62     und speichern
B953   A5 63      LDA $63     FAC
B955   49 FF      EOR #$FF    invertieren
B957   85 63      STA $63     und speichern
B959   A5 64      LDA $64     FAC
B95B   49 FF      EOR #$FF    invertieren
B95D   85 64      STA $64     und speichern
B95F   A5 65      LDA $65     FAC
B961   49 FF      EOR #$FF    invertieren
B963   85 65      STA $65     und speichern
B965   A5 70      LDA $70     FAC-Rundungsbyte
B967   49 FF      EOR #$FF    invertieren
B969   85 70      STA $70     und speichern
B96B   E6 70      INC $70     Mantisse erhöhen
B96D   D0 0E      BNE $B97D   nicht Null? dann RTS

Einsprung von $BC23

B96F   E6 65      INC $65     FAC erhöhen
B971   D0 0A      BNE $B97D   nicht Null? dann RTS
B973   E6 64      INC $64     FAC erhöhen
```

```
B975   D0 06      BNE $B97D   nicht Null? dann RTS
B977   E6 63      INC $63     FAC erhöhen
B979   D0 02      BNE $B97D   nicht Null? dann RTS
B97B   E6 62      INC $62     FAC erhöhen
B97D   60         RTS         Rücksprung

Einsprung von $BADF, $BD9D

B97E   A2 0F      LDX #$0F    Nummer für 'OVERFLOW'
B980   4C 37 A4   JMP $A437   Fehlermeldung ausgeben

****************************** Rechtsverschieben eines
                               Registers

Einsprung von $BA5B

B983   A2 25      LDX #$25    Offset-Zeiger auf Register
B985   B4 04      LDY $04,X   FAC-
B987   84 70      STY $70     Rundungsbyte
B989   B4 03      LDY $03,X   1 mal
B98B   94 04      STY $04,X   verschieben
B98D   B4 02      LDY $02,X   2 mal
B98F   94 03      STY $03,X   verschieben
B991   B4 01      LDY $01,X   3 mal
B993   94 02      STY $02,X   verschieben
B995   A4 68      LDY $68     FAC-
B997   94 01      STY $01,X   Rundungsbyte

Einsprung von $B862, $BCB5

B999   69 08      ADC #$08    Zähler um 8 erhöhen
B99B   30 E8      BMI $B985   größer als 0?
B99D   F0 E6      BEQ $B985   wenn nicht, dann weiter
                              verschieben
B99F   E9 08      SBC #$08    Zähler um 8 vermindern
B9A1   A8         TAY         Zähler sichern
B9A2   A5 70      LDA $70     FAC-Rundungsbyte laden
B9A4   B0 14      BCS $B9BA   wenn Null, dann CLC, RTS
B9A6   16 01      ASL $01,X   höchstwertiges Bit =1?,
B9A8   90 02      BCC $B9AC   wenn nicht, dann zu $B9AC
```

```
B9AA   F6 01      INC $01,X   höchste Mantissenstelle
                              erhöhen
B9AC   76 01      ROR $01,X   sämtliche
B9AE   76 01      ROR $01,X   Stellen

Einsprung von $B8A0, $BCC6

B9B0   76 02      ROR $02,X   um ein
B9B2   76 03      ROR $03,X   Bit nach
B9B4   76 04      ROR $04,X   rechts
B9B6   6A         ROR  A      verschieben
B9B7   C8         INY         Zähler um eins erhöhen
B9B8   D0 EC      BNE $B9A6   verschieben bis Zähler =0
B9BA   18         CLC         Carry löschen
B9BB   60         RTS         Rücksprung

****************************  Konstanten für  LOG
B9BC   81 00 00 00 00         1
B9C1   03                     3 = Polynomgrad, dann 4
                              Koeffizienten
B9C2   7F 5E 56 CB 79          .434255942
B9C7   80 13 9B 0B 64          .576584541
B9CC   80 76 38 93 16          .961800759
B9D1   82 38 AA 3B 20         2.88539007
B9D5   80 35 04 F3 34          .707106781 = 1/SQR(2)
B9DB   81 35 04 F3 34         1.41421356 = SQR(2)
B9E0   80 80 00 00 00         -.5
B9E5   80 31 72 17 F8          .693147181  =  LOG(2)

****************************  BASIC-Funktion LOG

Einsprung von $BFA3

B9EA   20 2B BC   JSR $BC2B   Vorzeichen holen
B9ED   F0 02      BEQ $B9F1   null ?, dann fertig
B9EF   10 03      BPL $B9F4   positiv ?, dann ok
B9F1   4C 48 B2   JMP $B248   'ILLEGAL QUANTITY'
B9F4   A5 61      LDA $61     Exponent
B9F6   E9 7F      SBC #$7F    normalisieren
B9F8   48         PHA         und merken
```

```
B9F9   A9 80      LDA #$80    Zahl in Bereich 0.5 bis 1
B9FB   85 61      STA $61     bringen
B9FD   A9 D6      LDA $D6     Zeiger auf
B9FF   A0 B9      LDY $B9     Konstante 1/SQR(2)
BA01   20 67 B8   JSR $B867   zu FAC addieren
BA04   A9 DB      LDA #$DB    Zeiger auf
BA06   A0 B9      LDY #$B9    Konstante SQR(2)
BA08   20 0F BB   JSR $BB0F   SQR(2) durch FAC dividieren
BA0B   A9 BC      LDA #$BC    Zeiger
BA0D   A0 B9      LDY #$B9    auf Konstante 1
BA0F   20 50 B8   JSR $B850   1 minus FAC
BA12   A9 C1      LDA #$C1    Zeiger auf
BA14   A0 B9      LDY #$B9    Polynomkoeffizienten
BA16   20 43 E0   JSR $E043   Polynomberechnung
BA19   A9 E0      LDA #$E0    Zeiger auf
BA1B   A0 B9      LDY #$B9    Konstante -0.5
BA1D   20 67 B8   JSR $B867   zu FAC addieren
BA20   68         PLA         Exponent zurückholen
BA21   20 7E BD   JSR $BD7E   FAC = FAC + FAC
BA24   A9 E5      LDA #$E5    Zeiger auf
BA26   A0 B9      LDY #$B9    Konstante LOG(2)

****************************** Multiplikation  FAC =
                               Konstante (A/Y) * FAC

Einsprung von $BE04, $BFAA, $BFF1, $E04C, $E056, $E070,
             $E0C9

BA28   20 8C BA   JSR $BA8C   Konstante nach ARG

****************************** Multiplikation  FAC = ARG *
                               FAC
BA2B   D0 03      BNE $BA30   nicht null ?
BA2D   4C 8B BA   JMP $BA8B   RTS
BA30   20 B7 BA   JSR $BAB7   Exponent berechnen
BA33   A9 00      LDA #$00    Alle
BA35   85 26      STA $26     Funktions-
BA37   85 27      STA $27     register
BA39   85 28      STA $28     lö-
BA3B   85 29      STA $29     schen
```

```
BA3D   A5 70      LDA $70     bitweise
BA3F   20 59 BA   JSR $BA59   Multiplikation
BA42   A5 65      LDA $65     bitweise
BA44   20 59 BA   JSR $BA59   Multiplikation
BA47   A5 64      LDA $64     bitweise
BA49   20 59 BA   JSR $BA59   Multiplikation
BA4C   A5 63      LDA $63     bitweise
BA4E   20 59 BA   JSR $BA59   Multiplikation
BA51   A5 62      LDA $62     bitweise
BA53   20 5E BA   JSR $BA5E   Multiplikation
BA56   4C 8F BB   JMP $BB8F   Register nach FAC,
                              linksbündig machen

****************************** bitweise Multiplikation

Einsprung von $BA3F, $BA44, $BA49, $BA4E

BA59   D0 03      BNE $BA5E   Rechtsverschieben
BA5B   4C 83 B9   JMP $B983   des Registers

Einsprung von $BA53

BA5E   4A         LSR  A      binäre Multiplikation
BA5F   09 80      ORA #$80    des Akkus
BA61   A8         TAY         mit ARG.
BA62   90 19      BCC $BA7D   Das Ergebnis kommt
BA64   18         CLC         in das
BA65   A5 29      LDA $29     Register für
BA67   65 6D      ADC $6D     Funktionen.
BA69   85 29      STA $29     Bei gesetztem Bit
BA6B   A5 28      LDA $28     im Akku
BA6D   65 6C      ADC $6C     wird ARG
BA6F   85 28      STA $28     zum
BA71   A5 27      LDA $27     Funktionsregister
BA73   65 6B      ADC $6B     addiert.
BA75   85 27      STA $27     Zusätzlich
BA77   A5 26      LDA $26     werden
BA79   65 6A      ADC $6A     die
BA7B   85 26      STA $26     Funktionsregister
BA7D   66 26      ROR $26     noch
```

```
BA7F   66 27      ROR $27     verdoppelt.
BA81   66 28      ROR $28     Die Routine
BA83   66 29      ROR $29     arbeitet
BA85   66 70      ROR $70     im selben
BA87   98         TYA         Prinzip
BA88   4A         LSR  A      wie
BA89   D0 D6      BNE $BA61   bei $B34C.
```

Einsprung von $BA2D

```
BA8B   60         RTS         Rücksprung
```

****************************** ARG = Konstante (A/Y)

Einsprung von $B850, $B867, $BA28, $BB0F

```
BA8C   85 22      STA $22     Die
BA8E   84 23      STY $23     Konstante,
BA90   A0 04      LDY #$04    auf
BA92   B1 22      LDA ($22),Y die
BA94   85 6D      STA $6D     das
BA96   88         DEY         Akku
BA97   B1 22      LDA ($22),Y und
BA99   85 6C      STA $6C     das
BA9B   88         DEY         Y-Reg
BA9C   B1 22      LDA ($22),Y zeigt, nach ARG.
BA9E   85 6B      STA $6B     Die
BAA0   88         DEY         gesamten
BAA1   B1 22      LDA ($22),Y Vor-
BAA3   85 6E      STA $6E     zei-
BAA5   45 66      EOR $66     chen
BAA7   85 6F      STA $6F     von
BAA9   A5 6E      LDA $6E     FAC
BAAB   09 80      ORA #$80    und
BAAD   85 6A      STA $6A     ARG
BAAF   88         DEY         ver-
BAB0   B1 22      LDA ($22),Y knüp-
BAB2   85 69      STA $69     fen
BAB4   A5 61      LDA $61     FAC-Exponent
BAB6   60         RTS         Rücksprung
```

```
Einsprung von $BA30, $BB1E

BAB7   A5 69      LDA $69     wenn Exponent von ARG=0,

Einsprung von $E03F

BAB9   F0 1F      BEQ $BADA   dann zu $BADA
BABB   18         CLC         FAC- und ARG-
BABC   65 61      ADC $61     Exponent
BABE   90 04      BCC $BAC4   addieren
BAC0   30 1D      BMI $BADF   wenn überlauf, dann
                              'OVERFLOW ERROR'
BAC2   18         CLC         Carry
BAC3   2C         .BYTE $2C   löschen
BAC4   10 14      BPL $BADA   Wenn Unterlauf, dann zu $BADA
BAC6   69 80      ADC #$80    ergibt
BAC8   85 61      STA $61     FAC-
BACA   D0 03      BNE $BACF   Exponent
BACC   4C FB B8   JMP $B8FB   FAC = 0
BACF   A5 6F      LDA $6F     FAC- und ARG-Vorzeichen
                              verknüpfen
BAD1   85 66      STA $66     und speichern
BAD3   60         RTS         Rücksprung

Einsprung von $E00B

BAD4   A5 66      LDA $66     wenn positives
BAD6   49 FF      EOR #$FF    Vorzeichen, dann
BAD8   30 05      BMI $BADF   'OVERFLOW ERROR'
BADA   68         PLA         Einsprungadresse
BADB   68         PLA         vom Stack holen
BADC   4C F7 B8   JMP $B8F7   FAC = 0
BADF   4C 7E B9   JMP $B97E   'OVERFLOW ERROR'
```

```
****************************** FAC = FAC * 10

Einsprung von $A9F2, $AA0E, $BD5B, $BD71, $BE21

BAE2   20 0C BC   JSR $BC0C   FAC runden und nach ARG
BAE5   AA         TAX         FAC-Exponent
BAE6   F0 10      BEQ $BAF8   FAC gleich null, dann fertig
BAE8   18         CLC         Exponent + 2
BAE9   69 02      ADC #$02    entspricht mal 4
BAEB   B0 F2      BCS $BADF   Übertrag ?

Einsprung von $AA04

BAED   A2 00      LDX #$00    Vergleichsbyte
BAEF   86 6F      STX $6F     löschen
BAF1   20 77 B8   JSR $B877   FAC = FAC + ARG  entspricht
                              mal 5
BAF4   E6 61      INC $61     Exponent erhöhen entspricht
                              mal 2
BAF6   F0 E7      BEQ $BADF   Übertrag, dann 'OVERFLOW'
BAF8   60         RTS         Rücksprung

******************************
BAF9   84 20 00 00 00         Fließkommakonstante 10

****************************** FAC = FAC / 10

Einsprung von $BD52, $BE28

BAFE   20 0C BC   JSR $BC0C   FAC runden und nach ARG
BB01   A9 F9      LDA #$F9    Zeiger
BB03   A0 BA      LDY #$BA    auf
BB05   A2 00      LDX #$00    Konstante 10

Einsprung von $E274

BB07   86 6F      STX $6F     Vergleichsbyte löschen
BB09   20 A2 BB   JSR $BBA2   Konstante 10 nach FAC
BB0C   4C 12 BB   JMP $BB12   FAC = ARG / FAC
```

```
****************************** FAC = Konstante (A/Y) / FAC

Einsprung von $BA08, $E2D9, $E321

BB0F   20 8C BA   JSR $BA8C   Konstante (A/Y) nach ARG

****************************** FAC = ARG / FAC

Einsprung von $BB0C

BB12   F0 76      BEQ $BB8A   FAC gleich null,
                              'DIVISION BY ZERO'
BB14   20 1B BC   JSR $BC1B   FAC runden
BB17   A9 00      LDA #$00    Vorzeichen
BB19   38         SEC         von FAC-
BB1A   E5 61      SBC $61     Exponent
BB1C   85 61      STA $61     wechseln
BB1E   20 B7 BA   JSR $BAB7   Exponent des Ergebnisses
                              bestimmen
BB21   E6 61      INC $61     wenn Exponentenüberlauf,
BB23   F0 BA      BEQ $BADF   dann 'OVERFLOW ERROR'
BB25   A2 FC      LDX #$FC    Zeiger
BB27   A9 01      LDA #$01    auf
BB29   A4 6A      LDY $6A     Funktionsregister
BB2B   C4 62      CPY $62     diese
BB2D   D0 10      BNE $BB3F   Routine
BB2F   A4 6B      LDY $6B     vergleicht
BB31   C4 63      CPY $63     das
BB33   D0 0A      BNE $BB3F   FAC
BB35   A4 6C      LDY $6C     und
BB37   C4 64      CPY $64     das
BB39   D0 04      BNE $BB3F   ARG
BB3B   A4 6D      LDY $6D     byte-
BB3D   C4 65      CPY $65     weise
BB3F   08         PHP         Statusregister retten
BB40   2A         ROL  A      Carry gelöscht,
BB41   90 09      BCC $BB4C   dann zu $BB4C
BB43   E8         INX         Ergebnis
BB44   95 29      STA $29,X   aufbauen
BB46   F0 32      BEQ $BB7A   wenn X-Reg =0, dann zu $BB7A
```

```
BB48   10 34      BPL $BB7E   wenn X-Reg =1, dann zu $BB7E
BB4A   A9 01      LDA #$01    wenn
BB4C   28         PLP         FAC kleiner oder gleich
BB4D   B0 0E      BCS $BB5D   ARG, dann zu $BB5D

Einsprung von $BB77

BB4F   06 6D      ASL $6D     Das
BB51   26 6C      ROL $6C     ARG
BB53   26 6B      ROL $6B     ver-
BB55   26 6A      ROL $6A     doppeln
BB57   B0 E6      BCS $BB3F   wenn Überlauf, dann zu $BB3F
BB59   30 CE      BMI $BB29   wenn Bit 7 gesetzt, dann
                              zu $BB29
BB5B   10 E2      BPL $BB3F   ansonsten zu $BB3F
BB5D   A8         TAY         Die
BB5E   A5 6D      LDA $6D     Mantisse
BB60   E5 65      SBC $65     von
BB62   85 6D      STA $6D     ARG
BB64   A5 6C      LDA $6C     minus
BB66   E5 64      SBC $64     der
BB68   85 6C      STA $6C     Mantisse
BB6A   A5 6B      LDA $6B     von
BB6C   E5 63      SBC $63     FAC
BB6E   85 6B      STA $6B     sub-
BB70   A5 6A      LDA $6A     tra-
BB72   E5 62      SBC $62     hie-
BB74   85 6A      STA $6A     ren
BB76   98         TYA         und wieder
BB77   4C 4F BB   JMP $BB4F   zu $BB4C
BB7A   A9 40      LDA #$40    unbedingter
BB7C   D0 CE      BNE $BB4C   Sprung
BB7E   0A         ASL  A      den
BB7F   0A         ASL  A      Akku
BB80   0A         ASL  A      mit
BB81   0A         ASL  A      64
BB82   0A         ASL  A      multi-
BB83   0A         ASL  A      plizieren
```

```
BB84   85 70      STA $70     Ergeben = Rundungsstelle
BB86   28         PLP         Statusregister aus Stack
BB87   4C 8F BB   JMP $BB8F   Hilfsregister nach FAC

******************************
BB8A   A2 14      LDX #$14    Nummer für 'DIVISION BY ZERO'
BB8C   4C 37 A4   JMP $A437   Fehlermeldung ausgeben

******************************

Einsprung von $BA56, $BB87

BB8F   A5 26      LDA $26     Hilfs-
BB91   85 62      STA $62     register
BB93   A5 27      LDA $27     ($26 - $29)
BB95   85 63      STA $63     nach
BB97   A5 28      LDA $28     FAC
BB99   85 64      STA $64     über-
BB9B   A5 29      LDA $29     tra-
BB9D   85 65      STA $65     gen
BB9F   4C D7 B8   JMP $B8D7   FAC linksbündig machen

****************************** Konstante (A/Y) nach FAC
                               übertragen

Einsprung von $A78F, $AD42, $AEA2, $AFA4, $BB09, $BF78,
              $E0C2, $E2C9

BBA2   85 22      STA $22     Zeiger
BBA4   84 23      STY $23     setzen
BBA6   A0 04      LDY #$04    Zähler setzen
BBA8   B1 22      LDA ($22),Y LOW-Byte
BBAA   85 65      STA $65     der
BBAC   88         DEY         Mantisse
BBAD   B1 22      LDA ($22),Y und
BBAF   85 64      STA $64     HIGH-
BBB1   88         DEY         Byte
BBB2   B1 22      LDA ($22),Y der
BBB4   85 63      STA $63     Mantisse
BBB6   88         DEY         in
```

```
BBB7   B1 22      LDA ($22),Y FAC
BBB9   85 66      STA $66     holen
BBBB   09 80      ORA #$80    Vorzeichen
BBBD   85 62      STA $62     der
BBBF   88         DEY         Man-
BBC0   B1 22      LDA ($22),Y tisse
BBC2   85 61      STA $61     Exponent
BBC4   84 70      STY $70     Rundungsstelle
BBC6   60         RTS         Rücksprung

******************************

Einsprung von $E05D

BBC7   A2 5C      LDX #$5C    Adresse LOW
BBC9   2C         .BYTE $2C   Akku #4

****************************** FAC nach Akku #3 übertragen

Einsprung von $E047, $E2B4

BBCA   A2 57      LDX #$57    Adresse LOW Akku #3
BBCC   A0 00      LDY #$00    Adresse HIGH
BBCE   F0 04      BEQ $BBD4   unbedingter Sprung

Einsprung von $A9D4, $AD52

****************************** FAC nach Variable übertragen
BBD0   A6 49      LDX $49     Variablenadresse
BBD2   A4 4A      LDY $4A     holen

Einsprung von $B420, $BF88, $E0F6

BBD4   20 1B BC   JSR $BC1B   FAC runden
BBD7   86 22      STX $22     Zeiger auf
BBD9   84 23      STY $23     Zieladresse
BBDB   A0 04      LDY #$04    Zähler setzen
BBDD   A5 65      LDA $65     LOW-Byte der Mantisse
BBDF   91 22      STA ($22),Y Den
BBE1   88         DEY         FAC
```

```
BBE2   A5 64      LDA $64      in
BBE4   91 22      STA ($22),Y  den
BBE6   88         DEY          Ziel-
BBE7   A5 63      LDA $63      bereich
BBE9   91 22      STA ($22),Y  über-
BBEB   88         DEY          tragen
BBEC   A5 66      LDA $66      FAC-Vorzeichen
BBEE   09 7F      ORA #$7F     Die Bits 0 bis 6 setzen
BBF0   25 62      AND $62      Vorzeichen auf
BBF2   91 22      STA ($22),Y  Speicherformat
BBF4   88         DEY          bringen
BBF5   A5 61      LDA $61      FAC-Exponent
BBF7   91 22      STA ($22),Y  übertragen
BBF9   84 70      STY $70      FAC-Rundungsstelle löschen
BBFB   60         RTS          Rücksprung

****************************** ARG nach FAC übertragen

Einsprung von $AFFC, $B86C

BBFC   A5 6E      LDA $6E      ARG-Vorzeichen

Einsprung von $BF9E

BBFE   85 66      STA $66      in FAC-Reg übertragen
BC00   A2 05      LDX #$05     5 Bytes
BC02   B5 68      LDA $68,X    ARG in
BC04   95 60      STA $60,X    FAC
BC06   CA         DEX          übertragen
BC07   D0 F9      BNE $BC02    schon alle Zeichen ?
BC09   86 70      STX $70      FAC-Rundungsstelle löschen
BC0B   60         RTS          Rücksprung

****************************** FAC nach ARG übertragen

Einsprung von $A9FC, $BAE2, $BAFE, $BD7F, $BF71, $E26B,
              $E277

BC0C   20 1B BC   JSR $BC1B    FAC runden
```

```
Einsprung von $E002

BC0F   A2 06      LDX #$06    6 Zeichen
BC11   B5 60      LDA $60,X   FAC in
BC13   95 68      STA $68,X   ARG
BC15   CA         DEX         übertragen
BC16   D0 F9      BNE $BC11   schon alle Zeichen ?
BC18   86 70      STX $70     FAC-Rundungsstelle löschen
BC1A   60         RTS         Rücksprung

****************************** FAC runden

Einsprung von $A9C4, $AE43, $BB14, $BBD4, $BC0C

BC1B   A5 61      LDA $61     Exponent null ?,
BC1D   F0 FB      BEQ $BC1A   dann fertig
BC1F   06 70      ASL $70     Rundungsstelle größer $7F ?
BC21   90 F7      BCC $BC1A   nein, dann fertig

Einsprung von $BFFA

BC23   20 6F B9   JSR $B96F   Mantisse um eins erhöhen
BC26   D0 F2      BNE $BC1A   jetzt null ?
BC28   4C 38 B9   JMP $B938   nach rechts verschieben,
                              Exponent erhöhen

****************************** Vorzeichen von FAC holen

Einsprung von $A79F, $B9EA, $BC39, $E097

BC2B   A5 61      LDA $61     wenn null,
BC2D   F0 09      BEQ $BC38   dann RTS
BC2F   A5 66      LDA $66     FAC-Vorzeichen

Einsprung von $BC98

BC31   2A         ROL  A      holen
BC32   A9 FF      LDA #$FF    negativ?
BC34   B0 02      BCS $BC38   dann RTS
```

```
BC36   A9 01      LDA #$01    sonst positiv
BC38   60         RTS         Rücksprung

****************************** BASIC-Funktion SGN
BC39   20 2B BC   JSR $BC2B   Vorzeichen holen
BC3C   85 62      STA $62     und in FAC speichern
BC3E   A9 00      LDA #$00    $63
BC40   85 63      STA $63     löschen
BC42   A2 88      LDX #$88    Exponent

Einsprung von $B39B

BC44   A5 62      LDA $62     Vorzeichen
BC46   49 FF      EOR #$FF    invertieren
BC48   2A         ROL  A      und nach links rollen

Einsprung von $BDD4

BC49   A9 00      LDA #$00    Die Adressen
BC4B   85 65      STA $65     $65
BC4D   85 64      STA $64     und $64 löschen

Einsprung von $AF81

BC4F   86 61      STX $61     Exponent
BC51   85 70      STA $70     Rundungsstelle
BC53   85 66      STA $66     löschen
BC55   4C D2 B8   JMP $B8D2   linksbündig machen

****************************** BASIC-Funktion  ABS
BC58   46 66      LSR $66     Vorzeichenbit löschen
BC5A   60         RTS         Rücksprung

****************************** Vergleich Konstante (A/Y) mit
                              FAC

Einsprung von $B027, $B1C9, $BE0F, $BE1A, $BF96

BC5B   85 24      STA $24     Zeiger auf
```

Einsprung von $AD57

```
BC5D   84 25     STY $25      Konstante
BC5F   A0 00     LDY #$00     Zähler setzen
BC61   B1 24     LDA ($24),Y  Exponent
BC63   C8        INY          Zähler erhöhen
BC64   AA        TAX          ins X-Reg
BC65   F0 C4     BEQ $BC2B    null?, dann Vorzeichen von
                              FAC holen
BC67   B1 24     LDA ($24),Y  Konstante
BC69   45 66     EOR $66      FAC-Vorzeichen
BC6B   30 C2     BMI $BC2F    verschiedene Vorzeichen?,
                              dann zu $BC2F
BC6D   E4 61     CPX $61      Exponenten vergleichen
BC6F   D0 21     BNE $BC92    falls verschieden, dann zu
                              $BC92
BC71   B1 24     LDA ($24),Y  das
BC73   09 80     ORA #$80     erste
BC75   C5 62     CMP $62      Byte
BC77   D0 19     BNE $BC92    vergleichen
BC79   C8        INY          Zähler erhöhen
BC7A   B1 24     LDA ($24),Y  das zweite
BC7C   C5 63     CMP $63      Byte
BC7E   D0 12     BNE $BC92    vergleichen
BC80   C8        INY          Zähler erhöhen
BC81   B1 24     LDA ($24),Y  das dritte
BC83   C5 64     CMP $64      Byte
BC85   D0 0B     BNE $BC92    vergleichen
BC87   C8        INY          Zähler erhöhen
BC88   A9 7F     LDA #$7F     FAC-Rundungsstelle mit
BC8A   C5 70     CMP $70      $7F vergleichen
BC8C   B1 24     LDA ($24),Y  letzte Stellen, gemäß Ver-
BC8E   E5 65     SBC $65      gleich der Rundungsstelle,
                              subtrahieren
BC90   F0 28     BEQ $BCBA    wenn alle Stellen gleich
                              sind, dann RTS
BC92   A5 66     LDA $66      FAC-Vorzeichen
BC94   90 02     BCC $BC98    ist die Konstante kleiner
                              FAC, dann zu $BC98
```

```
BC96   49 FF      EOR #$FF    Ergebnis kleiner, dann
                              invertieren
BC98   4C 31 BC   JMP $BC31   Flag für Ergebnis setzen

****************************** Umwandlung Fließkomma nach
                              Integer

Einsprung von $AA11, $B1CE, $B801, $BCD2, $BE32

BC9B   A5 61      LDA $61     Exponent
BC9D   F0 4A      BEQ $BCE9   null ?
BC9F   38         SEC         Integer-
BCA0   E9 A0      SBC #$A0    Exponent
BCA2   24 66      BIT $66     wenn FAC positiv,
BCA4   10 09      BPL $BCAF   dann zu $BCAF
BCA6   AA         TAX         FAC
BCA7   A9 FF      LDA #$FF    Rundungsbyte
BCA9   85 68      STA $68     setzen
BCAB   20 4D B9   JSR $B94D   Mantisse von FAC invertieren
BCAE   8A         TXA         Exponent in Akku
BCAF   A2 61      LDX #$61    FAC-Offset-Zeiger
BCB1   C9 F9      CMP #$F9    wenn Exponent größer als
BCB3   10 06      BPL $BCBB   -8, dann zu BCBB
BCB5   20 99 B9   JSR $B999   FAC rechtsverschieben
BCB8   84 68      STY $68     FAC-Rundungsbyte löschen
BCBA   60         RTS         Rücksprung

BCBB   A8         TAY         Akku löschen
BCBC   A5 66      LDA $66     FAC-Vorzeichen laden
BCBE   29 80      AND #$80    das
BCC0   46 62      LSR $62     FAC-
BCC2   05 62      ORA $62     Vorzeichen
BCC4   85 62      STA $62     isolieren
BCC6   20 B0 B9   JSR $B9B0   FAC bitweise nach rechts
                              verschieben
BCC9   84 68      STY $68     FAC-Rundungsbyte löschen
BCCB   60         RTS         Rücksprung
```

```
****************************** BASIC-Funktion  INT

Einsprung von $BF8F, $E00E, $E27A

BCCC   A5 61      LDA $61     Exponent
BCCE   C9 A0      CMP #$A0    ganze Zahl ?
BCD0   B0 20      BCS $BCF2   ja, dann fertig
BCD2   20 9B BC   JSR $BC9B   FAC nach Integer wandeln
BCD5   84 70      STY $70     Rundungsstelle löschen
BCD7   A5 66      LDA $66     Vorzeichen in Akku
BCD9   84 66      STY $66     und positiv machen
BCDB   49 80      EOR #$80    Bei
BCDD   2A         ROL  A      negativen Vorzeichen
BCDE   A9 A0      LDA #$A0    das
BCE0   85 61      STA $61     Carry-
BCE2   A5 65      LDA $65     flag
BCE4   85 07      STA $07     löschen
BCE6   4C D2 B8   JMP $B8D2   FAC linksbündig machen
BCE9   85 62      STA $62     Mantisse
BCEB   85 63      STA $63     mit
BCED   85 64      STA $64     Nullen
BCEF   85 65      STA $65     füllen
BCF1   A8         TAY         Y-Reg löschen
BCF2   60         RTS         Rücksprung

****************************** Umwandlung ASCII nach
                               Fließkommaformat

Einsprung von $AC89, $AE8F, $B7DA

BCF3   A0 00      LDY #$00    Wert festlegen
BCF5   A2 0A      LDX #$0A    Zähler stellen
BCF7   94 5D      STY $5D,X   den Bereich
BCF9   CA         DEX         von $5D bis $66 mit
BCFA   10 FB      BPL $BCF7   Nullen füllen
BCFC   90 0F      BCC $BD0D   wenn erstes Zeichen eine
                              Ziffer, dann zu $BD0D
BCFE   C9 2D      CMP #$2D    Nummer für '-'?
BD00   D0 04      BNE $BD06   wenn nicht, dann zu $BD06
BD02   86 67      STX $67     Flag für negativ
```

```
BD04   F0 04      BEQ $BD0A   unbedingter Sprung
BD06   C9 2B      CMP #$2B    Nummer für '+'
BD08   D0 05      BNE $BD0F   wenn nicht, dann zu $BD0F

Einsprung von $BD7B

BD0A   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
BD0D   90 5B      BCC $BD6A   wenn Ziffer, dann zu $BD6A
BD0F   C9 2E      CMP #$2E    Nummer für '.'
BD11   F0 2E      BEQ $BD41   wenn ja, dann zu $BD41
BD13   C9 45      CMP #$45    Nummer für 'E'
BD15   D0 30      BNE $BD47   wenn nicht, dann zu $BD47
BD17   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
BD1A   90 17      BCC $BD33   wenn Ziffer, dann zu $BD33
BD1C   C9 AB      CMP #$AB    '-' BASIC-Kode
BD1E   F0 0E      BEQ $BD2E   wenn ja, dann zu $BD2E
BD20   C9 2D      CMP #$2D    Nummer für '-'
BD22   F0 0A      BEQ $BD2E   wenn ja, dann zu $BD2E
BD24   C9 AA      CMP #$AA    '+' BASIC-Kode
BD26   F0 08      BEQ $BD30   wenn ja, dann zu $BD30
BD28   C9 2B      CMP #$2B    Nummer für '+'
BD2A   F0 04      BEQ $BD30   wenn ja, dann zu $BD30
BD2C   D0 07      BNE $BD35   unbedingter Sprung
BD2E   66 60      ROR $60     Bit 7 setzen

Einsprung von $BDB0

BD30   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
BD33   90 5C      BCC $BD91   wenn Ziffer, dann zu $BD91
BD35   24 60      BIT $60     Bit 7 gesetzt ?
BD37   10 0E      BPL $BD47   wenn nicht, dann zu $BD47
BD39   A9 00      LDA #$00    Vorzeichen des
BD3B   38         SEC         Exponenten
BD3C   E5 5E      SBC $5E     wechseln
BD3E   4C 49 BD   JMP $BD49   weiter bei $BD49
BD41   66 5F      ROR $5F     Aufruf durch Dezimalpunkt
BD43   24 5F      BIT $5F     schon zweiter Dezimalpunkt
BD45   50 C3      BVC $BD0A   wenn nicht, dann weiter
BD47   A5 5E      LDA $5E     Zahl gemäß
```

```
Einsprung von $BD3E

BD49   38          SEC          Position
BD4A   E5 5D       SBC $5D      des Dezimalpunkts
BD4C   85 5E       STA $5E      und Exponenten anpassen
BD4E   F0 12       BEQ $BD62    Zahl= Null, dann zu $BD62
BD50   10 09       BPL $BD5B    Zahl kleiner als $7F
BD52   20 FE BA    JSR $BAFE    FAC = FAC / 10
BD55   E6 5E       INC $5E      Zahl erhöhen
BD57   D0 F9       BNE $BD52    unbedingter
BD59   F0 07       BEQ $BD62    Sprung
BD5B   20 E2 BA    JSR $BAE2    FAC = FAC * 10
BD5E   C6 5E       DEC $5E      Zahl gemäß
BD60   D0 F9       BNE $BD5B    Exponenten anpassen
BD62   A5 67       LDA $67      wenn negativ,
BD64   30 01       BMI $BD67    dann Vorzeichen invertieren
BD66   60          RTS          Rücksprung
BD67   4C B4 BF    JMP $BFB4    Vorzeichenwechsel  FAC = -FAC

BD6A   48          PHA          Aufruf durch Mantisse
BD6B   24 5F       BIT $5F      wenn Vorkommastelle,
BD6D   10 02       BPL $BD71    dann zu $BD71
BD6F   E6 5D       INC $5D      Zähler erhöhen
BD71   20 E2 BA    JSR $BAE2    FAC = FAC * 10
BD74   68          PLA          ASC II in
BD75   38          SEC          Ziffer umwandeln
BD76   E9 30       SBC #$30     '0' abziehen gibt hex
BD78   20 7E BD    JSR $BD7E    addiert nächste Stelle zu FAC
BD7B   4C 0A BD    JMP $BD0A    nächstes Zeichen

Einsprung von $AA29, $BA21, $BD78

BD7E   48          PHA          Wert aus Stack
BD7F   20 0C BC    JSR $BC0C    FAC nach ARG
BD82   68          PLA          Wert in Stack
BD83   20 3C BC    JSR $BC3C    Accu in höchste Stelle von FAC
BD86   A5 6E       LDA $6E      FAC-Vorzeichen und
BD88   45 66       EOR $66      ARG-Vorzeichen
BD8A   85 6F       STA $6F      verknüpfen
BD8C   A6 61       LDX $61      erste Stelle von FAC holen
```

```
BD8E   4C 6A B8   JMP $B86A    FAC = FAC + ARG
BD91   A5 5E      LDA $5E      Aufruf durch 'E'
BD93   C9 0A      CMP #$0A     wenn dritte Exponentenziffer,
BD95   90 09      BCC $BDA0    dann zu $BDA0
BD97   A9 64      LDA #$64     wenn Vorzeichen
BD99   24 60      BIT $60      negativ,
BD9B   30 11      BMI $BDAE    dann Unterlauf
BD9D   4C 7E B9   JMP $B97E    zu 'OVERFLOW ERROR'

BDA0   0A         ASL  A       Den
BDA1   0A         ASL  A       Exponenten
BDA2   18         CLC          mit
BDA3   65 5E      ADC $5E      10
BDA5   0A         ASL  A       multi-
BDA6   18         CLC          plizieren
BDA7   A0 00      LDY #$00     Zähler setzen
BDA9   71 7A      ADC ($7A),Y  Exponenten-
BDAB   38         SEC          ziffer
BDAC   E9 30      SBC #$30     addie-
BDAE   85 5E      STA $5E      ren
BDB0   4C 30 BD   JMP $BD30    nächstes Zeichen holen

****************************** Konstanten für Fließkomma
                               nach ASCII
BDB3   9B 3E BC 1F FD          99999999.9
BDB8   9E 6E 6B 27 FD          999999999
BDBD   9E 6E 6B 28 00          1E9

****************************** Ausgabe der Zeilennummer
                               bei Fehlermeldung

Einsprung von $A471

BDC2   A9 71      LDA #$71     Zeiger
BDC4   A0 A3      LDY #$A3     auf 'in'
BDC6   20 DA BD   JSR $BDDA    String ausgeben
BDC9   A5 3A      LDA $3A      laufende
BDCB   A6 39      LDX $39      Zeilennummer holen
```

```
****************************** positive Integerzahl
                               in A/X ausgeben

Einsprung von $A6EA, $E43A

BDCD   85 62      STA $62      für Umwandlung
BDCF   86 63      STX $63      in FAC schreiben
BDD1   A2 90      LDX #$90     Exponent
BDD3   38         SEC          = 16
BDD4   20 49 BC   JSR $BC49    Integer nach Fließkomma
                               wandeln
BDD7   20 DF BD   JSR $BDDF    FAC nach ASCII wandeln

Einsprung von $BDC6

BDDA   4C 1E AB   JMP $AB1E    String ausgeben

****************************** FAC nach ASCII-Format
                               wandeln und nach $100

Einsprung von $AABC

BDDD   A0 01      LDY #$01     Stringzeiger

Einsprung von $B46A, $BDD7

BDDF   A9 20      LDA #$20     ' ' Leerzeichen für positive
                               Zahl
BDE1   24 66      BIT $66      wenn Vorzeichen
BDE3   10 02      BPL $BDE7    positiv ?, dann zu $BDE7
BDE5   A9 2D      LDA #$2D     '-' Minuszeichen für
                               negative Zahl
BDE7   99 FF 00   STA $00FF,Y  in
BDEA   85 66      STA $66      Pufferbereich
BDEC   84 71      STY $71      schreiben
BDEE   C8         INY          Zähler erhöhen
BDEF   A9 30      LDA #$30     '0'
BDF1   A6 61      LDX $61      Exponent
BDF3   D0 03      BNE $BDF8    wenn Zahl nicht null ?
BDF5   4C 04 BF   JMP $BF04    dann fertig
```

```
BDF8  A9 00     LDA #$00    FAC
BDFA  E0 80     CPX #$80    mit 1 vergleichen
BDFC  F0 02     BEQ $BE00   wenn ja ,dann zu $BE00
BDFE  B0 09     BCS $BE09   FAC größer 1
BE00  A9 BD     LDA #$BD    Zeiger auf
BE02  A0 BD     LDY #$BD    Konstante  1E9
BE04  20 28 BA  JSR $BA28   Konstante (Zeiger A/Y) * FAC
BE07  A9 F7     LDA #$F7    = -9
BE09  85 5D     STA $5D     $ 5D = -9
BE0B  A9 B8     LDA #$B8    Zeiger auf
BE0D  A0 BD     LDY #$BD    Konstante  999999999
BE0F  20 5B BC  JSR $BC5B   Vergleich Konstante
                            (Zeiger A/Y) mit FAC
BE12  F0 1E     BEQ $BE32   gleich
BE14  10 12     BPL $BE28   kleiner
BE16  A9 B3     LDA #$B3    Zeiger auf
BE18  A0 BD     LDY #$BD    Konstante  99999999.9
BE1A  20 5B BC  JSR $BC5B   Vergleich Konstante
                            (Zeiger A/Y) mit FAC
BE1D  F0 02     BEQ $BE21   gleich
BE1F  10 0E     BPL $BE2F   kleiner
BE21  20 E2 BA  JSR $BAE2   FAC = FAC * 10
BE24  C6 5D     DEC $5D     Dezimalexponent erniedrigen
BE26  D0 EE     BNE $BE16   schon 0?
BE28  20 FE BA  JSR $BAFE   FAC = FAC / 10
BE2B  E6 5D     INC $5D     Dezimalexponent erhöhen
BE2D  D0 DC     BNE $BE0B   überlauf ?
BE2F  20 49 B8  JSR $B849   FAC = FAC + .5 , runden
BE32  20 9B BC  JSR $BC9B   FAC nach Integer
BE35  A2 01     LDX #$01    FAC ist nun im Bereich von
BE37  A5 5D     LDA $5D     1E8 bis 1E9, $5D hat Wert
BE39  18        CLC         von Zehnerpotenz
BE3A  69 0A     ADC #$0A    Zahl =0.01
BE3C  30 09     BMI $BE47   Betrag kleiner 0.1 ?
BE3E  C9 0B     CMP #$0B    wenn ja, dann
BE40  B0 06     BCS $BE48   Betrag größer 1E9 ?
BE42  69 FF     ADC #$FF    die
BE44  AA        TAX         Be-
BE45  A9 02     LDA #$02    rechnung
BE47  38        SEC         des
```

```
BE48   E9 02      SBC #$02     Exponenten-
BE4A   85 5E      STA $5E      flags
BE4C   86 5D      STX $5D      Negative Darstellung des
BE4E   8A         TXA          Exponenten
BE4F   F0 02      BEQ $BE53    wenn 0.1, dann zu $BE53
BE51   10 13      BPL $BE66    wenn nicht 0.01, dann
                               zu $BE66
BE53   A4 71      LDY $71      Zeiger für Polynomauswertung
BE55   A9 2E      LDA #$2E     Nummer für '.'
BE57   C8         INY          Zeiger erhöhen
BE58   99 FF 00   STA $00FF,Y  in Stringbereich
BE5B   8A         TXA          schreiben
BE5C   F0 06      BEQ $BE64    wenn 0.1, dann zu $BE64
BE5E   A9 30      LDA #$30     Nummer für '0'
BE60   C8         INY          Zeiger erhöhen
BE61   99 FF 00   STA $00FF,Y  in Stringbereich
BE64   84 71      STY $71      schreiben
BE66   A0 00      LDY #$00     Zeiger

Einsprung von $AF56

BE68   A2 80      LDX #$80     stellen
BE6A   A5 65      LDA $65      Durch
BE6C   18         CLC          Addition
BE6D   79 19 BF   ADC $BF19,Y  und
BE70   85 65      STA $65      Subtraktion
BE72   A5 64      LDA $64      der
BE74   79 18 BF   ADC $BF18,Y  Werte
BE77   85 64      STA $64      aus
BE79   A5 63      LDA $63      der
BE7B   79 17 BF   ADC $BF17,Y  Tabelle
BE7E   85 63      STA $63      werden
BE80   A5 62      LDA $62      die
BE82   79 16 BF   ADC $BF16,Y  einzelnen
BE85   85 62      STA $62      Ziffern
BE87   E8         INX          des
BE88   B0 04      BCS $BE8E    Zahlen-
BE8A   10 DE      BPL $BE6A    Strings
BE8C   30 02      BMI $BE90    be-
BE8E   30 DA      BMI $BE6A    rech-
```

```
BE90  8A        TXA          net
BE91  90 04     BCC $BE97    alles addiert?, wenn nicht,
                             dann zu $BE97
BE93  49 FF     EOR #$FF     Ergebnis mit 10
BE95  69 0A     ADC #$0A     komplementieren
BE97  69 2F     ADC #$2F     '0' - 1
BE99  C8        INY          Zähler
BE9A  C8        INY          ent-
BE9B  C8        INY          sprechend
BE9C  C8        INY          erhöhen
BE9D  84 47     STY $47      Zähler sichern
BE9F  A4 71     LDY $71      Zeiger auf Stringbereich laden
BEA1  C8        INY          und erhöhen
BEA2  AA        TAX          Ziffer
BEA3  29 7F     AND #$7F     in
BEA5  99 FF 00  STA $00FF,Y  Stringbereich
BEA8  C6 5D     DEC $5D      bringen
BEAA  D0 06     BNE $BEB2    wenn Einerstelle nicht
                             erreicht, dann zu $BEB2
BEAC  A9 2E     LDA #$2E     Nummer für '.'
BEAE  C8        INY          Zähler erhöhen
BEAF  99 FF 00  STA $00FF,Y  in Stringbereich schreiben
BEB2  84 71     STY $71      Zähler speichern
BEB4  A4 47     LDY $47      Neuen Zähler holen
BEB6  8A        TXA          FAC-
BEB7  49 FF     EOR #$FF     Um-
BEB9  29 80     AND #$80     wand-
BEBB  AA        TAX          lung
BEBC  C0 24     CPY #$24     Tabellenende ereicht,
BEBE  F0 04     BEQ $BEC4    dann zu $BEC4
BEC0  C0 3C     CPY #$3C     Tabellenende bei
                             TI$-Berechnung
BEC2  D0 A6     BNE $BE6A    nicht erreicht, dann zu $BE6A
BEC4  A4 71     LDY $71      Zähler wieder holen
BEC6  B9 FF 00  LDA $00FF,Y  letzte Stelle suchen
BEC9  88        DEY          Zähler erniedrigen
BECA  C9 30     CMP #$30     Nummer für '0'
BECC  F0 F8     BEQ $BEC6    wenn ja, dann zu $BEC6
BECE  C9 2E     CMP #$2E     Nummer für '.'
BED0  F0 01     BEQ $BED3    wenn ja, dann zu $BED3
```

```
BED2   C8          INY          Zähler erhöhen
BED3   A9 2B       LDA #$2B     Nummer für '+'
BED5   A6 5E       LDX $5E      wenn Flag nicht gesetzt,
BED7   F0 2E       BEQ $BF07    dann zu $BF07
BED9   10 08       BPL $BEE3    wenn Exponent positiv, dann
                                zu $BEE3
BEDB   A9 00       LDA #$00     Den
BEDD   38          SEC          Exponenten
BEDE   E5 5E       SBC $5E      be-
BEE0   AA          TAX          rechnen
BEE1   A9 2D       LDA #$2D     Nummer für '-'
BEE3   99 01 01    STA $0101,Y  in Stringbereich schreiben
BEE6   A9 45       LDA #$45     Nummer für 'E'
BEE8   99 00 01    STA $0100,Y  in Stringbereich schreiben
BEEB   8A          TXA          Zehner-
BEEC   A2 2F       LDX #$2F     stelle
BEEE   38          SEC          für
BEEF   E8          INX          den
BEF0   E9 0A       SBC #$0A     Exponenten
BEF2   B0 FB       BCS $BEEF    berechnen
BEF4   69 3A       ADC #$3A     '9' + 1
BEF6   99 03 01    STA $0103,Y  in Stringbereich schreiben
BEF9   8A          TXA          Zehnerstelle
BEFA   99 02 01    STA $0102,Y  in Stringbereich schreiben
BEFD   A9 00       LDA #$00     Puffer mit $0
BEFF   99 04 01    STA $0104,Y  abschließen
BF02   F0 08       BEQ $BF0C    unbedingter Sprung

Einsprung von $BDF5

BF04   99 FF 00    STA $00FF,Y  Puffer
BF07   A9 00       LDA #$00     mit $0
BF09   99 00 01    STA $0100,Y  abschließen
BF0C   A9 00       LDA #$00     Zeiger auf
BF0E   A0 01       LDY #$01     Puffer $100
BF10   60          RTS          Rücksprung
```

```
*****************************
BF11   80 00 00 00 00          Konstante 0.5 für
                               SQR-Funktion

***************************** Konstanten für Gleitkomma
                               nach ASCII
32-Bit Binärzahlen mit Vorzeichen
BF16   FA 0A 1F 00             -100 000 000
BF1A   00 98 96 80               10 000 000
BF1E   FF F0 BD C0               -1 000 000
BF22   00 01 86 A0                  100 000
BF26   FF FF D8 F0                  -10 000
BF2A   00 00 03 E8                    1 000
BF2E   FF FF FF 9C                    - 100
BF32   00 00 00 0A                       10
BF36   FF FF FF FF                       -1

***************************** Konstanten für Umwandlung
                               TI nach TI$
BF3A   FF DF 0A 80             -2 160 000
BF3E   00 03 4B C0                216 000
BF42   FF FF 73 60                -36 000
BF46   00 00 0E 10                  3 600
BF4A   FF FF FD A8                  - 600
BF4E   00 00 00 3C                     60
BF52   EC

BF53   AA ...
BF70   ...  AA

*****************************  BASIC-Funktion  SQR
BF71   20 0C BC    JSR $BC0C   FAC runden und nach ARG
BF74   A9 11       LDA #$11    Zeiger auf
BF76   A0 BF       LDY #$BF    Konstante 0.5

*****************************  Potenzierung  FAC = ARG
                               hoch Konstante (A/Y)
BF78   20 A2 BB    JSR $BBA2   Konstante nach FAC
```

```
**************************** Potenzierung  FAC = ARG
                              hoch FAC
BF7B   F0 70      BEQ $BFED   wenn FAC=0, dann zu $BFED
BF7D   A5 69      LDA $69     Exponent ARG = Basis
BF7F   D0 03      BNE $BF84   nicht null ?,
BF81   4C F9 B8   JMP $B8F9   dann fertig
BF84   A2 4E      LDX #$4E    Zeiger auf
BF86   A0 00      LDY #$00    Hilfsakku
BF88   20 D4 BB   JSR $BBD4   FAC nach Hilfsakku
BF8B   A5 6E      LDA $6E     Exponent FAC = Potenzexponent
BF8D   10 0F      BPL $BF9E   kleiner eins ?,
BF8F   20 CC BC   JSR $BCCC   dann INT-Funktion
BF92   A9 4E      LDA #$4E    Zeiger auf
BF94   A0 00      LDY #$00    Hilfsakku
BF96   20 5B BC   JSR $BC5B   mit FAC vergleichen
BF99   D0 03      BNE $BF9E   Exponent nicht ganzzahlig,
                              dann zu $BF9E
BF9B   98         TYA         Akku= 4
BF9C   A4 07      LDY $07     Exponentenstelle
BF9E   20 FE BB   JSR $BBFE   ARG nach FAC
BFA1   98         TYA         Exponentenstelle
BFA2   48         PHA         in Stack
BFA3   20 EA B9   JSR $B9EA   LOG-Funktion
BFA6   A9 4E      LDA #$4E    Zeiger auf
BFA8   A0 00      LDY #$00    Hilfsakku
BFAA   20 28 BA   JSR $BA28   mit FAC multiplizieren
BFAD   20 ED BF   JSR $BFED   EXP-Funktion
BFB0   68         PLA         Exponent aus Stack
BFB1   4A         LSR  A      wenn Exponent gradzahlig,
BFB2   90 0A      BCC $BFBE   dann fertig

**************************** Vorzeichenwechsel

Einsprung von $BD67, $E030, $E29D, $E2AA, $E313, $E33A

BFB4   A5 61      LDA $61     Exponent
BFB6   F0 06      BEQ $BFBE   Zahl gleich null, dann fertig
BFB8   A5 66      LDA $66     Vorzeichen
BFBA   49 FF      EOR #$FF    invertieren und
```

```
BFBC   85 66      STA $66     speichern
BFBE   60         RTS         Rücksprung

**************************   Konstanten für   EXP
BFBF   81 38 AA 3B 29        1.44269504  = 1/LOG(2)
BFC4   07                    7 = Polynomgrad, 8
                             Koeffizienten
BFC5   71 34 58 3E 56        2.14987637E-5
BFCA   74 16 7E B3 1B        1.4352314E-4
BFCF   77 2F EE E3 85        1.34226348E-3
BFD4   7A 1D 84 1C 2A        9.614011701E-3
BFD9   7C 63 59 58 0A        .0555051269
BFDE   7E 75 FD E7 C6        .240226385
BFE3   80 31 72 18 10        .693147186
BFE8   81 00 00 00 00        1

**************************   BASIC-Funktion  EXP

Einsprung von $BFAD

BFED   A9 BF      LDA #$BF    Zeiger auf
BFEF   A0 BF      LDY #$BF    Konstante 1/LOG(2)
BFF1   20 28 BA   JSR $BA28   mit FAC multiplizieren
BFF4   A5 70      LDA $70     80 zu Rundungsstelle
BFF6   69 50      ADC #$50    addieren
BFF8   90 03      BCC $BFFD   wenn kleiner als $FF, dann
                              zu $BFFD
BFFA   20 23 BC   JSR $BC23   Mantisse von FAC um
                              eins erhöhen
BFFD   4C 00 E0   JMP $E000   weiter bei $E000

Einsprung von $BFFD

E000   85 56      STA $56     Rundungsstelle
E002   20 0F BC   JSR $BC0F   FAC nach ARG bringen
E005   A5 61      LDA $61     Exponent
E007   C9 88      CMP #$88    Zahl größer 128 ?,
E009   90 03      BCC $E00E   dann zu $E00E
E00B   20 D4 BA   JSR $BAD4   falls positiv 'OVERFLOW'
E00E   20 CC BC   JSR $BCCC   INTEGER-Funktion
```

```
E011   A5 07      LDA $07     ganze Zahl
E013   18         CLC         Zahl
E014   69 81      ADC #$81    gleich
E016   F0 F3      BEQ $E00B   127 ?, dann zu $E00B
E018   38         SEC         ansonsten
E019   E9 01      SBC #$01    subtrahieren
E01B   48         PHA         und in Stack
E01C   A2 05      LDX #$05    FAC
E01E   B5 69      LDA $69,X   und
E020   B4 61      LDY $61,X   ARG
E022   95 61      STA $61,X   ver-
E024   94 69      STY $69,X   tauschen
E026   CA         DEX         Zähler erniedrigen
E027   10 F5      BPL $E01E   schon alle Zeichen?
E029   A5 56      LDA $56     Rundungs-
E02B   85 70      STA $70     stelle
E02D   20 53 B8   JSR $B853   ARG - FAC
E030   20 B4 BF   JSR $BFB4   Vorzeichenwechsel
E033   A9 C4      LDA #$C4    Zeiger auf
E035   A0 BF      LDY #$BF    Polynomkoeffizienten
E037   20 59 E0   JSR $E059   Polynom berechnen
E03A   A9 00      LDA #$00    Vergleichsbyte
E03C   85 6F      STA $6F     löschen
E03E   68         PLA         Zahl aus Stack
E03F   20 B9 BA   JSR $BAB9   Exponenten von
                              FAC und ARG addieren
E042   60         RTS         Rücksprung

****************************** Polynomberechnung
                              y=a1*x+a2*x^3+a3*x^5+...

Einsprung von $BA16, $E2B1, $E328

E043   85 71      STA $71     Zeiger auf
E045   84 72      STY $72     Polynomkoeffizienten
E047   20 CA BB   JSR $BBCA   FAC nach Akku #3 bringen
E04A   A9 57      LDA #$57    Zeiger auf Akku #3
E04C   20 28 BA   JSR $BA28   FAC * Akku #3  (quadrieren)
E04F   20 5D E0   JSR $E05D   Polynomberechnung
E052   A9 57      LDA #$57    Zeiger auf
```

```
E054   A0 00      LDY #$00    Akku #3
E056   4C 28 BA   JMP $BA28   FAC = FAC * Akku #3

****************************** Polynomberechnung
                              y=a0+a1*x+a2*x^2+a3*x^3+...

Einsprung von $E037

E059   85 71      STA $71     Zeiger auf
E05B   84 72      STY $72     Polynomgrad

Einsprung von $E04F

E05D   20 C7 BB   JSR $BBC7   FAC nach Akku #4 bringen
E060   B1 71      LDA ($71),Y Polynomgrad
E062   85 67      STA $67     als Zähler
E064   A4 71      LDY $71     Zeiger für Polynomauswertung
E066   C8         INY         Zeiger erhöhen,
E067   98         TYA         zeigt dann
E068   D0 02      BNE $E06C   auf ersten Koeffizienten
E06A   E6 72      INC $72     Zeiger
E06C   85 71      STA $71     für
E06E   A4 72      LDY $72     Polynomauswertung
E070   20 28 BA   JSR $BA28   FAC = FAC * Konstante
E073   A5 71      LDA $71     Zeiger in
E075   A4 72      LDY $72     (A/Y)
E077   18         CLC         Zeiger
E078   69 05      ADC #$05    um 5 erhöhen - nächste Zahl
E07A   90 01      BCC $E07D   wenn kleiner, dann zu $E07D
E07C   C8         INY         ansonsten erhöhen
E07D   85 71      STA $71     Zeiger für
E07F   84 72      STY $72     Polynomauswertung speichern
E081   20 67 B8   JSR $B867   FAC = FAC + Konstante
E084   A9 5C      LDA #$5C    Zeiger auf
E086   A0 00      LDY #$00    Akku #4
E088   C6 67      DEC $67     Zähler erniedrigen
E08A   D0 E4      BNE $E070   schon alle, nein, dann
                              zu $E070
E08C   60         RTS         Rücksprung
```

```
****************************** Konstanten für  RND
E08D   98 35 44 7A 00         11879546
E092   68 28 B1 46 00         3.92767774E-4

****************************** BASIC-Funktion  RND
E097   20 2B BC    JSR $BC2B  Vorzeichen holen
E09A   30 37       BMI $E0D3  negativ ?, dann zu $E0D3
E09C   D0 20       BNE $E0BE  nicht Null?, dann zu $E0BE
E09E   20 F3 FF    JSR $FFF3  Basis-Adresse CIA holen
E0A1   86 22       STX $22    als Zeiger
E0A3   84 23       STY $23    speichern
E0A5   A0 04       LDY #$04   Zähler setzen
E0A7   B1 22       LDA ($22),Y LOW-Byte Timer A laden
E0A9   85 62       STA $62    und speichern
E0AB   C8          INY        Zähler erhöhen
E0AC   B1 22       LDA ($22),Y HIGH-Byte Timer A laden
E0AE   85 64       STA $64    und speichern
E0B0   A0 08       LDY #$08   Zähler neu setzen
E0B2   B1 22       LDA ($22),Y TOD  1/10 sec laden
E0B4   85 63       STA $63    und speichern
E0B6   C8          INY        Zähler erhöhen
E0B7   B1 22       LDA ($22),Y TOD  sec laden
E0B9   85 65       STA $65    und speichern
E0BB   4C E3 E0    JMP $E0E3  weiter bei $E0E3
E0BE   A9 8B       LDA #$8B   Zeiger auf
E0C0   A0 00       LDY #$00   letzten RND-Wert
E0C2   20 A2 BB    JSR $BBA2  nach FAC holen
E0C5   A9 8D       LDA #$8D   Zeiger auf
E0C7   A0 E0       LDY #$E0   Konstante
E0C9   20 28 BA    JSR $BA28  FAC = FAC * Konstante
E0CC   A9 92       LDA #$92   Zeiger auf
E0CE   A0 E0       LDY #$E0   Konstante
E0D0   20 67 B8    JSR $B867  FAC = FAC + Konstante
E0D3   A6 65       LDX $65    alle
E0D5   A5 62       LDA $62    Stel-
E0D7   85 65       STA $65    len
E0D9   86 62       STX $62    im
E0DB   A6 63       LDX $63    FAC
```

```
E0DD   A5 64      LDA $64     ver-
E0DF   85 63      STA $63     tau-
E0E1   86 64      STX $64     schen

Einsprung von $E0BB

E0E3   A9 00      LDA #$00    Vorzeichen
E0E5   85 66      STA $66     positiv
E0E7   A5 61      LDA $61     Exponent in
E0E9   85 70      STA $70     Rundungsstelle
E0EB   A9 80      LDA #$80    Zufallszahl
E0ED   85 61      STA $61     speichern
E0EF   20 D7 B8   JSR $B8D7   FAC linksbündig machen
E0F2   A2 8B      LDX #$8B    Zeiger auf
E0F4   A0 00      LDY #$00    letzten RND-Wert

Einsprung von $E2C2

E0F6   4C D4 BB   JMP $BBD4   FAC runden und speichern

**************************** Fehlerauswertung nach
                              I/O-Routinen

Einsprung von $E1D1

E0F9   C9 F0      CMP #$F0    RS 232 OPEN oder CLOSE ?
E0FB   D0 07      BNE $E104   nein
E0FD   84 38      STY $38     BASIC-RAM Ende
E0FF   86 37      STX $37     neu setzen
E101   4C 63 A6   JMP $A663   und zum CLR-Befehl
E104   AA         TAX         Fehlernummer nach X
E105   D0 02      BNE $E109   nicht Null ?
E107   A2 1E      LDX #$1E    sonst Nummer für 'BREAK'
E109   4C 37 A4   JMP $A437   Fehlermeldung ausgeben
```

```
****************************** BASIC  BSOUT

Einsprung von $AB47

E10C   20 D2 FF   JSR $FFD2   ein Zeichen ausgeben
E10F   B0 E8      BCS $E0F9   Fehler ?
E111   60         RTS         Rücksprung

****************************** BASIC  BASIN

Einsprung von $A562

E112   20 CF FF   JSR $FFCF   ein Zeichen holen
E115   B0 E2      BCS $E0F9   Fehler ?
E117   60         RTS         Rücksprung

****************************** BASIC  CKOUT

Einsprung von $AA93

E118   20 AD E4   JSR $E4AD   Ausgabegerät setzen
E11B   B0 DC      BCS $E0F9   Fehler ?
E11D   60         RTS         Rücksprung

****************************** BASIC  CHKIN

Einsprung von $AB8F, $ABAF

E11E   20 C6 FF   JSR $FFC6   Eingabegerät setzen
E121   B0 D6      BCS $E0F9   Fehler ?
E123   60         RTS         Rücksprung

****************************** BASIC  GETIN

Einsprung von $AC35

E124   20 E4 FF   JSR $FFE4   ein Zeichen holen
E127   B0 D0      BCS $E0F9   Fehler ?
E129   60         RTS         Rücksprung
```

```
****************************** SYS-Befehl
E12A   20 8A AD   JSR $AD8A   FRMNUM, numerischen
                              Ausdruck holen
E12D   20 F7 B7   JSR $B7F7   in Adressformat wandeln,
                              nach $14/$15
E130   A9 E1      LDA #$E1    Rück-
E132   48         PHA         sprungadresse
E133   A9 46      LDA #$46    auf
E135   48         PHA         Stack
E136   AD 0F 03   LDA $030F   Status,
E139   48         PHA         in Stack
E13A   AD 0C 03   LDA $030C   Akku,
E13D   AE 0D 03   LDX $030D   X-Register und
E140   AC 0E 03   LDY $030E   Y-Register übergeben
E143   28         PLP         Status setzen
E144   6C 14 00   JMP ($0014) Routine aufrufen
E147   08         PHP         Status speichern
E148   8D 0C 03   STA $030C   Akku,
E14B   8E 0D 03   STX $030D   X-Register,
E14E   8C 0E 03   STY $030E   Y-Register und
E151   68         PLA         Status
E152   8D 0F 03   STA $030F   wieder speichern
E155   60         RTS         Rücksprung

****************************** SAVE-Befehl
E156   20 D4 E1   JSR $E1D4   Parameter (Filenamen, Prim.
                              und Sek. Adresse)
E159   A6 2D      LDX $2D     Endadresse gleich
E15B   A4 2E      LDY $2E     BASIC-Rücksprung
E15D   A9 2B      LDA #$2B    Startadresse gleich Zeiger
                              auf BASIC Anfang
E15F   20 D8 FF   JSR $FFD8   Save-Routine
E162   B0 95      BCS $E0F9   Fehler ?
E164   60         RTS         Rücksprung

****************************** VERIFY-Befehl
E165   A9 01      LDA #$01    Verify-
E167   2C         .BYTE $2C   Flag
```

```
****************************** LOAD-Befehl
E168   A9 00      LDA #$00     Load-Flag
E16A   85 0A      STA $0A      speichern
E16C   20 D4 E1   JSR $E1D4    Parameter holen
E16F   A5 0A      LDA $0A      Flag
E171   A6 2B      LDX $2B      Startadresse gleich
E173   A4 2C      LDY $2C      BASIC-Start
E175   20 D5 FF   JSR $FFD5    Load-Routine
E178   B0 57      BCS $E1D1    Fehler ?
E17A   A5 0A      LDA $0A      Load/Verify - Flag
E17C   F0 17      BEQ $E195    Load ?
E17E   A2 1C      LDX #$1C     Offset für 'VERIFY ERROR'
E180   20 B7 FF   JSR $FFB7    Status holen
E183   29 10      AND #$10     Fehler-Bit isolieren
E185   D0 17      BNE $E19E    Statusbit gesetzt, dann
                               Fehler
E187   A5 7A      LDA $7A      muß HIGH-Byte $7B sein
E189   C9 02      CMP #$02     Test auf Direkt-Modus
E18B   F0 07      BEQ $E194    ja, dann fertig
E18D   A9 64      LDA #$64     Zeiger auf
E18F   A0 A3      LDY #$A3     'OK'
E191   4C 1E AB   JMP $AB1E    ausgeben
E194   60         RTS          Rücksprung
E195   20 B7 FF   JSR $FFB7    Status holen
E198   29 BF      AND #$BF     EOF-Bit löschen
E19A   F0 05      BEQ $E1A1    kein Fehler
E19C   A2 1D      LDX #$1D     Offset für 'LOAD ERROR'
E19E   4C 37 A4   JMP $A437    Fehlermeldung ausgeben
E1A1   A5 7B      LDA $7B      Direkt-
E1A3   C9 02      CMP #$02     modus testen
E1A5   D0 0E      BNE $E1B5    nein, dann weiter
E1A7   86 2D      STX $2D      Endadresse gleich
E1A9   84 2E      STY $2E      Rücksprung
E1AB   A9 76      LDA #$76     Zeiger auf
E1AD   A0 A3      LDY #$A3     'READY'
E1AF   20 1E AB   JSR $AB1E    String ausgeben
E1B2   4C 2A A5   JMP $A52A    Programmzeilen neu binden,
                               CLR
E1B5   20 8E A6   JSR $A68E    CHRGET-Zeiger auf
                               Programmstart
```

```
E1B8   20 33 A5   JSR $A533   Programmzeilen neu binden
E1BB   4C 77 A6   JMP $A677   RESTORE, BASIC initialisieren

**************************** BASIC-Befehl OPEN
E1BE   20 19 E2   JSR $E219   Parameter holen
E1C1   20 C0 FF   JSR $FFC0   OPEN-Routine
E1C4   B0 0B      BCS $E1D1   Fehler ?
E1C6   60         RTS         Rücksprung

**************************** BASIC-Befehl CLOSE
E1C7   20 19 E2   JSR $E219   Parameter holen
E1CA   A5 49      LDA $49     Filenummer
E1CC   20 C3 FF   JSR $FFC3   CLOSE-Routine
E1CF   90 C3      BCC $E194   kein Fehler, RTS
E1D1   4C F9 E0   JMP $E0F9   zur Fehlerauswertung

**************************** Parameter für LOAD und SAVE
                              holen

Einsprung von $E156, $E16C

E1D4   A9 00      LDA #$00    Default für Länge des
                              Filenamen
E1D6   20 BD FF   JSR $FFBD   Filenamenparameter setzen
E1D9   A2 01      LDX #$01    Default für Gerätenummer
E1DB   A0 00      LDY #$00    Sekundäradresse
E1DD   20 BA FF   JSR $FFBA   Fileparameter setzen
E1E0   20 06 E2   JSR $E206   weitere Zeichen ?
E1E3   20 57 E2   JSR $E257   Filenamen holen
E1E6   20 06 E2   JSR $E206   weitere Zeichen ?
E1E9   20 00 E2   JSR $E200   Geräteadresse holen
E1EC   A0 00      LDY #$00    Sekundäradresse
E1EE   86 49      STX $49     Geräteadresse
E1F0   20 BA FF   JSR $FFBA   Fileparameter setzen
E1F3   20 06 E2   JSR $E206   weitere Zeichen ?
E1F6   20 00 E2   JSR $E200   Sekundäradresse holen
E1F9   8A         TXA         in Akku schieben
E1FA   A8         TAY         Sekundäradresse
E1FB   A6 49      LDX $49     Gerätenummer
E1FD   4C BA FF   JMP $FFBA   Fileparameter setzen
```

```
******************************

Einsprung von $E1E9, $E1F6, $E231, $E245

E200   20 0E E2   JSR $E20E   prüft auf Komma und weitere
                              Zeichen
E203   4C 9E B7   JMP $B79E   holt Byte-Wert nach X

****************************** prüft auf weitere Zeichen

Einsprung von $E1E0, $E1E6, $E1F3, $E22E, $E242, $E251

E206   20 79 00   JSR $0079   CHRGOT letztes Zeichen
E209   D0 02      BNE $E20D   weiteres Zeichen, dann
                              Rückkehr
E20B   68         PLA         sonst Rückkehr zur
E20C   68         PLA         übergeordneten Routine
E20D   60         RTS         Rücksprung

******************************

Einsprung von $E200, $E254

E20E   20 FD AE   JSR $AEFD   prüft auf Komma

Einsprung von $E21E

E211   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen
E214   D0 F7      BNE $E20D   weitere Zeichen, dann
                              Rückkehr
E216   4C 08 AF   JMP $AF08   'SYNTAX ERROR'

****************************** Parameter für OPEN holen

Einsprung von $E1BE, $E1C7

E219   A9 00      LDA #$00    Default für Länge des
                              Filenamens
E21B   20 BD FF   JSR $FFBD   Filenamenparameter setzen
```

```
E21E   20 11 E2   JSR $E211   weitere Zeichen ?
E221   20 9E B7   JSR $B79E   holt logische Filenummer
                              nach X-Reg
E224   86 49      STX $49     und speichern
E226   8A         TXA         logische Filenummer
E227   A2 01      LDX #$01    Default für Geräteadresse
E229   A0 00      LDY #$00    Sekundäradresse
E22B   20 BA FF   JSR $FFBA   Fileparameter setzen
E22E   20 06 E2   JSR $E206   weitere Zeichen ?
E231   20 00 E2   JSR $E200   holt Geräteadresse
E234   86 4A      STX $4A     und speichern
E236   A0 00      LDY #$00    Sekundäradresse
E238   A5 49      LDA $49     logische Filenummer
E23A   E0 03      CPX #$03    Gerätenummer kleiner 3 ?
E23C   90 01      BCC $E23F   ja
E23E   88         DEY         sonst Sekundäradresse auf
                              255 (keine Sek-Adr)
E23F   20 BA FF   JSR $FFBA   Fileparameter setzen
E242   20 06 E2   JSR $E206   weitere Zeichen ?
E245   20 00 E2   JSR $E200   holt Sekundäradresse
E248   8A         TXA         in Akku schieben
E249   A8         TAY         Sekundäradresse
E24A   A6 4A      LDX $4A     Gerätenummer
E24C   A5 49      LDA $49     logische Filenummer
E24E   20 BA FF   JSR $FFBA   Fileparameter setzen
E251   20 06 E2   JSR $E206   weitere Zeichen ?
E254   20 0E E2   JSR $E20E   prüft auf Komma

Einsprung von $E1E3

E257   20 9E AD   JSR $AD9E   FRMEVL Ausdruck holen
E25A   20 A3 B6   JSR $B6A3   holt Stringparameter, FRESTR
E25D   A6 22      LDX $22     Adresse des
E25F   A4 23      LDY $23     Filenamens
E261   4C BD FF   JMP $FFBD   Filenamenparameter setzen

****************************** BASIC-Funktion  COS
E264   A9 E0      LDA #$E0    Zeiger auf
E266   A0 E2      LDY #$E2    Konstante  Pi/2
E268   20 67 B8   JSR $B867   zu FAC addieren
```

```
****************************** BASIC-Funktion  SIN

Einsprung von $E2BB

E26B   20 0C BC   JSR $BC0C   FAC runden und nach ARG
E26E   A9 E5      LDA #$E5    Zeiger auf
E270   A0 E2      LDY #$E2    Konstante Pi*2
E272   A6 6E      LDX $6E     Vorzeichen von ARG
E274   20 07 BB   JSR $BB07   FAC durch 2*Pi dividieren
E277   20 0C BC   JSR $BC0C   FAC runden und nach ARG
E27A   20 CC BC   JSR $BCCC   INT - Funktion
E27D   A9 00      LDA #$00    Vergleichsbyte
E27F   85 6F      STA $6F     löschen
E281   20 53 B8   JSR $B853   ARG minus FAC
E284   A9 EA      LDA #$EA    Zeiger auf
E286   A0 E2      LDY #$E2    Konstante 0.25
E288   20 50 B8   JSR $B850   0.25 - FAC
E28B   A5 66      LDA $66     Vorzeichen laden
E28D   48         PHA         Vorzeichen in Stack
E28E   10 0D      BPL $E29D   positiv ?
E290   20 49 B8   JSR $B849   FAC + 0.5
E293   A5 66      LDA $66     Vorzeichen
E295   30 09      BMI $E2A0   negativ ?
E297   A5 12      LDA $12     Vorzeichen laden
E299   49 FF      EOR #$FF    und umdrehen
E29B   85 12      STA $12     Vorzeichen speichern

Einsprung von $E2DD

E29D   20 B4 BF   JSR $BFB4   Vorzeichen wechseln
E2A0   A9 EA      LDA #$EA    Zeiger auf
E2A2   A0 E2      LDY #$E2    Konstante 0.25
E2A4   20 67 B8   JSR $B867   FAC + 0.25
E2A7   68         PLA         Vorzeichen holen
E2A8   10 03      BPL $E2AD   positiv ?
E2AA   20 B4 BF   JSR $BFB4   Vorzeichen wechseln
E2AD   A9 EF      LDA #$EF    Zeiger auf
E2AF   A0 E2      LDY #$E2    Polynomkoeffizienten
E2B1   4C 43 E0   JMP $E043   Polynom berechnen
****************************** BASIC-Funktion  TAN
```

```
E2B4   20 CA BB   JSR $BBCA   FAC nach Akku#3
E2B7   A9 00      LDA #$00    Flag
E2B9   85 12      STA $12     setzen
E2BB   20 6B E2   JSR $E26B   SIN berechnen
E2BE   A2 4E      LDX #$4E    Zeiger auf
E2C0   A0 00      LDY #$00    Hilfsakku
E2C2   20 F6 E0   JSR $E0F6   FAC nach Hilfsakku
E2C5   A9 57      LDA #$57    Zeiger auf
E2C7   A0 00      LDY #$00    Akku#3
E2C9   20 A2 BB   JSR $BBA2   Akku#3 nach FAC
E2CC   A9 00      LDA #$00    Vorzeichen
E2CE   85 66      STA $66     löschen
E2D0   A5 12      LDA $12     Flag
E2D2   20 DC E2   JSR $E2DC   COS berechnen
E2D5   A9 4E      LDA #$4E    Zeiger auf
E2D7   A0 00      LDY #$00    Hilfsakku (SIN)
E2D9   4C 0F BB   JMP $BB0F   durch FAC dividieren

Einsprung von $E2D2

E2DC   48         PHA         COS
E2DD   4C 9D E2   JMP $E29D   berechnen

****************************** Konstanten für SIN und COS
E2E0   81 49 0F DA A2         1.57079633   Pi/2
E2E5   83 49 0F DA A2         6.28318531   2*Pi
E2EA   7F 00 00 00 00          .25
E2EF   05                     5 = Polynomgrad, 6
                              Koeffizienten
E2F0   84 E6 1A 2D 1B         -14.3813907
E2F5   86 28 07 FB F8          42.0077971
E2FA   87 99 68 89 01         -76.7041703
E2FF   87 23 35 DF E1          81.6052237
E304   86 A5 5D E7 28         -41.3147021
E309   83 49 0F DA A2           6.28318531   2*Pi
```

```
****************************** BASIC-Funktion  ATN
E30E   A5 66      LDA $66      Vorzeichen
E30F   48         PHA          retten
E311   10 03      BPL $E316    positiv ?
E313   20 B4 BF   JSR $BFB4    Vorzeichen vertauschen
E316   A5 61      LDA $61      Exponent
E318   48         PHA          retten
E319   C9 81      CMP #$81     Zahl mit 1 vergleichen
E31B   90 07      BCC $E324    kleiner ?
E31D   A9 BC      LDA #$BC     Zeiger auf
E31F   A0 B9      LDY #$B9     Konstante 1
E321   20 0F BB   JSR $BB0F    1 durch FAC dividieren
                               (Kehrwert)
E324   A9 3E      LDA #$3E     Zeiger auf
E326   A0 E3      LDY #$E3     Polynomkoeffizienten
E328   20 43 E0   JSR $E043    Polynom berechnen
E32B   68         PLA          Exponent zurückholen
E32C   C9 81      CMP #$81     war Zahl
E32E   90 07      BCC $E337    kleiner 1, dann zu $E337
E330   A9 E0      LDA #$E0     Zeiger auf
E332   A0 E2      LDY #$E2     Konstante Pi/2
E334   20 50 B8   JSR $B850    Pi/2 minus FAC
E337   68         PLA          Vorzeichen holen
E338   10 03      BPL $E33D    positiv ?
E33A   4C B4 BF   JMP $BFB4    Vorzeichen wechseln
E33D   60         RTS          Rücksprung

****************************** Fließkommakonstanten für
                               ATN-Funktion
E33E   0B                      11 = Polynomgrad, dann 12
                               Koeffizienten
E33F   76 B3 83 BD D3          -6.84793912E-04
E344   79 1E F4 A6 F5           4.85094216E-03
E349   7B 83 FC B0 10           -.0161117015
E34E   7C 0C 1F 67 CA            .034209638
E353   7C DE 53 CB C1           -.054279133
E358   7D 14 64 70 4C            .0724571965
E35D   7D B7 EA 51 7A           -.0898019185
E362   7D 63 30 88 7E            .110932413
E367   7E 92 44 99 3A           -.142839808
```

```
E36C   7E 4C CC 91 C7          .19999912
E371   7F AA AA AA 13          -.333333316
E376   81 00 00 00 00          1

****************************** BASIC NMI-Einsprung

Einsprung von $FE6F

E37B   20 CC FF    JSR $FFCC   CLRCH
E37E   A9 00       LDA #$00    Eingabegerät gleich
E380   85 13       STA $13     Tastatur
E382   20 7A A6    JSR $A67A   BASIC initialisieren
E385   58          CLI         Interrupt freigeben

Einsprung von $A714, $A854

E386   A2 80       LDX #$80    Flag für kein Fehler
E388   6C 00 03    JMP ($0300) BASIC Warmstart Vektor
                               JMP $E38B

Einsprung von $A437, $E388

E388   8A          TXA         Fehlernummer in Akku
E38C   30 03       BMI $E391   kein Fehler, dann 'ready.'
E38E   4C 3A A4    JMP $A43A   Fehlermeldung ausgeben
E391   4C 74 A4    JMP $A474   Ready - Modus

****************************** BASIC Kaltstart

Einsprung von $FCFF

E394   20 53 E4    JSR $E453   BASIC-Vektoren setzen
E397   20 BF E3    JSR $E3BF   RAM initialisieren
E39A   20 22 E4    JSR $E422   Einschaltmeldung ausgeben
E39D   A2 FB       LDX #$FB    Stackzeiger
E39F   9A          TXS         setzen
E3A0   D0 E4       BNE $E386   zum Warmstart
```

```
****************************** Kopie der CHRGET-Routine
E3A2   E6 7A      INC $7A     LOW-Byte Zeiger erhöhen
E3A4   D0 02      BNE $E3A8   Zeiger in BASIC-Text erhöhen
E3A6   E6 7B      INC $7B     HIGH-Byte Zeiger erhöhen
E3A8   AD 60 EA   LDA $EA60   BASIC-Adresse laden
E3AB   C9 3A      CMP #$3A    keine Zahl,
E3AD   B0 0A      BCS $E3B9   dann fertig
E3AF   C9 20      CMP #$20    ' ' Leerzeichen überlesen
E3B1   F0 EF      BEQ $E3A2   ja, nächstes Zeichen
E3B3   38         SEC         Test auf
E3B4   E9 30      SBC #$30    Ziffer,
E3B6   38         SEC         dann
E3B7   E9 D0      SBC #$D0    C=1
E3B9   60         RTS         Rücksprung

****************************** Anfangswert für RND-Funktion
E3BA   80 4F C7 52 58         .811635157

****************************** RAM für BASIC initialisieren

Einsprung von $E397

E3BF   A9 4C      LDA #$4C    JMP
E3C1   85 54      STA $54     für Funktionen
E3C3   8D 10 03   STA $0310   für USR-Funktion
E3C6   A9 48      LDA #$48    Zeiger auf
E3C8   A0 B2      LDY #$B2    'ILLEGAL QUANTITY'
E3CA   8D 11 03   STA $0311   als USR-Vektor
E3CD   8C 12 03   STY $0312   speichern
E3D0   A9 91      LDA #$91    Adresse
E3D2   A0 B3      LDY #$B3    $B391
E3D4   85 05      STA $05     als Vektor für
E3D6   84 06      STY $06     Fest-/Fließkomma-Wandlung
E3D8   A9 AA      LDA #$AA    Adresse
E3DA   A0 B1      LDY #$B1    $B1AA
E3DC   85 03      STA $03     als Vektor für
E3DE   84 04      STY $04     Fließ-/Festkomma-Wandlung
E3E0   A2 1C      LDX #$1C    Zähler setzen
E3E2   BD A2 E3   LDA $E3A2,X CHRGET-Routine
E3E5   95 73      STA $73,X   ins
```

```
E3E7  CA        DEX         RAM kopieren
E3E8  10 F8     BPL $E3E2   schon alles?
E3EA  A9 03     LDA #$03    Schrittweise
E3EC  85 53     STA $53     für Garbage Collection
E3EE  A9 00     LDA #$00    FAC-Rundungsbyte
E3F0  85 68     STA $68     löschen
E3F2  85 13     STA $13     Eingabegerät gleich
E3F4  85 18     STA $18     Tastatur
E3F6  A2 01     LDX #$01    Dummys
E3F8  8E FD 01  STX $01FD   für Linkadresse beim
E3FB  8E FC 01  STX $01FC   Zeileneinbau
E3FE  A2 19     LDX #$19    Zeiger für
E400  86 16     STX $16     Stringverwaltung
E402  38        SEC         RAM-
E403  20 9C FF  JSR $FF9C   Start holen
E406  86 2B     STX $2B     als BASIC-Start
E408  84 2C     STY $2C     speichern
E40A  38        SEC         RAM-
E40B  20 99 FF  JSR $FF99   Ende holen
E40E  86 37     STX $37     als
E410  84 38     STY $38     BASIC-
E412  86 33     STX $33     Ende
E414  84 34     STY $34     speichern
E416  A0 00     LDY #$00    $00
E418  98        TYA         an
E419  91 2B     STA ($2B),Y BASIC-Start
E41B  E6 2B     INC $2B     den
E41D  D0 02     BNE $E421   BASIC-
E41F  E6 2C     INC $2C     Start + 1
E421  60        RTS         Programmnde

*****************************

Einsprung von $E39A

E422  A5 2B     LDA $2B     Zeiger auf
E424  A4 2C     LDY $2C     BASIC-RAM Start
E426  20 08 A4  JSR $A408   prüft auf Platz im Speicher
E429  A9 73     LDA #$73    Zeiger auf
E42B  A0 E4     LDY #$E4    Einschaltmeldung
```

```
E42D   20 1E AB   JSR $AB1E   String ausgeben
E430   A5 37      LDA $37     BASIC-
E432   38         SEC         Ende
E433   E5 2B      SBC $2B     minus
E435   AA         TAX         BASIC-Start
E436   A5 38      LDA $38     gleich
E438   E5 2C      SBC $2C     Bytes free
E43A   20 CD BD   JSR $BDCD   Anzahl ausgeben
E43D   A9 60      LDA #$60    Zeiger auf
E43F   A0 E4      LDY #$E4    'BASIC BYTES FREE'
E441   20 1E AB   JSR $AB1E   String ausgeben
E444   4C 44 A6   JMP $A644   zum NEW-Befehl

****************************** Tabelle der BASIC-Vektoren
E447   8B E3 83 A4 7C A5 1A A7
E44F   E4 A7 86 AE

******************************

Einsprung von $E394

E453   A2 0B      LDX #$0B    Die
E455   BD 47 E4   LDA $E447,X BASIC-
E458   9D 00 03   STA $0300,X Vektoren
E45B   CA         DEX         laden
E45C   10 F7      BPL $E455   schon alle?
E45E   60         RTS         Rücksprung

****************************** Betriebssystem

******************************  System-Meldungen
E45F   00 20 42 41 53 49 43 20 basic bytes free
E467   42 59 54 45 53 20 46 52
E46F   45 45 0D 00
E473   93 0D 20 20 20 20 2A 2A (clr)  **** commodore 64 basic v2 ****
E47B   2A 2A 20 43 4F 4D 4D 4F (cr) (cr) 64k ram system
E483   44 4F 52 45 20 36 34 20
E48B   42 41 53 49 43 20 56 32
E493   20 2A 2A 2A 2A 0D 0D 20
E49B   36 34 4B 20 52 41 4D 20
```

```
E4A3   53 59 53 54 45 4D 20 20
E4AB   00

E4AC   5C

****************************** BASIC-CKOUT Routine

Einsprung von $E118

E4AD   48         PHA          Akkuinhalt in Stack
E4AE   20 C9 FF   JSR $FFC9    CKOUT  Ausgabegerät setzen
E4B1   AA         TAX          Fehlernummer nach X
E4B2   68         PLA          Akkuinhalt zurückholen
E4B3   90 01      BCC $E4B6    kein Fehler ?
E4B5   8A         TXA          Fehlernummer wieder in Akku
E4B6   60         RTS          Rücksprung
E4B7   AA  .....
E4D9   .....  AA

****************************** Hintergrundfarbe setzen

Einsprung von $EA07

E4DA   AD 21 D0   LDA $D021    Farbe holen
E4DD   91 F3      STA ($F3),Y ins Farbram schreiben
E4DF   60         RTS          Rücksprung

****************************** wartet auf Commodore-Taste

Einsprung von $F763

E4E0   69 02      ADC #$02     2*256/60 = 8.5 Sekunden
                               warten
E4E2   A4 91      LDY $91      Flag testen
E4E4   C8         INY          und erhöhen
E4E5   D0 04      BNE $E4EB    Taste gedrückt ?
E4E7   C5 A1      CMP $A1      Zeit noch nicht um ?,
E4E9   D0 F7      BNE $E4E2    dann warten
E4EB   60         RTS          Rücksprung
```

```
****************************** Timerkonstanten für RS 232
                               Baud Rate, PAL-Version

E4EC   19 26        $2619  = 9753     50 Baud
E4EE   44 19        $1944  = 6468     75 Baud
E4F0   1A 11        $111A  = 4378    110 Baud
E4F2   E8 0D        $0DE8  = 3560    134.5 Baud
E4F4   70 0C        $0C70  = 3184    150 Baud
E4F6   06 06        $0606  = 1542    300 Baud
E4F8   D1 02        $02D1  =  736    600 Baud
E4FA   37 01        $0137  =  311   1200 Baud
E4FC   AE 00        $00AE  =  174   1800 Baud
E4FE   69 00        $0069  =  105   2400 Baud

****************************** Basis-Adresse des CIAs holen

Einsprung von $FFF3

E500   A2 00      LDX #$00    Adresse
E502   A0 DC      LDY #$DC    $DC00
E504   60         RTS         Rücksprung

****************************** holt Anzahl der Zeilen und
                               Spalten

Einsprung von $FFED

E505   A2 28      LDX #$28    40 Spalten
E507   A0 19      LDY #$19    25 Zeilen
E509   60         RTS         Rücksprung

****************************** Cursor setzen (C=0) / holen
                               (C=1)

Einsprung von $FFF0

E50A   B0 07      BCS $E513   Carry gesetzt, dann zu $E513
E50C   86 D6      STX $D6     Zeile
E50E   84 D3      STY $D3     Spalte
E510   20 6C E5   JSR $E56C   Cursor setzen
```

```
E513   A6 D6      LDX $D6     Zeile
E515   A4 D3      LDY $D3     Spalte
E517   60         RTS         Rücksprung

****************************** Bildschirm Reset

Einsprung von $FE6C, $FF5B

E518   20 A0 E5   JSR $E5A0   Videocontroller
                              initialisieren
E51B   A9 00      LDA #$00    Shift-
E51D   8D 91 02   STA $0291   Commodore ermöglichen
E520   85 CF      STA $CF     Cursor nicht in Blinkphase
E522   A9 48      LDA #$48    Adresse
E524   8D 8F 02   STA $028F   ($028F) =  $EB48
E527   A9 EB      LDA #$EB    setzen
E529   8D 90 02   STA $0290   = Zeiger auf Adressen für
                              Tastaturdekodierung
E52C   A9 0A      LDA #$0A    10
E52E   8D 89 02   STA $0289   max. Länge des
                              Tastaturpuffers
E531   8D 8C 02   STA $028C   Zähler für
                              Repeat-Geschwindigkeit
E534   A9 0E      LDA #$0E    hellblau
E536   8D 86 02   STA $0286   Augenblickliche Farbe
E539   A9 04      LDA #$04    Repeat-
E53B   8D 8B 02   STA $028B   Geschwindigkeit
E53E   A9 0C      LDA #$0C    Cursor
E540   85 CD      STA $CD     Blinkzeit
E542   85 CC      STA $CC     Cursor Blinkflag

****************************** Bildschirm löschen

Einsprung von $E86E

E544   AD 88 02   LDA $0288   Speicherseite für
                              Bildschirm-RAM
E547   09 80      ORA #$80    Adressen
E549   A8         TAY         der
E54A   A9 00      LDA #$00    Bild-
```

```
E54C   AA          TAX          schirm-
E54D   94 D9       STY $D9,X    zeilen
E54F   18          CLC          40 addieren
E550   69 28       ADC #$28     (eine Zeile)
E552   90 01       BCC $E555    kein Übertrag, dann
                                HIGH-Byte nicht erhöhen
E554   C8          INY          HIGH-Byte erhöhen
E555   E8          INX          LOW-Byte erhöhen
E556   E0 1A       CPX #$1A     26, alle Zeilen ?
E558   D0 F3       BNE $E54D    nein, dann weiter
E55A   A9 FF       LDA #$FF     Kennzeichnung der
E55C   95 D9       STA $D9,X    26, Zeile
E55E   A2 18       LDX #$18     24, Anzahl der Zeilen minus 1
E560   20 FF E9    JSR $E9FF    Bildschirmzeile löschen
E563   CA          DEX          Zähler erniedrigen
E564   10 FA       BPL $E560    schon alle?

****************************** Cursor Home

Einsprung von $E59D, $E78F

E566   A0 00       LDY #$00     Löschen der
E568   84 D3       STY $D3      Cursorspalte und
E56A   84 D6       STY $D6      Cursorzeile

****************************** Cursorpos. berechnen,
                                Bildschirmzeiger setzen

Einsprung von $E510, $E70E, $E847, $E88E

E56C   A6 D6       LDX $D6      Cursorzeile
E56E   A5 D3       LDA $D3      Cursorspalte
E570   B4 D9       LDY $D9,X    HIGH-Bytes für Doppelzeilen
E572   30 08       BMI $E57C    einfache Zeile, dann zu $E57C
E574   18          CLC          Spalte
E575   69 28       ADC #$28     +40
E577   85 D3       STA $D3      und speichern
E579   CA          DEX          nächste Zeile
E57A   10 F4       BPL $E570    schon alle?
E57C   20 F0 E9    JSR $E9F0    Zeiger auf Video-RAM setzen
```

```
E57F   A9 27      LDA #$27    39 Spalten
E581   E8         INX         Zeiger auf Bildschirmtabelle
                               erhöhen
E582   B4 D9      LDY $D9,X   HIGH-Byte Startadresse der
                               Zeile in Y-REG schreiben
E584   30 06      BMI $E58C   Verzweige falls größer,
                               gleich 128
E586   18         CLC         Cursor eine Zeile
E587   69 28      ADC #$28    tiefer setzen (+40 Spalten)
E589   E8         INX         Zeiger auf Bildschirmtabelle
                               erhöhen
E58A   10 F6      BPL $E582   unbedingter Sprung
E58C   85 D5      STA $D5     Zeilenlänge speichern
E58E   4C 24 EA   JMP $EA24   Zeiger auf Farb-RAM berechnen
                                Rücksprung

Einsprung von $E621

E591   E4 C9      CPX $C9     wenn Cursorzeile
E593   F0 03      BEQ $E598   gleich null, dann Rücksprung
E595   4C ED E6   JMP $E6ED   Adresse für zugehörige
                               Zeilennummer nach $D1/$D2
E598   60         RTS         Rücksprung
E599   EA         NOP         no operation

****************************
E59A   20 A0 E5   JSR $E5A0   Videocontroller
                               initialisieren
E59D   4C 66 E5   JMP $E566   Cursor Home

**************************** Videocontroller
                               initialisieren

Einsprung von $E518, $E59A

E5A0   A9 03      LDA #$03    Ausgabe auf
E5A2   85 9A      STA $9A     Bildschirm
E5A4   A9 00      LDA #$00    Eingabe von
E5A6   85 99      STA $99     Tastatur
E5A8   A2 2F      LDX #$2F    47
```

```
E5AA   BD B8 EC   LDA $ECB8,X Konstanten
E5AD   9D FF CF   STA $CFFF,X in Videokontroller schreiben
E5B0   CA         DEX         Zähler erniedrigen
E5B1   D0 F7      BNE $E5AA   schon alle?
E5B3   60         RTS         Rücksprung

****************************** Zeichen aus Tastaturpuffer
                               holen

Einsprung von $E5E7, $F147

E5B4   AC 77 02   LDY $0277   erstes Zeichen holen
E5B7   A2 00      LDX #$00    Zähler auf Null
E5B9   BD 78 02   LDA $0278,X Puffer nach
E5BC   9D 77 02   STA $0277,X vorne aufrücken
E5BF   E8         INX         Zähler erhöhen
E5C0   E4 C6      CPX $C6     mit Anzahl der
E5C2   D0 F5      BNE $E5B9   Zeichen vergleichen
E5C4   C6 C6      DEC $C6     Zeichenzahl erniedrigen
E5C6   98         TYA         Zeichen in Akku holen
E5C7   58         CLI         Interrupt freigeben
E5C8   18         CLC         Carry löschen
E5C9   60         RTS         Rücksprung

****************************** Warteschleife für
                               Tastatureingabe
E5CA   20 16 E7   JSR $E716   Zeichen auf Bildschirm
                               ausgeben
E5CD   A5 C6      LDA $C6     Anzahl der
E5CF   85 CC      STA $CC     gedrückten
E5D1   8D 92 02   STA $0292   Tasten
E5D4   F0 F7      BEQ $E5CD   keine Taste gedrückt ?,
                               dann warten
E5D6   78         SEI         Interrupt verhindern
E5D7   A5 CF      LDA $CF     Cursor in Blink-Phase ?
E5D9   F0 0C      BEQ $E5E7   nein
E5DB   A5 CE      LDA $CE     Zeichen unter dem Cursor
E5DD   AE 87 02   LDX $0287   Farbe unter dem Cursor
E5E0   A0 00      LDY #$00    Cursor nicht
E5E2   84 CF      STY $CF     in Blinkphase
```

```
E5E4  20 13 EA   JSR $EA13    Zeichen und Farbe setzen
E5E7  20 B4 E5   JSR $E5B4    Zeichen aus Tastaturpuffer
                              holen
E5EA  C9 83      CMP #$83     Kode für
E5EC  D0 10      BNE $E5FE    'SHIFT RUN' ?
E5EE  A2 09      LDX #$09     9 Zeichen
E5F0  78         SEI          Interrupt verhindern
E5F1  86 C6      STX $C6      Zeichenzahl merken
E5F3  BD E6 EC   LDA $ECE6,X  'LOAD (cr) RUN (cr)'
E5F6  9D 76 02   STA $0276,X  in Tastaturpuffer holen
E5F9  CA         DEX          nächstes Zeichen
E5FA  D0 F7      BNE $E5F3    schon alle ?
E5FC  F0 CF      BEQ $E5CD    und auswerten
E5FE  C9 0D      CMP #$0D     'CR'
E600  D0 C8      BNE $E5CA    nein ?, dann zurück zur
                              Warteschleife
E602  A4 D5      LDY $D5      Länge der Bildschirmzeile
E604  84 D0      STY $D0      CR-Flag setzen
E606  B1 D1      LDA ($D1),Y  Zeichen vom Bildschirm holen
E608  C9 20      CMP #$20     Leerzeichen
E60A  D0 03      BNE $E60F    am Ende
E60C  88         DEY          der
E60D  D0 F7      BNE $E606    Zeile
E60F  C8         INY          eliminieren
E610  84 C8      STY $C8      Position als Index merken
E612  A0 00      LDY #$00     Cursorspalte
E614  8C 92 02   STY $0292    gleich Null
E617  84 D3      STY $D3      Cursorposition auf Null
E619  84 D4      STY $D4      Hochkommaflag löschen
E61B  A5 C9      LDA $C9      wenn Cursorzeile schon durch
E61D  30 1B      BMI $E63A    scrollen verschwunden, dann
                              zu $E63A
E61F  A6 D6      LDX $D6      Cursorzeile
E621  20 ED E6   JSR $E591    Adresse für Startzeile setzen
E624  E4 C9      CPX $C9      Fehler bei Eingabe ?,
E626  D0 12      BNE $E63A    dann nochmal lesen
E628  A5 CA      LDA $CA      letzte Spalte
E62A  85 D3      STA $D3      in Spaltenzeiger bringen
E62C  C5 C8      CMP $C8      mit Index vergleichen
```

```
E62E   90 0A      BCC $E63A    wenn kleiner, dann Zeile
                               auswerten
E630   B0 2B      BCS $E65D    wenn größer oder gleich, dann
                               keine Eingabe

****************************** Ein Zeichen vom Bildschirm
                               holen

Einsprung von $F163, $F170

E632   98         TYA          die
E633   48         PHA          Re-
E634   8A         TXA          gister
E635   48         PHA          retten
E636   A5 D0      LDA $D0      CR-Flag
E638   F0 93      BEQ $E5CD    nein, dann zur Warteschleife
E63A   A4 D3      LDY $D3      Spalte
E63C   B1 D1      LDA ($D1),Y  Zeichen vom Bildschirm holen
E63E   85 D7      STA $D7      und
E640   29 3F      AND #$3F     nach
E642   06 D7      ASL $D7      ASCII
E644   24 D7      BIT $D7      wandeln
E646   10 02      BPL $E64A    wenn Bit 6 nicht gesetzt,
                               dann zu $E64A
E648   09 80      ORA #$80     Bit 7 setzen
E64A   90 04      BCC $E650    Zeichen nicht revers ?, dann
                               zu $E650
E64C   A6 D4      LDX $D4      Hochkommaflag nicht
E64E   D0 04      BNE $E654    gesetzt ?, dann zu $E654
E650   70 02      BVS $E654    wenn ja, dann zu $E654
E652   09 40      ORA #$40     Bit 6 im Zeichen setzen
E654   E6 D3      INC $D3      Cursor eins weiter setzen
E656   20 84 E6   JSR $E684    auf Hochkomma testen
E659   C4 C8      CPY $C8      Cursor in letzter Spalte ?
E65B   D0 17      BNE $E674    wenn nicht, dann zu $E674
E65D   A9 00      LDA #$00     Zeile
E65F   85 D0      STA $D0      vollständig gelesen
E661   A9 0D      LDA #$0D     'CR'
E663   A6 99      LDX $99      ans Ende der Zeile setzen
E665   E0 03      CPX #$03     Eingabe vom Bildschirm ?
```

```
E667   F0 06      BEQ $E66F   ja, dann zu $E66F
E669   A6 9A      LDX $9A     Ausgabe auf Bildschirm
E66B   E0 03      CPX #$03    ja, dann
E66D   F0 03      BEQ $E672   zu $E672
E66F   20 16 E7   JSR $E716   Zeichen auf Bildschirm
                              schreiben
E672   A9 0D      LDA #$0D    Wert für
E674   85 D7      STA $D7     'CR'
E676   68         PLA         die
E677   AA         TAX         Register
E678   68         PLA         zürück-
E679   A8         TAY         holen
E67A   A5 D7      LDA $D7     Bildschirm-Kode
E67C   C9 DE      CMP #$DE    mit Kode für Pi vergleichen
E67E   D0 02      BNE $E682   nein ?, dann fertig
E680   A9 FF      LDA #$FF    ja ?, durch BASIC-Kode
                              für Pi ersetzen
E682   18         CLC         Carry löschen
E683   60         RTS         Rücksprung

****************************** auf Hochkomma testen

Einsprung von $E656, $E73F

E684   C9 22      CMP #$22    '"' ?
E686   D0 08      BNE $E690   nein ?, dann fertig
E688   A5 D4      LDA $D4     Hochkomma-
E68A   49 01      EOR #$01    Flag
E68C   85 D4      STA $D4     umdrehen
E68E   A9 22      LDA #$22    Hochkomma-Code wieder-
                              herstellen
E690   60         RTS         Rücksprung

****************************** Zeichen auf Bildschirm
                              ausgeben

Einsprung von $E7E0

E691   09 40      ORA #$40    Bit 6 im Zeichen setzen
```

```
Einsprung von $E742

E693   A6 C7      LDX $C7     RVS ?
E695   F0 02      BEQ $E699   Umwandlung in Bildschirmcode

Einsprung von $E749, $E782, $E82F

E697   09 80      ORA #$80    ja, dann Bit 7 setzen
E699   A6 D8      LDX $D8     wenn Einfügzähler Null,
E69B   F0 02      BEQ $E69F   dann zu $E69F
E69D   C6 D8      DEC $D8     Zähler erniedrigen
E69F   AE 86 02   LDX $0286   Farbkode
E6A2   20 13 EA   JSR $EA13   Zeichen in Bildschirm-RAM
                              schreiben
E6A5   20 B6 E6   JSR $E6B6   Tabelle der Zeilenanfänge
                              aktualisieren

Einsprung von $E7AA, $E7CB, $E826, $E861, $E867, $E871, $E89E,
              $EC5B, $EC75

E6A8   68         PLA        Y-Reg
E6A9   A8         TAY        aus Stack
E6AA   A5 D8      LDA $D8    wenn Einfügzähler Null,
E6AC   F0 02      BEQ $E6B0  dann zu $E6B0
E6AE   46 D4      LSR $D4    Hochkommamodus löschen
E6B0   68         PLA        X-Reg
E6B1   AA         TAX        aus Stack
E6B2   68         PLA        Akku aus Stack
E6B3   18         CLC        Carry löschen
E6B4   58         CLI        Interrupt freigeben
E6B5   60         RTS        Rücksprung

****************************** HIGH-Byte für Zeilenanfänge
                              neu berechnen

Einsprung von $E6A6

E6B6   20 B3 E8   JSR $E8B3   Zeilenzeiger erhöhen
E6B9   E6 D3      INC $D3     Cursorspalte erhöhen
E6BB   A5 D5      LDA $D5     Zeilenlänge holen
```

```
E6BD  C5 D3     CMP $D3     Vergleich mit Cursorspalte
E6BF  B0 3F     BCS $E700   nicht überschritten, dann RTS
E6C1  C9 4F     CMP #$4F    79 Zeichen (Doppelzeile) ?
E6C3  F0 32     BEQ $E6F7   wenn ja, dann zu $E6F7
E6C5  AD 92 02  LDA $0292   Zeilenübergang nicht
E6C8  F0 03     BEQ $E6CD   im Editmodus, dann zu $E6CD
E6CA  4C 67 E9  JMP $E967   neue Zeile einfügen
E6CD  A6 D6     LDX $D6     Zeile
E6CF  E0 19     CPX #$19    25 ?
E6D1  90 07     BCC $E6DA   wenn ja, dann zu $E6DA
E6D3  20 EA E8  JSR $E8EA   SCROLL
E6D6  C6 D6     DEC $D6     Cursorzeilenzeiger
                            erniedrigen
E6D8  A6 D6     LDX $D6     Zähler holen

Einsprung von $E97E, $E9C2

E6DA  16 D9     ASL $D9,X   Zeile
E6DC  56 D9     LSR $D9,X   markieren
E6DE  E8        INX         Zähler erhöhen
E6DF  B5 D9     LDA $D9,X   Startzeile
E6E1  09 80     ORA #$80    markieren
E6E3  95 D9     STA $D9,X   und speichern
E6E5  CA        DEX         Zähler erniedrigen
E6E6  A5 D5     LDA $D5     Zeilenlänge
E6E8  18        CLC         mit
E6E9  69 28     ADC #$28    40 addieren
E6EB  85 D5     STA $D5     und speichern

Einsprung von $E595

E6ED  B5 D9     LDA $D9,X   keine Doppelzeile,
E6EF  30 03     BMI $E6F4   dann zu $E6F4
E6F1  CA        DEX         Zähler erniedrigen
E6F2  D0 F9     BNE $E6ED   noch nicht alle?, dann weiter
E6F4  4C F0 E9  JMP $E9F0   Zeiger auf Farb-RAM
                            für Zeile X
E6F7  C6 D6     DEC $D6     Cursorzeile erniedrigen
E6F9  20 7C E8  JSR $E87C   und initialisieren
E6FC  A9 00     LDA #$00    Spalte
```

```
E6FE   85 D3      STA $D3     auf Null
E700   60         RTS         Rücksprung

****************************** Rückschritt in vorhergehende
                               Zeile

Einsprung von $E753, $E864

E701   A6 D6      LDX $D6     Cursorzeile
E703   D0 06      BNE $E70B   wenn null, dann zu $E70B
E705   86 D3      STX $D3     Cursorspalte
E707   68         PLA         Sprungadresse
E708   68         PLA         aus Stack holen
E709   D0 9D      BNE $E6A8   unbedingter Sprung
E70B   CA         DEX         Zeilennummer
E70C   86 D6      STX $D6     erniedrigen
E70E   20 6C E5   JSR $E56C   Cursorposition berechnen
E711   A4 D5      LDY $D5     Zeilenlänge
E713   84 D3      STY $D3     speichern
E715   60         RTS         Rücksprung

****************************** Ausgabe auf Bildschirm

Einsprung von $E5CA, $E66F, $F1D2

E716   48         PHA         Zeichen
E717   85 D7      STA $D7     merken
E719   8A         TXA         die
E71A   48         PHA         Re-
E71B   98         TYA         gister
E71C   48         PHA         retten
E71D   A9 00      LDA #$00    Eingabeflag
E71F   85 D0      STA $D0     löschen
E721   A4 D3      LDY $D3     Cursorspalte
E723   A5 D7      LDA $D7     Zeichen
E725   10 03      BPL $E72A   wenn kleiner 128, dann
                               zu $E72A
E727   4C D4 E7   JMP $E7D4   Zeichen größer $7F behandeln
E72A   C9 0D      CMP #$0D    'CARRIAGE RETURN' ?
E72C   D0 03      BNE $E731   wenn nicht, dann zu $E731
```

```
E72E  4C 91 E8   JMP $E891    Return ausgeben
E731  C9 20      CMP #$20     ' '
E733  90 10      BCC $E745    druckendes Zeichen ?
E735  C9 60      CMP #$60     Zahl kleiner $60,
E737  90 04      BCC $E73D    dann keine Graphikzeichen
E739  29 DF      AND #$DF     Umwandlung in BS-Kode
E73B  D0 02      BNE $E73F    unbedingter Sprung
E73D  29 3F      AND #$3F     Umwandlung in BS-Kode
E73F  20 84 E6   JSR $E684    Test auf Hochkomma
E742  4C 93 E6   JMP $E693    zur Ausgabe, ASCII-Kode
                              in BS-Code
E745  A6 D8      LDX $D8      wenn Einfügzähler =0,
E747  F0 03      BEQ $E74C    dann zu $E74C
E749  4C 97 E6   JMP $E697    ASC II-Kode in BS-Code
E74C  C9 14      CMP #$14     nicht 'DEL' ?,
E74E  D0 2E      BNE $E77E    dann zu $E77E
E750  98         TYA          erste Spalte =0
E751  D0 06      BNE $E759    dann zu $E759
E753  20 01 E7   JSR $E701    zurück in vorherige Zeile
E756  4C 73 E7   JMP $E773    Zeichen in Cursorposition
                              eliminieren
E759  20 A1 E8   JSR $E8A1    Rückschritt prüfen
E75C  88         DEY          Zeiger erniedrigen
E75D  84 D3      STY $D3      und speichern
E75F  20 24 EA   JSR $EA24    Zeiger auf Farb-RAM berechnen
E762  C8         INY          Zeiger erhöhen
E763  B1 D1      LDA ($D1),Y  Zeichen vom Bildschirm
E765  88         DEY          Zeiger erniedrigen
E766  91 D1      STA ($D1),Y  eins nach links schieben
E768  C8         INY          Zeiger erhöhen
E769  B1 F3      LDA ($F3),Y  Farbe
E76B  88         DEY          Zeiger erniedrigen
E76C  91 F3      STA ($F3),Y  eins nach links schieben
E76E  C8         INY          Zeiger erhöhen
E76F  C4 D5      CPY $D5      Endspalte nicht
E771  D0 EF      BNE $E762    erreicht, dann weiter
```

Einsprung von $E756

```
E773  A9 20     LDA #$20     Blank
E775  91 D1     STA ($D1),Y  einfügen
E777  AD 86 02  LDA $0286    Farbcode
E77A  91 F3     STA ($F3),Y  setzen
E77C  10 4D     BPL $E7CB    fertig
E77E  A6 D4     LDX $D4      Hochkomma-Modus ?
E780  F0 03     BEQ $E785    nein
E782  4C 97 E6  JMP $E697    Zeichen revers ausgeben
E785  C9 12     CMP #$12     'RVS ON' ?
E787  D0 02     BNE $E78B    nein, dann
E789  85 C7     STA $C7      Flag für RVS setzen
E78B  C9 13     CMP #$13     'HOME' ?
E78D  D0 03     BNE $E792    nein
E78F  20 66 E5  JSR $E566    ja, Cursor Home
E792  C9 1D     CMP #$1D     'Cursor right' ?
E794  D0 17     BNE $E7AD    nein
E796  C8        INY          Zeiger erhöhen
E797  20 B3 E8  JSR $E8B3    Cursorposition prüfen
E79A  84 D3     STY $D3      neuer Zeiger
E79C  88        DEY          Zeiger erniedrigen
E79D  C4 D5     CPY $D5      keine neue Zeile ?,
E79F  90 09     BCC $E7AA    dann fertig
E7A1  C6 D6     DEC $D6      Zeiger erniedrigen
E7A3  20 7C E8  JSR $E87C    Zeile initialisieren
E7A6  A0 00     LDY #$00     Spalte
E7A8  84 D3     STY $D3      gleich null
E7AA  4C A8 E6  JMP $E6A8    fertig
E7AD  C9 11     CMP #$11     'Cursor down' ?
E7AF  D0 1D     BNE $E7CE    nein
E7B1  18        CLC          plus
E7B2  98        TYA          40,
E7B3  69 28     ADC #$28     eine Zeile
E7B5  A8        TAY          tiefer
E7B6  E6 D6     INC $D6      Zeiger erhöhen
E7B8  C5 D5     CMP $D5      neue Zeile erreicht?
E7BA  90 EC     BCC $E7A8    nein, dann zu $E7A8
E7BC  F0 EA     BEQ $E7A8    ja, dann zu $E7A8
E7BE  C6 D6     DEC $D6      Zeiger erniedrigen
```

```
E7C0   E9 28      SBC #$28    40 abziehen
E7C2   90 04      BCC $E7C8   genügend abgezogen, dann
                              zu $E7C8
E7C4   85 D3      STA $D3     Spalte setzen
E7C6   D0 F8      BNE $E7C0   noch mal
E7C8   20 7C E8   JSR $E87C   Zeile initialisieren
E7CB   4C A8 E6   JMP $E6A8   fertig
E7CE   20 CB E8   JSR $E8CB   prüft auf Farbcodes
E7D1   4C 44 EC   JMP $EC44   Test auf weitere
                              Sonderzeichen

****************************** Zeichen größer $127

Einsprung von $E727

E7D4   29 7F      AND #$7F    Kode größer 127,
                              Bit 7 löschen
E7D6   C9 7F      CMP #$7F    nicht 'Pi' ?
E7D8   D0 02      BNE $E7DC   dann zu $E7DC
E7DA   A9 5E      LDA #$5E    Bildschirmkode für Pi
E7DC   C9 20      CMP #$20    Steuerzeichen ?
E7DE   90 03      BCC $E7E3   ja
E7E0   4C 91 E6   JMP $E691   druckendes Zeichen ausgeben
E7E3   C9 0D      CMP #$0D    nicht 'Shift return' ?
E7E5   D0 03      BNE $E7EA   dann zu $E7EA
E7E7   4C 91 E8   JMP $E891   neue Zeile
E7EA   A6 D4      LDX $D4     Hochkomma-Modus ?
E7EC   D0 3F      BNE $E82D   ja, Steuerzeichen revers
                              ausgeben
E7EE   C9 14      CMP #$14    nicht 'INS' ?,
E7F0   D0 37      BNE $E829   dann zu $E829
E7F2   A4 D5      LDY $D5     Zeilenlänge
E7F4   B1 D1      LDA ($D1),Y letztes Zeichen in Zeile
E7F6   C9 20      CMP #$20    gleich Leerzeichen ?
E7F8   D0 04      BNE $E7FE   nein, dann zu $E7FE
E7FA   C4 D3      CPY $D3     Cursor in letzter Spalte ?
E7FC   D0 07      BNE $E805   nein, dann zu $E805
E7FE   C0 4F      CPY #$4F    79 ? maximale Zeilenlänge
E800   F0 24      BEQ $E826   letzte Spalte, dann keine
                              Aktion
```

```
E802  20 65 E9   JSR $E965    Leerzeile einfügen
E805  A4 D5      LDY $D5      Zeilenlänge
E807  20 24 EA   JSR $EA24    Zeiger auf Farbram berechnen
E80A  88         DEY          Zeiger erniedrigen
E80B  B1 D1      LDA ($D1),Y Zeichen vom Bildschirm
E80D  C8         INY          Zeiger erhöhen
E80E  91 D1      STA ($D1),Y eins nach rechts schieben
E810  88         DEY          Zeiger erniedrigen
E811  B1 F3      LDA ($F3),Y und Farbe
E813  C8         INY          Zeiger erhöhen
E814  91 F3      STA ($F3),Y verschieben
E816  88         DEY          Zeiger erniedrigen
E817  C4 D3      CPY $D3      bis zur aktuellen Position
                              aufrücken
E819  D0 EF      BNE $E80A    nicht ?, dann weiter
E81B  A9 20      LDA #$20     Leerzeichen
E81D  91 D1      STA ($D1),Y an augenblickliche Position
                              schreiben
E81F  AD 86 02   LDA $0286    Farbe
E822  91 F3      STA ($F3),Y setzen
E824  E6 D8      INC $D8      Anzahl der Inserts erhöhen
E826  4C A8 E6   JMP $E6A8    Ende der Zeichenausgabe
E829  A6 D8      LDX $D8      Zähler Null?
E82B  F0 05      BEQ $E832    dann zu $E832
E82D  09 40      ORA #$40     Bit 6 setzen
E82F  4C 97 E6   JMP $E697    und Zeichen ausgeben

******************************
E832  C9 11      CMP #$11     nicht Cursor up ?,
E834  D0 16      BNE $E84C    dann zu $E84C
E836  A6 D6      LDX $D6      Zeile
E838  F0 37      BEQ $E871    null, dann fertig
E83A  C6 D6      DEC $D6      Zeilennummer um eins erniedrigen
E83C  A5 D3      LDA $D3      Spalte
E83E  38         SEC          40
E83F  E9 28      SBC #$28     abziehen
E841  90 04      BCC $E847    nicht in Doppelzeile ?,
                              dann zu $E847
E843  85 D3      STA $D3      Cursorspalte
E845  10 2A      BPL $E871    positiv, ok
```

```
E847   20 6C E5   JSR $E56C   Bildschirmzeiger neu setzen
E84A   D0 25      BNE $E871   unbedingter Sprung
E84C   C9 12      CMP #$12    nicht 'RVS OFF' ?,
E84E   D0 04      BNE $E854   dann zu $E854
E850   A9 00      LDA #$00    RVS-Flag
E852   85 C7      STA $C7     löschen
E854   C9 1D      CMP #$1D    nicht  'Cursor left' ?,
E856   D0 12      BNE $E86A   dann zu $E86A
E858   98         TYA         wenn erste Spalte,
E859   F0 09      BEQ $E864   dann zu $E864
E85B   20 A1 E8   JSR $E8A1   Cursorzeile erniedrigen
E85E   88         DEY         Zähler erniedrigen
E85F   84 D3      STY $D3     Cursorspalte
E861   4C A8 E6   JMP $E6A8   fertig
E864   20 01 E7   JSR $E701   Rückschritt in vorherige
                              Zeile
E867   4C A8 E6   JMP $E6A8   fertig
E86A   C9 13      CMP #$13    nicht 'CLR SCREEN' ?,
E86C   D0 06      BNE $E874   dann zu $E874
E86E   20 44 E5   JSR $E544   Bildschirm löschen
E871   4C A8 E6   JMP $E6A8   fertig
E874   09 80      ORA #$80    Bit 7 wiederherstellen
E876   20 CB E8   JSR $E8CB   auf Farbcode prüfen
E879   4C 4F EC   JMP $EC4F   prüft auf Umschaltung
                              Text/Grafik

Einsprung von $E6F9, $E7A3, $E7C8, $E89B

E87C   46 C9      LSR $C9     Flag für Zeilenwechsel
E87E   A6 D6      LDX $D6     Cursorzeilenzeiger
E880   E8         INX         Zeiger erhöhen
E881   E0 19      CPX #$19    noch nicht letzte Zeile ?,
E883   D0 03      BNE $E888   dann zu $E888
E885   20 EA E8   JSR $E8EA   Bildschirm scrollen
E888   B5 D9      LDA $D9,X   nächste Zeile, dann
E88A   10 F4      BPL $E880   wieder scrollen
E88C   86 D6      STX $D6     neue Zeile
E88E   4C 6C E5   JMP $E56C   Cursorposition berechnen
```

```
Einsprung von $E72E, $E7E7

E891   A2 00      LDX #$00    Einfüg-
E893   86 D8      STX $D8     zähler löschen
E895   86 C7      STX $C7     Flag für RVS löschen
E897   86 D4      STX $D4     Quote-Modus löschen
E899   86 D3      STX $D3     Cursor in erste Spalte
E89B   20 7C E8   JSR $E87C   Zeile initialisieren
E89E   4C A8 E6   JMP $E6A8   fertig

Einsprung von $E759, $E85B

E8A1   A2 02      LDX #$02    maximale Zeilenanzahl
E8A3   A9 00      LDA #$00    wenn Cursorspalte
E8A5   C5 D3      CMP $D3     gleich Akku,
E8A7   F0 07      BEQ $E8B0   dann zu $E8B0
E8A9   18         CLC         40 addieren,
E8AA   69 28      ADC #$28    eine Zeile
E8AC   CA         DEX         schon zweimal addiert ?,
E8AD   D0 F6      BNE $E8A5   ja, dann weiter
E8AF   60         RTS         Rücksprung
E8B0   C6 D6      DEC $D6     Zeiger auf Cursorzeile
                              erniedrigen
E8B2   60         RTS         Rücksprung

Einsprung von $E6B6, $E797

E8B3   A2 02      LDX #$02    maximale Zeilenanzahl
E8B5   A9 27      LDA #$27    39, letzte Spalte
E8B7   C5 D3      CMP $D3     wenn Cursorspalte gleich
E8B9   F0 07      BEQ $E8C2   akku ?, dann zu $E8C2
E8BB   18         CLC         40
E8BC   69 28      ADC #$28    addieren
E8BE   CA         DEX         schon zweimal ?,
E8BF   D0 F6      BNE $E8B7   ja, dann weiter
E8C1   60         RTS         Rücksprung
E8C2   A6 D6      LDX $D6     wenn Cursorzeile
E8C4   E0 19      CPX #$19    gleich 25,
E8C6   F0 02      BEQ $E8CA   dann fertig
```

```
E8C8   E6 D6      INC $D6     Zeiger auf Cursorzeile
                              erhöhen
E8CA   60         RTS         Rücksprung

****************************** prüft auf Farbcodes

Einsprung von $E7CE, $E876

E8CB   A2 0F      LDX #$0F    Anzahl der Kodes
E8CD   DD DA E8   CMP $E8DA,X mit Farbcodetabelle
                              vergleichen
E8D0   F0 04      BEQ $E8D6   wenn gefunden, dann farbe
                              setzen
E8D2   CA         DEX         nächster Farbcode
E8D3   10 F8      BPL $E8CD   schon alle ?
E8D5   60         RTS         Rücksprung
E8D6   8E 86 02   STX $0286   Farbcode setzen
E8D9   60         RTS         Rücksprung

****************************** Tabelle der Farb-Kodes
E8DA   90 05 1C 9F 9C 1E 1F 9E
E8E2   81 95 96 97 98 99 9A 9B

****************************** Bildschirm scrollen

Einsprung von $E6D3, $E885, $E975

E8EA   A5 AC      LDA $AC     Alle
E8EC   48         PHA         wichtigen
E8ED   A5 AD      LDA $AD     Zeiger
E8EF   48         PHA         in
E8F0   A5 AE      LDA $AE     den
E8F2   48         PHA         Stack
E8F3   A5 AF      LDA $AF     schie-
E8F5   48         PHA         ben
E8F6   A2 FF      LDX #$FF    ab Zeile Null beginnen
E8F8   C6 D6      DEC $D6     Cursorzeiger
E8FA   C6 C9      DEC $C9     erniedrigen
E8FC   CE A5 02   DEC $02A5   Fortsetzungszeile erniedrigen
E8FF   E8         INX         Zeilennummer erhöhen
```

```
E900  20 F0 E9   JSR $E9F0   Zeiger auf Video-RAM für
                             Zeile X
E903  E0 18      CPX #$18    24
E905  B0 0C      BCS $E913   schon alle Zeilen ?
E907  BD F1 EC   LDA $ECF1,X LOW-Byte holen
E90A  85 AC      STA $AC     und speichern
E90C  B5 DA      LDA $DA,X   HIGH-Byte
E90E  20 C8 E9   JSR $E9C8   Bildschirmzeile nach oben
                             schieben
E911  30 EC      BMI $E8FF   nächste Zeile
E913  20 FF E9   JSR $E9FF   unterste Bildschirmzeile
                             löschen
E916  A2 00      LDX #$00    HIGH-
E918  B5 D9      LDA $D9,X   Bytes
E91A  29 7F      AND #$7F    und
E91C  B4 DA      LDY $DA,X   die
E91E  10 02      BPL $E922   Doppel-
E920  09 80      ORA #$80    zeilen
E922  95 D9      STA $D9,X   ver-
E924  E8         INX         schieben
E925  E0 18      CPX #$18    nicht 24 ?,
E927  D0 EF      BNE $E918   dann nochmal
E929  A5 F1      LDA $F1     Zeile
E92B  09 80      ORA #$80    als einfache Zeile
E92D  85 F1      STA $F1     auszeichnen
E92F  A5 D9      LDA $D9     wenn Fortsetzungszeile,
E931  10 C3      BPL $E8F6   dann nochmal
E933  E6 D6      INC $D6     Zeiger auf Cursor erhöhen
E935  EE A5 02   INC $02A5   Fortsetzungszeile erhöhen
E938  A9 7F      LDA #$7F    Kode
E93A  8D 00 DC   STA $DC00   für
E93D  AD 01 DC   LDA $DC01   Tastaturabfrage
E940  C9 FB      CMP #$FB    CTRL-Taste gedrückt ?
E942  08         PHP         Statusregister retten
E943  A9 7F      LDA #$7F    code für
E945  8D 00 DC   STA $DC00   Tastaturabfrage
E948  28         PLP         Statusregister holen
E949  D0 0B      BNE $E956   nicht gedrückt ?
E94B  A0 00      LDY #$00    Ver-
E94D  EA         NOP         zö-
```

```
E94E   CA        DEX        geru-
E94F   D0 FC     BNE $E94D  ngs-
E951   88        DEY        sch-
E952   D0 F9     BNE $E94D  leife
E954   84 C6     STY $C6    Anzahl der gedrückten
                            Tasten gleich null
E956   A6 D6     LDX $D6    alle

Einsprung von $E9C5

E958   68        PLA        benö-
E959   85 AF     STA $AF    tigten
E95B   68        PLA        Zei-
E95C   85 AE     STA $AE    ger
E95E   68        PLA        zu-
E95F   85 AD     STA $AD    rück-
E961   68        PLA        ho-
E962   85 AC     STA $AC    len
E964   60        RTS        Rücksprung

****************************** Einfügen einer
                              Fortsetzungszeile

Einsprung von $E802

E965   A6 D6     LDX $D6    Zeiger auf Cursorzeile
E967   E8        INX        Zeiger erhöhen
E968   B5 D9     LDA $D9,X  untere Zeile gleich
E96A   10 FB     BPL $E967  Cursorzeile, dann zu $E967
E96C   8E A5 02  STX $02A5  Zeilennummer
E96F   E0 18     CPX #$18   gleich
E971   F0 0E     BEQ $E981  24
E973   90 0C     BCC $E981  dann zu $E981
E975   20 EA E8  JSR $E8EA  Bildschirm scrollen
E978   AE A5 02  LDX $02A5  Zeilennummer
E97B   CA        DEX        erniedrigen
E97C   C6 D6     DEC $D6    Zeiger auf Cursorzeile
                            erniedrigen
E97E   4C DA E6  JMP $E6DA  Zeile initialisieren
```

```
******************************
E981  A5 AC     LDA $AC      Alle
E983  48        PHA          benötigten
E984  A5 AD     LDA $AD      Zeiger
E986  48        PHA          in
E987  A5 AE     LDA $AE      den
E989  48        PHA          Stack
E98A  A5 AF     LDA $AF      schie-
E98C  48        PHA          ben
E98D  A2 19     LDX #$19     25
E98F  CA        DEX          Zeilennummer
E990  20 F0 E9  JSR $E9F0    Zeilen-Zeiger berechnen
E993  EC A5 02  CPX $02A5    alle Zeilen verschoben ?,
E996  90 0E     BCC $E9A6    wenn ja,
E998  F0 0C     BEQ $E9A6    dann zu $E9A6
E99A  BD EF EC  LDA $ECEF,X  LOW-Byte des Zeilenanfangs
E99D  85 AC     STA $AC      setzen
E99F  B5 D8     LDA $D8,X    HIGH-Byte setzen
E9A1  20 C8 E9  JSR $E9C8    Zeile nach oben schieben
E9A4  30 E9     BMI $E98F    Unbedingter Sprung
E9A6  20 FF E9  JSR $E9FF    Bildschirmzeile löschen
E9A9  A2 17     LDX #$17     HIGH-Byte-Tabelle
E9AB  EC A5 02  CPX $02A5    verschieben
E9AE  90 0F     BCC $E9BF    alles verschoben ?
E9B0  B5 DA     LDA $DA,X    HIGH-
E9B2  29 7F     AND #$7F     Byte-
E9B4  B4 D9     LDY $D9,X    und
E9B6  10 02     BPL $E9BA    Doppelzeilen-
E9B8  09 80     ORA #$80     Tabelle
E9BA  95 DA     STA $DA,X    nach
E9BC  CA        DEX          unten schieben
E9BD  D0 EC     BNE $E9AB    schon alles ?
E9BF  AE A5 02  LDX $02A5    Zeilennummer
E9C2  20 DA E6  JSR $E6DA    MSB neu berechnen
E9C5  4C 58 E9  JMP $E958    Register zurückholen, RTS

****************************** Zeile nach oben schieben
E9C8  29 03     AND #$03     Bildschirmzeiger
E9CA  0D 88 02  ORA $0288    für neue Zeile
E9CD  85 AD     STA $AD      berechnen
```

```
E9CF   20 E0 E9   JSR $E9E0   Zeiger für neue Zeile
                              berechnen
E9D2   A0 27      LDY #$27    39 Zeichen
E9D4   B1 AC      LDA ($AC),Y alle
E9D6   91 D1      STA ($D1),Y Zeichen
E9D8   B1 AE      LDA ($AE),Y und
E9DA   91 F3      STA ($F3),Y Farbe übertragen
E9DC   88         DEY         nächstes Zeichen
E9DD   10 F5      BPL $E9D4   schon alle ?
E9DF   60         RTS         Rücksprung

****************************** Bildschirmzeile für
                              Scrollzeile berechnen
E9E0   20 24 EA   JSR $EA24   Zeiger auf Farb-RAM berechnen
E9E3   A5 AC      LDA $AC     Zeiger
E9E5   85 AE      STA $AE     für Zeile
E9E7   A5 AD      LDA $AD     speichern
E9E9   29 03      AND #$03    Startadresse
E9EB   09 D8      ORA #$D8    des Video-RAM
E9ED   85 AF      STA $AF     berechnen
E9EF   60         RTS         Rücksprung

****************************** Zeiger auf Video-RAM für
                              Zeile X
E9F0   BD F0 EC   LDA $ECF0,X LOW-Byte
E9F3   85 D1      STA $D1     holen
E9F5   B5 D9      LDA $D9,X   HIGH-Byte
E9F7   29 03      AND #$03    des
E9F9   0D 88 02   ORA $0288   Video-
E9FC   85 D2      STA $D2     RAM
E9FE   60         RTS         Rücksprung

****************************** Bildschirmzeile X löschen

Einsprung von $E560, $E913, E9A6

E9FF   A0 27      LDY #$27    40-1 Spalten
EA01   20 F0 E9   JSR $E9F0   Zeilenpointer (D1/D2) setzen
EA04   20 24 EA   JSR $EA24   Pointer (F3/F4) für Farb-RAM
                              berechnen
```

```
EA07   A9 20      LDA #$20    Leerzeichen
EA09   91 D1      STA ($D1),Y ins Video-RAM schreiben
EA0B   20 DA E4   JSR $E4DA   Hintergrundfarbe setzen
EA0E   EA         NOP
EA0F   88         DEY         schon 40 Zeichen gelöscht?
EA10   10 F5      BPL $EA07   wenn nicht, fortfahren
EA12   60         RTS         Rücksprung zum Hauptprogramm

****************************

Einsprung von $E5E4, $E6A2

EA13   A8         TAY         Akku retten
EA14   A9 02      LDA #$02
EA16   85 CD      STA $CD     Blinkzähler bei
                              Repeatfunktion setzen

EA18   20 24 EA   JSR $EA24   Pointer für Farb-RAM
                              berechnen
EA1B   98         TAX         Akku wieder holen

**************************** Zeichen und Farbe auf
                              Bildschirm setzen

Einsprung von $EA5E

EA1C   A4 D3      LDY $D3     Spaltenposition
EA1E   91 D1      STA ($D1),Y Zeichen in Akku auf
                              Bildschirm
EA20   8A         TXA         Farb-Code von x in Akku
EA21   91 F3      STA ($F3),Y in Farb-RAM schreiben
EA23   60         RTS         Rücksprung zum Hauptprogramm

**************************** Zeiger auf Farb-RAM berechnen

Einsprung von $E58E, $E75F, $E807, $E9E0, $EA04, $EA18,
             $EA4F

EA24   A5 D1      LDA $D1     $D1/$D2 = Zeiger auf
                              Video-RAM-Position
```

```
EA26   85 F3      STA $F3     LOW-Byte auf Zeichenposition
                              = LOW-Byte auf Farbposition
EA28   A5 D2      LDA $D2     HIGH-Byte der Zeichenposition
EA2A   29 03      AND #$03    mit HIGH-Byte der Farb-RAM-
EA2C   09 D8      ORA #$D8    Position = $D8 verknüpfen und
EA2E   85 F4      STA $F4     in $F4 = speichern
EA30   60         RTS         Rücksprung zum Hauptprogramm

****************************** Interrupt-Routine

Einsprung von $FF58

EA31   20 EA FF   JSR $FFEA   Stop-Taste, Zeit erhöhen
EA34   A5 CC      LDA $CC     Blink-Flag für Cursor
EA36   D0 29      BNE $EA61   nicht blinkend, dann weiter
EA38   C6 CD      DEC $CD     Blinkzähler erniedrigen
EA3A   D0 25      BNE $EA61   nicht Null, dann weiter
EA3C   A9 14      LDA #$14    Blinkzähler wieder auf 20
                              setzen
EA3E   85 CD      STA $CD     und speichern
EA40   A4 D3      LDY $D3     Cursorspalte
EA42   46 CF      LSR $CF     Blinkschalter eins dann C=1
EA44   AE 87 02   LDX $0287   Farbe unter Cursor
EA47   B1 D1      LDA ($D1),Y Zeichen-Kode holen
EA49   B0 11      BCS $EA5C   Blinkschalter war ein, dann
                              weiter
EA4B   E6 CF      INC $CF     Blinkschalter ein
EA4D   85 CE      STA $CE     Zeichen unter Cursor merken
EA4F   20 24 EA   JSR $EA24   Zeiger in Farb-RAM berechnen
EA52   B1 F3      LDA ($F3),Y Farb-Code holen
EA54   8D 87 02   STA $0287   und merken
EA57   AE 86 02   LDX $0286   Farb-Code unter Cursor
EA5A   A5 CE      LDA $CE     Zeichen unter Cursor holen
EA5C   49 80      EOR #$80    RVS-Bit umdrehen
EA5E   20 1C EA   JSR $EA1C   Zeichen und Farbe setzen
EA61   A5 01      LDA $01     Prozessorport laden
EA63   29 10      AND #$10    prüft Rekorder-Taste
EA65   F0 0A      BEQ $EA71   gedrückt, dann verzweige
EA67   A0 00      LDY #$00    Wert für keine Taste gedrückt
EA69   84 C0      STY $C0     Rekorder-Flag setzen
```

```
EA6B   A5 01      LDA $01     Prozessorport laden
EA6D   09 20      ORA #$20    Rekoder-Motor ausschalten
EA6F   D0 08      BNE $EA79   unbedingter Sprung
EA71   A5 C0      LDA $C0     lade Rekorder-Flag
EA73   D0 06      BNE $EA7B   verzweige, wenn Motor läuft

EA75   A5 01      LDA $01     Prozessorport laden
EA77   29 1F      AND #$1F    Rekorder-Motor einschalten
EA79   85 01      STA $01     und wieder speichern
EA7B   20 87 EA   JSR $EA87   Tastaturabfrage
EA7E   AD 0D DC   LDA $DC0D   IRQ-Flag löschen
EA81   68         PLA         Accu aus dem Stapel holen
EA82   A8         TAY         und in Y-Register schieben
EA83   68         PLA         Accu aus dem Stapel holen
EA84   AA         TAX         und in X-Register schieben
EA85   68         PLA         und Rückkehr vom Interrupt
EA86   40         RTI

**************************   Tastaturabfrage

Einsprung von $EA7B, $FF9F

EA87   A9 00      LDA #$00
EA89   8D 8D 02   STA $028D   Shift/CTRL Flag rücksetzen
EA8C   A0 40      LDY #$40    $40 = keine Taste gedrückt
EA8E   84 CB      STY $CB     Kode für gedrückte Taste
EA90   8D 00 DC   STA $DC00   alle Bits des Port A löschen
EA93   AE 01 DC   LDX $DC01   Port B laden
EA96   E0 FF      CPX #$FF    keine Taste gedrückt ?
EA98   F0 61      BEQ $EAFB   dann beenden
EA9A   A8         TAY         Y-Register löschen
EA9B   A9 81      LDA #$81
EA9D   85 F5      STA $F5     $F5/$F6 = Zeiger auf
EA9F   A9 EB      LDA #$EB    Tastaturtabelle setzen
EAA1   85 F6      STA $F6
EAA3   A9 FE      LDA #$FE    erstes Bit für erste
                              Matrixzeile löschen
EAA5   8D 00 DC   STA $DC00   und in Port A schreiben
EAA8   A2 08      LDX #$08    8 Matrixzeilen
EAAA   48         PHA         Bitstellung für Matrix retten
```

```
EAAB   AD 01 DC   LDA $DC01    Port B laden und
EAAE   CD 01 DC   CMP $DC01    Tastatur entprellen
EAB1   D0 F8      BNE $EAAB    noch nicht entprellt ?
EAB3   4A         LSR          Bits nacheinander ins Carry
                               schieben
EAB4   B0 16      BCS $EACC    '1' gleich nicht gedrückt
EAB6   48         PHA          Bitstelung retten
EAB7   B1 F5      LDA ($F5),Y  ASCII-Kode aus Tabelle holen
EAB9   C9 05      CMP #$05     größer als 4, dann keine
                               Control-Taste
EABB   B0 0C      BCS $EAC9    verzweige bei größer/gleich 5
EABD   C9 03      CMP #$03     Kode für STOP-Taste ?
EABF   F0 08      BEQ $EAC9    falls ja, dann verzweige
EAC1   0D 8D 02   ORA $028D    entsprechendes Flag für SHIFT
EAC4   8D 8D 02   STA $028D    COMMOD.-Taste oder CTRL
                               setzen
EAC7   10 02      BPL $EACB    unbedingter Sprung
EAC9   84 CB      STY $CB      Nummer der Taste merken
EACB   68         PLA          Akku holen
EACC   C8         INY          Zähler für Taste erhöhen
EACD   C0 41      CPY #$41     schon alle Tasten?
EACF   B0 0B      BCS $EADC    wenn ja, verzweige
EAD1   CA         DEX          nächste Matrix-Spalte
EAD2   D0 DF      BNE $EAB3    unbedingter Sprung
EAD4   38         SEC          Carry setzen
EAD5   68         PLA          gespeicherte Bitfolge holen
EAD6   2A         ROL  A       verschieben und
EAD7   8D 00 DC   STA $DC00    in Port A schreiben
EADA   D0 CC      BNE $EAA8    unbedingter Sprung
EADC   68         PLA          Stapel normalisieren
EADD   6C 8F 02   JMP ($028F)  JMP $EB48 setzt Zeiger auf
                               Tabelle

Einsprung von $EB76

EAE0   A4 CB      LDY $CB      Nummer der Taste
EAE2   B1 F5      LDA ($F5),Y  ASCII-Wert aus Tabelle
                               holen
EAE4   AA         TAX          Tastenwert retten
```

```
EAE5  C4 C5     CPY $C5     mit letzter Taste
                            vergleichen
EAE7  F0 07     BEQ $EAF0   verzweige wenn gleiche Taste
EAE9  A0 10     LDY #$10    Wert für Repeatverzögerung
EAEB  8C 02     STY $028C   in Repeat-Verzögerungszähler
EAEE  D0 36     BNE $EB26   unbedingter Sprung
EAF0  29 7F     AND #$7F    Bit 7 löschen
EAF2  2C 8A 02  BIT $028A   Repeat-Funktion für alle
                            Tasten ?
EAF5  30 16     BMI $EB0D   Bit 7 gesetzt, dann alle
                            Tasten wiederholen
EAF7  70 49     BVS $EB42   Bit 6 gesetzt, dann
                            keine Wiederholung
EAF9  C9 7F     CMP #$7F    keine Taste?
EAFB  F0 29     BEQ $EB26   ja, dann verzweige
EAFD  C9 14     CMP #$14    'DEL', 'INST' Kode
EAFF  F0 0C     BEQ $EB0D   wenn ja, verzweige
EB01  C9 20     CMP #$20    Leerzeichen
EB03  F0 08     BEQ $EB0D   wenn ja, verzweige
EB05  C9 1D     CMP #$1D    Cursor right, left
EB07  F0 04     BEQ $EB0D   wenn ja, verzweige
EB09  C9 11     CMP #$11    Cursor down, up
EB0B  D0 35     BNE $EB42   verzweige wenn keine Taste
                            zu wiederholen ist
EB0D  AC 8C 02  LDY $028C   Repeatverzögerungszähler
EB10  F0 05     BEQ $EB17   wenn abgelaufen, so verzweige
EB12  CE 8C 02  DEC $028C   herunterzählen
EB15  D0 2B     BNE $EB42   0? nein dann verzweige
EB17  CE 8B 02  DEC $028B   Repeatgeschwindigkeitszähler
EB1A  D0 26     BNE $EB42   0? nein dann verzweige
EB1C  A0 04     LDY #$04    Repeatgeschwindigkeits-
EB1E  8C 8B 02  STY $028B   zähler neu setzen
EB21  A4 C6     LDY $C6     Anzahl der Zeichen im
                            Tastaturpuffer
EB23  88        DEY         herunterzählen
EB24  10 1C     BPL $EB42   mehr als ein Zeichen im
                            Puffer, dann ignorieren
EB26  A4 CB     LDY $CB     Tastennummermatrixcode
EB28  84 C5     STY $C5     umspeichern
EB2A  AC 8D 02  LDY $028D   sowie die Flags für SHIFT
```

```
EB2D   8C 8E 02   STY $028E    COMMOD.-Taste und CTRL
EB30   E0 FF      CPX #$FF     Tastatur-Kode ungültig ?
EB32   F0 0E      BEQ $EB42    ja, dann ignorieren
EB34   8A         TXA          gerettete Taste wieder holen
EB35   A6 C6      LDX $C6      Anzahl der Zeichen im
                               Tastaturpuffer
EB37   EC 89 02   CPX $0289    mit Maximalzahl vergleichen
EB3A   B0 06      BCS $EB42    Puffer voll, dann Zeichen
                               ignorieren
EB3C   9D 77 02   STA $0277,X Zeichen in Tastaturpuffer
                               schreiben
EB3F   E8         INX          Zeichenanzahl erhöhen und
EB40   86 C6      STX $C6      abspeichern
EB42   A9 7F      LDA #$7F     Tastatur-Matrix Abfrage
EB44   8D 00 DC   STA $DC00    auf Normalwert
EB47   60         RTS          Rücksprung

**************************     Prüft auf Shift, CTRL,
                               Commodore

Einsprung von $EADD

EB48   AD 8D 02   LDA $028D    Flag für Shift/CTRL
EB4B   C9 03      CMP #$03     SHIFT und COMMOD.-Taste
                               gedrückt?
EB4D   D0 15      BNE $EB64    nein dann zum Dekodieren
EB4F   CD 8E 02   CMP $028E    waren beide Tasten vorher
                               schon vorher gedrückt
EB52   F0 EE      BEQ $EB42    ja, dann zum Ende
EB54   AD 91 02   LDA $0291    Shift-Commodore erlaubt ?
EB57   30 1D      BMI $EB76    nein, zurück zur
                               Dekodierung
EB59   AD 18 D0   LDA $D018    Zeichensatzzeiger laden
EB5C   49 02      EOR #$02     Umschaltung Klein
                               -Großschreibung und
EB5E   8D 18 D0   STA $D018    wieder speichern
EB61   4C 76 EB   JMP $EB76    fertig
EB64   0A         ASL  A       Wert mit 2 multiplizieren,
                               da jede Adresse 2 Bytes hat
EB65   C9 08      CMP #$08     vergleiche mit CTRL
```

```
EB67   90 02      BCC $EB6B   nein dann verzweige
EB69   A9 06      LDA #$06    Tabellenpointer für CTRL
EB6B   AA         TAX         in X Register übertragen
EB6C   BD 79 EB   LDA $EB79,X LOW-Byte der Tabellenadresse
                              laden
EB6F   85 F5      STA $F5     und in die Zeigeradresse
                              LOW schreiben
EB71   BD 7A EB   LDA $EB7A,X HIGH-Byte der Tabellenadresse
                              laden
EB74   85 F6      STA $F6     und in die Zeigeradresse
                              HIGH schreiben

Einsprung von $EB61

EB76   4C E0 EA   JMP $EAE0   zurück zur Dekodierung

******************************* Zeiger auf Tastatur-
                               Dekodiertabellen
EB79   81 EB C2 EB 03 EC 78 EC

******************************* Tastatur-Dekodiertabelle 1
                               ungeshifted
EB81   14 0D 1D 88 85 86 87 11
EB89   33 57 41 34 5A 53 45 01
EB91   35 52 44 36 43 46 54 58
EB99   37 59 47 38 42 48 55 56
EBA1   39 49 4A 30 4D 4B 4F 4E
EBA9   2B 50 4C 2D 2E 3A 40 2C
EBB1   5C 2A 3B 13 01 3D 5E 2F
EBB9   31 5F 04 32 20 02 51 03
EBC1   FF

******************************* Tastatur-Dekodierung,
                               Tabelle 2 geshifted
EBC2   94 8D 9D 8C 89 8A 8B 91
EBC9   23 D7 C1 24 DA D3 C5 01
EBD2   25 D2 C4 26 C3 C6 D4 D8
EBDA   27 D9 C7 28 C2 C8 D5 D6
EBE2   29 C9 CA 30 CD CB CF CE
EBEA   DB D0 CC DD 3E 5B BA 3C
```

```
EBF2   A9 C0 5D 93 01 3D DE 3F
EBFA   21 5F 04 22 A0 02 D1 83
EC02   FF

****************************** Tastatur-Dekodierung,
                                Tabelle 3, mit 'C='-Taste
EC03   94 8D 9D 8C 89 8A 8B 91
EC0B   96 B3 B0 97 AD AE B1 01
EC13   98 B2 AC 99 BC BB A3 BD
EC1B   9A B7 A5 9B BF B4 B8 BE
EC23   29 A2 B5 30 A7 A1 B9 AA
EC2B   A6 AF B6 DC 3E 5B A4 3C
EC33   A8 DF 5D 93 01 3D DE 3F
EC3B   81 5F 04 95 A0 02 AB 83
EC43   FF

****************************** prüft auf Steuerzeichen

Einsprung von $E7D1

EC44   C9 0E      CMP #$0E    chr$(14) Großschrift
EC46   D0 07      BNE $EC4F   verzweige wenn nein
EC48   AD 18 D0   LDA $D018   Character-Generator
EC4B   09 02      ORA #$02    auf Großschrift-Modus
EC4D   D0 09      BNE $EC58   unbedingter Sprung

Einsprung von $E879

EC4F   C9 8E      CMP #$8E    chr$(142) Kleinschrift
EC51   D0 0B      BNE $EC5E   verzweige wenn nein
EC53   AD 18 D0   LDA $D018   Character-Generator
EC56   29 FD      AND #$FD    Kleinschrift-Modus
EC58   8D 18 D0   STA $D018   setzen
EC5B   4C A8 E6   JMP $E6A8   Ausgabe abschließen
EC5E   C9 08      CMP #$08    chr$(8) Code zur Blockierung
                              SHIFT und COMMOD.-Taste
EC60   D0 07      BNE $EC69   verzweige wenn nein
EC62   A9 80      LDA #$80    oberstes Bit des
EC64   0D 91 02   ORA $0291   Shift-Commodore Flags setzen
EC67   30 09      BMI $EC72   unbedingter Sprung
```

```
EC69   C9 09      CMP #$09    chr$(9) Code zur Freigabe von
                              SHIFT und COMMOD.-Taste
EC6B   D0 EE      BNE $EC5B   verzweige wenn nein
EC6D   A9 7F      LDA #$7F    oberstes Bit des
EC6F   2D 91 02   AND $0291   Shift-Commodore Flags löschen
EC72   8D 91 02   STA $0291   Wert speichern
EC75   4C A8 E6   JMP $E6A8   Ausgabe abschließen

**************************    Tastaturdekodierung,
                              Tabelle 4, mit CTRL-Taste
EC78   FF FF FF FF FF FF FF FF
EC80   1C 17 01 9F 1A 13 05 FF
EC88   9C 12 04 1E 03 06 14 18
EC90   1F 19 07 9E 02 18 15 16
EC98   12 09 0A 92 0D 0B 0F 0E
ECA0   FF 10 0C FF FF 1B 00 FF
ECA8   1C FF 1D FF FF 1F 1E FF
ECB0   90 06 FF 05 FF FF 11 FF
ECB8   FF
**************************    Konstanten für
                              Videocontroller
ECB9   00 00 00 00 00 00 00 00
ECC1   00 00 00 00 00 00 00 00
ECC9   00 9B 37 00 00 00 08 00
ECD1   14 0F 00 00 00 00 00 00
ECD9   0E 06 01 02 03 04 00 01
ECE1   02 03 04 05 06 07

**************************    Text nach Drücken von SHIFT
                              RUN/STOP
ECE7   4C 4F 41 44 0D 52 55 4E  'load (cr) run (cr)'
ECEA   0D

**************************    Tabelle der LSB der
                              Bildschirmzeilen-Anfänge
ECF0   00 28 50 78 A0 C8 F0 18
ECF8   40 68 90 B8 E0 08 30 58
ED00   80 A8 D0 F8 20 48 70 98
ED08   C0
```

```
****************************** IEC-Bus Routinen

****************************** TALK senden

Einsprung von $F238, F4CD, FFB4

ED09   09 40      ORA #$40    Bit für Talk setzen
ED0B   2C         .BYTE $2C   Skip nach $ED0E

****************************** LISTEN senden

Einsprung von $F27A, $F3E3, $F60D, $F648, $FFB1

ED0C   09 20      ORA #$20    Bit für Listen setzen
ED0E   20 A4 F0   JSR $F0A4   Ende der RS 232 Übertragung
                              abwarten

Einsprung von $EE00

ED11   48         PHA         Akku merken
ED12   24 94      BIT $94     Noch Zeichen im Puffer ?
ED14   10 0A      BPL $ED20   verzweige wenn nein
ED16   38         SEC         Carry setzen
ED17   66 A3      ROR $A3     Bit für EOI setzen
ED19   20 40 ED   JSR $ED40   Byte auf IEC-Bus ausgeben
ED1C   46 94      LSR $94     Flag für Zeichen im Puffer
                              löschen
ED1E   46 A3      LSR $A3     Flag für EOI löschen
ED20   68         PLA         Akku wiederholen und
ED21   85 95      STA $95     im Puffer speichern
ED23   78         SEI         Interruptflag setzen
ED24   20 97 EE   JSR $EE97   DATA auf LOW setzen
ED27   C9 3F      CMP #$3F    Akku kann nicht $3F sein
ED29   D0 03      BNE $ED2E   unbedingter Sprung
ED2B   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
ED2E   AD 00 DD   LDA $DD00   Port A laden
ED31   09 08      ORA #$08    ATN HIGH setzen und
ED33   8D 00 DD   STA $DD00   ausgeben
```

```
Einsprung von $EDBB, $EDC9

ED36   78         SEI         Interruptflag setzen
ED37   20 8E EE   JSR $EE8E   CLOCK auf HIGH setzen
ED3A   20 97 EE   JSR $EE97   DATA auf LOW setzen
ED3D   20 B3 EE   JSR $EEB3   eine Millisekunde warten

**************************   ein Byte auf IEC-Bus
                              ausgeben

Einsprung von $ED19, $EDE7

ED40   78         SEI         Interruptflag setzen
ED41   20 97 EE   JSR $EE97   DATA auf LOW setzen
ED44   20 A9 EE   JSR $EEA9   Hardware-Rückmeldung aus
                              DATA holen
ED47   B0 64      BCS $EDAD   DATA LOW, dann 'DEVICE NOT
                              PRESENT'
ED49   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
ED4C   24 A3      BIT $A3     Bit für EOI gesetzt?
ED4E   10 0A      BPL $ED5A   nein, dann verzweige
ED50   20 A9 EE   JSR $EEA9   DATA ins Carry
ED53   90 FB      BCC $ED50   warten bis Listener bereit
ED55   20 A9 EE   JSR $EEA9   DATA ins Carry
ED58   B0 FB      BCS $ED55   warten auf DATA HIGH
ED5A   20 A9 EE   JSR $EEA9   DATA ins Carry
ED5D   90 FB      BCC $ED5A   warten bis bereit für Daten
ED5F   20 8E EE   JSR $EE8E   CLOCK auf HIGH setzen
ED62   A9 08      LDA #$08    Bitzähler für serielle
ED64   85 A5      STA $A5     Ausgabe setzen  ($08 Bits)
ED66   AD 00 DD   LDA $DD00   Port A lesen
ED69   CD 00 DD   CMP $DD00   und entprellen
ED6C   D0 F8      BNE $ED66   verzweige wenn Änderung
ED6E   0A         ASL  A      Datenbit ins Carry
ED6F   90 3F      BCC $EDB0   DATA HIGH, dann 'TIME OUT'
ED71   66 95      ROR $95     nächstes Bit zur Ausgabe
                              bereitstellen
ED73   B0 05      BCS $ED7A   verzweige wenn Bit gesetzt
ED75   20 A0 EE   JSR $EEA0   DATA auf HIGH setzen
ED78   D0 03      BNE $ED7D   unbedingter Sprung
```

```
ED7A   20 97 EE   JSR $EE97   DATA auf LOW setzen
ED7D   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
ED80   EA         NOP         Listener
ED81   EA         NOP         8 Microsekunden Zeit zur
ED82   EA         NOP         Verarbeitung der
ED83   EA         NOP         Daten geben
ED84   AD 00 DD   LDA $DD00   Port A laden
ED87   29 DF      AND #$DF    DATA auf LOW
ED89   09 10      ORA #$10    und CLOCK auf HIGH
ED8B   8D 00 DD   STA $DD00   setzen
ED8E   C6 A5      DEC $A5     nächstes Bit
ED90   D0 D4      BNE $ED66   mache weiter wenn noch nicht
                              alle Bits gesendet
ED92   A9 04      LDA #$04    $04 als Timerwert setzen
ED94   8D 07 DC   STA $DC07   Timer B HIGH, ca. eine ms
ED97   A9 19      LDA #$19    und Timer B
ED99   8D 0F DC   STA $DC0F   starten
ED9C   AD 0D DC   LDA $DC0D   Interrupt control register
ED9F   AD 0D DC   LDA $DC0D   laden
EDA2   29 02      AND #$02    Timer B abgelaufen ?
EDA4   D0 0A      BNE $EDB0   ja, dann 'TIME OUT'
EDA6   20 A9 EE   JSR $EEA9   DATA ins Carry
EDA9   B0 F4      BCS $ED9F   warten auf DATA HIGH
EDAB   58         CLI         Interruptflag löschen
EDAC   60         RTS         Rücksprung

****************************
EDAD   A9 80      LDA #$80    'DEVICE NOT PRESENT'
EDAF   2C         .BYTE $2C   Skip nach $EDB2
EDB0   A9 03      LDA #$03    'TIME OUT'

Einsprung von $EE44

EDB2   20 1C FE   JSR $FE1C   Status setzen
EDB5   58         CLI         Interruptflag löschen
EDB6   18         CLC         Carry setzen
EDB7   90 4A      BCC $EE03   unbedingter Sprung
```

```
*****************************   Sekundäradresse nach LISTEN
                                 senden

Einsprung von $F286, $F3EA, $F612, $F651, $FF93

EDB9   85 95      STA $95      Sekundäradresse speichern
EDBB   20 36 ED   JSR $ED36   mit ATN HIGH ausgeben

Einsprung von $EDD0, $EE03, $F281

EDBE   AD 00 DD   LDA $DD00   Port A laden
EDC1   29 F7      AND #$F7     ATN rücksetzen, LOW
EDC3   8D 00 DD   STA $DD00   und ausgeben
EDC6   60         RTS         Rücksprung

*****************************   Sekundäradresse nach TALK
                                 ausgeben

Einsprung von $F245, $F4D2, $FF96

EDC7   85 95      STA $95      Sekundäradresse speichern
EDC9   20 36 ED   JSR $ED36   mit ATN ausgeben

Einsprung von $F23F

EDCC   78         SEI          Interruptflag setzen
EDCD   20 A0 EE   JSR $EEA0   DATA auf HIGH setzen
EDD0   20 BE ED   JSR $EDBE   ATN rücksetzen, LOW
EDD3   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
EDD6   20 A9 EE   JSR $EEA9   CLOCK-IN holen
EDD9   30 FB      BMI $EDD6   auf CLOCK HIGH warten
EDDB   58         CLI          Interruptflag löschen
EDDC   60         RTS          Rücksprung

*****************************   IECOUT ein Byte auf IEC-Bus
                                 ausgeben

Einsprung von $F1D8, $F3FE, $F61C, $F621, $F62B, $FFA8

EDDD   24 94      BIT $94      noch ein Byte auszugeben ?
```

```
EDDF   30 05      BMI $EDE6   verzweige wenn ja
EDE1   38         SEC         Carry setzen
EDE2   66 94      ROR $94     Flag für gepuffertes Byte
                              setzen
EDE4   D0 05      BNE $EDEB   unbedingter Sprung
EDE6   48         PHA         Byte merken
EDE7   20 40 ED   JSR $ED40   gepuffertes Byte auf Bus
                              ausgeben
EDEA   68         PLA         Byte zurückholen und
EDEB   85 95      STA $95     in Ausgaberegister holen
EDED   18         CLC         Carry löschen
EDEE   60         RTS         Rücksprung

**************************    UNTALK senden

Einsprung von $F340, $F528, $F528, $FFAB

EDEF   78         SEI         Interruptflag setzen
EDF0   20 8E EE   JSR $EE8E   CLOCK auf HIGH setzen
EDF3   AD 00 DD   LDA $DD00   Poar A laden
EDF6   09 08      ORA #$08    ATN HIGH setzen und
EDF8   8D 00 DD   STA $DD00   ausgeben
EDFB   A9 5F      LDA #$5F    Kennzeichnung für UNTALK
EDFD   2C         .BYTE $2C   Skip nach $EE00

**************************    UNLISTEN senden

Einsprung von $F339, $F63F, $F654, $FFAE

EDFE   A9 3F      LDA #$3F    Kennzeichnung für UNLISTEN
EE00   20 11 ED   JSR $ED11   ausgeben
EE03   20 BE ED   JSR $EDBE   ATN rücksetzen, LOW

Einsprung von $EE7D

EE06   8A         TXA         X-Register merken
EE07   A2 0A      LDX #$0A    Warteschleife von
EE09   CA         DEX         ca. 40 Mikrosekunden
EE0A   D0 FD      BNE $EE09   abwarten
EE0C   AA         TAX         X-Register wiederholen
```

```
EE0D   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
EE10   4C 97 EE   JMP $EE97   DATA auf LOW setzen

**************************   IECIN  ein Zeichen vom
                               IEC-Bus holen

Einsprung von $F1B5, $F4D5, $F4E0, $F501, $FFA5

EE13   78         SEI         Interruptflag setzen
EE14   A9 00      LDA #$00    $00 laden
EE16   85 A5      STA $A5     und Zähler löschen
EE18   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
EE1B   20 A9 EE   JSR $EEA9   CLOCK-IN LOW ?
EE1E   10 FB      BPL $EE1B   nein, dann warten
EE20   A9 01      LDA #$01    $01
EE22   8D 07 DC   STA $DC07   in Timer B HIGH schreiben
EE25   A9 19      LDA #$19    Timer
EE27   8D 0F DC   STA $DC0F   starten
EE2A   20 97 EE   JSR $EE97   DATA auf LOW setzen
EE2D   AD 0D DC   LDA $DC0D   Interrupt Control Register
EE30   AD 0D DC   LDA $DC0D   laden
EE33   29 02      AND #$02    Timer B abgelaufen ?
EE35   D0 07      BNE $EE3E   ja, 'TIME OUT'
EE37   20 A9 EE   JSR $EEA9   CLOCK-IN HIGH ?
EE3A   30 F4      BMI $EE30   nein, dann warten
EE3C   10 18      BPL $EE56   unbedingter Sprung
EE3E   A5 A5      LDA $A5     lade Zähler
EE40   F0 05      BEQ $EE47   verzweige wenn $00
EE42   A9 02      LDA #$02    'TIME OUT'
EE44   4C B2 ED   JMP $EDB2   Status setzen
EE47   20 A0 EE   JSR $EEA0   DATA auf HIGH setzen
EE4A   20 85 EE   JSR $EE85   CLOCK auf LOW setzen
EE4D   A9 40      LDA #$40    Bit 6 für 'END OR IDENTIFY'
EE4F   20 1C FE   JSR $FE1C   Status setzen
EE52   E6 A5      INC $A5     Zähler erhöhen
EE54   D0 CA      BNE $EE20   unbedingter Sprung
EE56   A9 08      LDA #$08    $08 als
EE58   85 A5      STA $A5     Bitzähler setzen
EE5A   AD 00 DD   LDA $DD00   Port A laden
EE5D   CD 00 DD   CMP $DD00   Änderung ?
```

```
EE60   D0 F8      BNE $EE5A   verzweige wenn ja
EE62   0A         ASL  A      Datenbit ins Carry schieben
EE63   10 F5      BPL $EE5A   erneut holen wenn CLOCK = 1
EE65   66 A4      ROR $A4     Datenbit in $A4 schieben
EE67   AD 00 DD   LDA $DD00   Port A laden
EE6A   CD 00 DD   CMP $DD00   Änderung ?
EE6D   D0 F8      BNE $EE67   verzweige wenn ja
EE6F   0A         ASL  A      Datenbit ins Carry schieben
EE70   30 F5      BMI $EE67   erneut wenn CLOCK = 0
EE72   C6 A5      DEC $A5     Bitzähler veringerrn
EE74   D0 E4      BNE $EE5A   verzweige wenn noch nicht
                              alle 8 Bits gesendet
EE76   20 A0 EE   JSR $EEA0   DATA auf HIGH setzen
EE79   24 90      BIT $90     Status
EE7B   50 03      BVC $EE80   verzweige wenn kein EOI' ?
EE7D   20 06 EE   JSR $EE06   warten und Bits '0' senden
EE80   A5 A4      LDA $A4     Datenbyte in Akku holen
EE82   58         CLI         Interruptflag löschen
EE83   18         CLC         Carry löschen
EE84   60         RTS         Rücksprung

**************************    CLOCK auf LOW setzen

Einsprung von $ED2B, $ED49, $ED7D, $EDD3, $EE0D, $EE18
             $EE4A

EE85   AD 00 DD   LDA $DD00   Port A laden
EE88   29 EF      AND #$EF    Bit 4 löschen
EE8A   8D 00 DD   STA $DD00   und wieder speichern
EE8D   60         RTS         Rücksprung

**************************    CLOCK auf HIGH setzen

Einsprung von $ED37, $ED5F, $EDF0, $FF7D

EE8E   AD 00 DD   LDA $DD00   Port A laden
EE91   09 10      ORA #$10    Bit 4 setzen
EE93   8D 00 DD   STA $DD00   und wieder speichern
EE96   60         RTS         Rücksprung
```

```
**************************  DATA auf LOW setzen

Einsprung von $ED24, $ED3A, $ED41, $ED7A, $EE10, $EE2A

EE97   AD 00 DD   LDA $DD00   Port A laden
EE9A   29 DF      AND #$DF    Bit 5 löschen
EE9C   8D 00 DD   STA $DD00   und wieder speichern
EE9F   60         RTS         Rücksprung

**************************  DATA auf HIGH setzen

Einsprung von $ED75, $EDCD, $EE47, $EE76

EEA0   AD 00 DD   LDA $DD00   Port A laden
EEA3   09 20      ORA #$20    Bit 5 setzen
EEA5   8D 00 DD   STA $DD00   und wieder speichern
EEA8   60         RTS         Rücksprung

**************************  Bit vom IEC-Bus ins
                            Carry-Flag holen

Einsprung von $ED44, $ED50, $ED55, $ED5A, $EDA6, $EDD6
              $EE1B

EEA9   AD 00 DD   LDA $DD00   Port A laden
EEAC   CD 00 DD   CMP $DD00   Änderung ?
EEAF   D0 F8      BNE $EEA9   verzweige wenn ja
EEB1   0A         ASL  A      Datenbit ins Carry schieben
EEB2   60         RTS         Rücksprung

**************************  Verzögerung 1 Millisekunde

Einsprung von $ED3D

EEB3   8A         TXA         X-Register retten
EEB4   A2 B8      LDX #$B8    X-Register mit $B8 laden
EEB6   CA         DEX         herunterzählen
EEB7   D0 FD      BNE $EEB6   verzweige wenn nicht fertig
EEB9   AA         TAX         X-Register wiederherstellen
EEBA   60         RTS         Rücksprung
```

```
**************************  RS 232 Ausgabe

Einsprung von $FE9D

EEBB   A5 B4      LDA $B4     Anzahl Bits zu senden
EEBD   F0 47      BEQ $EF06   verzweige wenn Byte schon
                              komplett übertragen
EEBF   30 3F      BMI $EF00   verzweige falls Stopbit
                              erforderlich
EEC1   46 B6      LSR $B6     nächstes Bit ins Carry
                              schieben
EEC3   A2 00      LDX #$00    '0' falls Datenbit = 0
EEC5   90 01      BCC $EEC8   verzweige wenn Datenbit
                              gelöscht
EEC7   CA         DEX         nein, dann X-Register =$FF
EEC8   8A         TXA         X-Register in Akku
EEC9   45 BD      EOR $BD     mit Register für Paritybit
                              verknüpfen
EECB   85 BD      STA $BD     und abspeichern
EECD   C6 B4      DEC $B4     Bitzähler erniedrigen
EECF   F0 06      BEQ $EED7   verzweige wenn alle Bits
                              übertragen
EED1   8A         TXA         alten Akku wiederherstellen
EED2   29 04      AND #$04    Bit 2 isolieren
EED4   85 B5      STA $B5     und ins Ausgaberegister
                              bringen
EED6   60         RTS         Rücksprung

EED7   A9 20      LDA #$20    Bit 5 (Parity)
EED9   2C 94 02   BIT $0294   RS 232 Befehlsregister
                              abfragen
EEDC   F0 14      BEQ $EEF2   verzweige wenn ohne Parity
EEDE   30 1C      BMI $EEFC   verzweige wenn feste Parität
EEE0   70 14      BVS $EEF6   verzweige wenn ungerade
                              Parität
EEE2   A5 BD      LDA $BD     verzweige wenn Parity
                              gleich eins
EEE4   D0 01      BNE $EEE7   verzweige wenn ja
EEE6   CA         DEX         Parity $FF
```

```
EEE7  C6 B4     DEC $B4     Bitzähler auf $FF
EEE9  AD 93 02  LDA $0293   RS 232 Kontrollregister laden
EEEC  10 E3     BPL $EED1   verzweige wenn zwei Stopbits
EEEE  C6 B4     DEC $B4     Bitzähler auf $FE
EEF0  D0 DF     BNE $EED1   unbedingter Sprung zur
                            Berechnung der Stopbits

EEF2  E6 B4     INC $B4     Bitzähler erhöhen, keine
                            Parity
EEF4  D0 F0     BNE $EEE6   unbedingter Sprung zur
                            Berechnung der Stopbits

EEF6  A5 BD     LDA $BD     Parity
EEF8  F0 ED     BEQ $EEE7   verzweige wenn gleich 0,
                            dann Null-Bit ausgeben
EEFA  D0 EA     BNE $EEE6   unbedingter Sprung 1-Bit
                            ausgeben

EEFC  70 E9     BVS $EEE7   Null-Bit ausgeben
EEFE  50 E6     BVC $EEE6   sonst 1-Bit ausgeben (feste
                            Parität)

EF00  E6 B4     INC $B4     Bitzähler erhöhen
EF02  A2 FF     LDX #$FF    Wert für Stopbit
EF04  D0 CB     BNE $EED1   unbedingter Sprung

Eisprung von $FF44

EF06  AD 94 02  LDA $0294   RS 232 Befehlsregister laden
EF09  4A        LSR  A      Bit 0 ins Carry
EF0A  90 07     BCC $EF13   verzweige wenn 3-Line
                            Handshake, Abfrage übergehen
EF0C  2C 01 DD  BIT $DD01   Port B abfragen
EF0F  10 1D     BPL $EF2E   verzweige wenn DSR fehlt
EF11  50 1E     BVC $EF31   verzweige wenn CTS fehlt
EF13  A9 00     LDA #$00    0 laden und
EF15  85 BD     STA $BD     Parity-Register löschen
EF17  85 B5     STA $B5     Register für zu sendendes
                            Bit (Startbit)
```

```
EF19   AE 98 02   LDX $0298    Anzahl der zu übertragenden
                                Bits
EF1C   86 B4      STX $B4      als Bitzähler merken
EF1E   AC 9D 02   LDY $029D    lade Zeiger für übertragenes
                                Byte
EF21   CC 9E 02   CPY $029E    alle Bytes übertragen ?
EF24   F0 13      BEQ $EF39    ja, dann abschließen
EF26   B1 F9      LDA ($F9),Y  Datenbyte aus RS 232 Puffer
                                holen
EF28   85 B6      STA $B6      zum Senden übergeben
EF2A   EE 9D 02   INC $029D    Pufferzeiger erhöhen
EF2D   60         RTS          Rücksprung

EF2E   A9 40      LDA #$40     DSR (Data Set Ready) fehlt
EF30   2C         .BYTE $2C    Skip nach $EF33
EF31   A9 10      LDA #$10     CTS (Clear To Send) fehlt
EF33   0D 97 02   ORA $0297    mit Status verknüpfen
EF36   8D 97 02   STA $0297    und setzen
EF39   A9 01      LDA #$01     NMI für

Einsprung von $EF8D, $F041, $F07A

EF3B   8D 0D DD   STA $DD0D    Timer A löschen
EF3E   4D A1 02   EOR $02A1    Flag für
EF41   09 80      ORA #$80     RS 232 umdrehen
EF43   8D A1 02   STA $02A1    und speichern
EF46   8D 0D DD   STA $DD0D    IRR setzen, alle übrigen
                                zulassen NMIs
EF49   60         RTS          Rücksprung

**************************     Anzahl der RS 232 Datenbits
                                berechnen

Einsprung von $F41D

EF4A   A2 09      LDX #$09     Zähler für Wortlänge
EF4C   A9 20      LDA #$20     Maskenwert für Bit 5
EF4E   2C 93 02   BIT $0293    Testen vom RS-232
                                Kontrollregister
EF51   F0 01      BEQ $EF54    verzweige wenn Bit 5 gelöscht
```

```
EF53   CA          DEX          Zähler für Wortlänge
                                vermindern
EF54   50 02       BVC $EF58    verzweige wenn Bit 6 gelöscht
EF56   CA          DEX          Wortlänge um zwei
EF57   CA          DEX          vermindern
EF58   60          RTS          Rücksprung

**************************      empfangenes Bit verarbeiten

Einsprung von $FF04

EF59   A6 A9       LDX $A9      Startbit ?
EF5B   D0 33       BNE $EF90    verzweige wenn ja
EF5D   C6 A8       DEC $A8      Bitzähler erniedrigen
EF5F   F0 36       BEQ $EF97    verzweige wenn alle Bits
                                empfangen
EF61   30 0D       BMI $EF70    verzweige wenn noch Stopbits
                                zu erwarten
EF63   A5 A7       LDA $A7      empfangenes Bit
EF65   45 AB       EOR $AB      mit Register für Parity
                                verknüpfen
EF67   85 AB       STA $AB      und abspeichern
EF69   46 A7       LSR $A7      empfangenes Bit ins Carry
EF6B   66 AA       ROR $AA      und in Empfangsregister
                                schieben
EF6D   60          RTS          Rücksprung

EF6E   C6 A8       DEC $A8      Bitzähler erniedrigen
EF70   A5 A7       LDA $A7      Stopbit
EF72   F0 67       BEQ $EFDB    verzweige wenn gleich Null
EF74   AD 93 02    LDA $0293    Kontrollregister laden
EF77   0A          ASL  A       Bit 7 (Anzahl Stopbits) ins
                                Carry
EF78   A9 01       LDA #$01     1 laden und mit der Anzahl
EF7A   65 A8       ADC $A8      von Bits und Stopbits
                                addieren
EF7C   D0 EF       BNE $EF6D    verzweige wenn noch nicht
                                alle Stopbits empfangen
```

```
Einsprung von $EFD8

EF7E   A9 90      LDA #$90     Wert für Freigabe von NMI
                               über die Flagleitung
EF80   8D 0D DD   STA $DD0D    Wert NMI freigeben
EF83   0D A1 02   ORA $02A1    auch im NMI Register
EF86   8D A1 02   STA $02A1    für RS 232 NMIs vermerken
EF89   85 A9      STA $A9      und Flag für Startbit setzen
EF8B   A9 02      LDA #$02     Bitwert für
EF8D   4C 3B EF   JMP $EF3B    NMI für Timer B löschen

EF90   A5 A7      LDA $A7      Startbit laden
EF92   D0 EA      BNE $EF7E    verzweige wenn ungleich Null
EF94   85 A9      STA $A9      Flag für Startbit
                               rücksetzen
EF96   60         RTS          Rücksprung

**************************     Empfangenes Byte
                               weiterverarbeiten
EF97   AC 9B 02   LDY $029B    Pufferzeiger laden
EF9A   C8         INY          und erhöhen
EF9B   CC 9C 02   CPY $029C    mit Empfangspuffer
                               vergleichen
EF9E   F0 2A      BEQ $EFCA    verzweige wenn voll, dann
                               Status setzen
EFA0   8C 9B 02   STY $029B    Pufferzeiger abspeichern
EFA3   88         DEY          und normalisieren
EFA4   A5 AA      LDA $AA      empfangenes Byte laden
EFA6   AE 98 02   LDX $0298    Anzahl Datenbits laden
EFA9   E0 09      CPX #$09     8 Bits plus ein Stopbit?
EFAB   F0 04      BEQ $EFB1    verzweige wenn ja, ok
EFAD   4A         LSR  A       sonst Bits in richtige
                               Position schieben
EFAE   E8         INX          Datenbitzähler um 1 erhöhen
EFAF   D0 F8      BNE $EFA9    unbedingter Sprung
EFB1   91 F7      STA ($F7),Y  Byte in RS 232 Puffer
                               schreiben
EFB3   A9 20      LDA #$20     Maskenwert für
                               Paritätsprüfung
```

```
EFB5   2C 94 02   BIT $0294    Bit 5 im Kommandregister
                               prüfen
EFB8   F0 B4      BEQ $EF6E    verzweige wenn Übertragung
                               ohne Parity
EFBA   30 B1      BMI $EF6D    verzweige wenn festes Bit
                               anstelle Parity
EFBC   A5 A7      LDA $A7      empfangenes Paritybit laden
EFBE   45 AB      EOR $AB      mit berechneter Parity
                               vergleichen
EFC0   F0 03      BEQ $EFC5    verzweige wenn gleich, ok
EFC2   70 A9      BVS $EF6D    gerade Parity, dann ok
EFC4   2C         .BYTE $2C    Skip nach $EFC7
EFC5   50 A6      BVC $EF6D    verzweige wenn ungerade
                               Parity, dann ok
EFC7   A9 01      LDA #$01     sonst Parity-Fehler
EFC9   2C         .BYTE $2C    Skip nach EFCC
EFCA   A9 04      LDA #$04     Empfängerpuffer voll
EFCC   2C         .BYTE $2C    Skip nach $EFCF
EFCD   A9 80      LDA #$80     Break-Befehl empfangen
EFCF   2C         .BYTE $2C    Skip nach $EFD2
EFD0   A9 02      LDA $#02     Rahmen-Fehler
EFD2   0D 97 02   ORA $0297    mit Code für RS-232 Status
                               verknüpfen
EFD5   8D 97 02   STA $0297    und speichern
EFD8   4C 7E EF   JMP $EF7E    zum Empfang des nächsten
                               Bytes springen

EFDB   A5 AA      LDA $AA      empfangenes Byte
EFDD   D0 F1      BNE $EFD0    ungleich 0, dann zu Rahmen-
                               Fehler
EFDF   F0 EC      BEQ $EFCD    sonst zu Break-Befehl
                               empfangen

**************************     RS-232 CKOUT, Ausgabe auf
                               RS-232

Einsprung von $F26C

EFE1   85 9A      STA $9A      Gerätenummer abspeichern
EFE3   AD 94 02   LDA $0294    RS 232 Kommandregister laden
```

```
EFE6  4A        LSR  A      Bit 0 (Handshake) ins Carry
EFE7  90 29     BCC $F012   verzweige wenn 3-Line-
                            Handshake
EFE9  A9 02     LDA #$02    Maske für DATA SET READY
EFEB  2C 01 DD  BIT $DD01   Port B auslesen
EFEE  10 1D     BPL $F00D   kein DSR, dann Fehler
EFF0  D0 20     BNE $F012   verzweige wenn kein Request
                            To Send
EFF2  AD A1 02  LDA $02A1   RS-232 NMI Status laden
EFF5  29 02     AND #$02    verknüpfe mit Bit für
                            Datenempfang aktiv
EFF7  D0 F9     BNE $EFF2   warten bis Empfang beendet
EFF9  2C 01 DD  BIT $DD01   Port B der NMI-CIA auslesen
EFFC  70 FB     BVS $EFF9   und auf Clear To Send warten
EFFE  AD 01 DD  LDA $DD01   Port B lesen
F001  09 02     ORA #$02    Bit für Request To Send setzen
F003  8D 01 DD  STA $DD01   und wieder zurückschreiben
F006  2C 01 DD  BIT $DD01   Port B holen und
F009  70 07     BVS $F012   auf Clear To Send warten
F00B  30 F9     BMI $F006   verzweige wenn nicht Data Set
                            Ready

Einsprung von $F459

F00D  A9 40     LDA #$40    Bit für fehlendes DSR
F00F  8D 97 02  STA $0297   Status setzen
F012  18        CLC         Carry für ok Kennzeichen
                            setzen
F013  60        RTS         Rücksprung

**************************  Ausgabe in RS 232 Puffer
F014  20 28 F0  JSR $F028   falls erforderlich Übertragung starten

Einsprung von $F208

F017  AC 9E 02  LDY $029E   Zeiger auf Ausgabepuffer
                            laden
F01A  C8        INY         und erhöhen
F01B  CC 9D 02  CPY $029D   und mit Lesezeiger
                            vergleichen
```

```
F01E   F0 F4      BEQ $F014   Puffer voll, dann warten
F020   8C 9E 02   STY $029E   neuen Wert für
                              Schreibzeiger merken
F023   88         DEY        und wieder normalisieren
F024   A5 9E      LDA $9E    auszugebendes Byte holen und
F026   91 F9      STA ($F9),Y in Puffer schreiben

Einsprung von $F014

F028   AD A1 02   LDA $02A1   RS 232 NMI Status laden
F02B   4A         LSR  A     Bit 0 testen (läuft
                              Sendebetrieb)
F02C   B0 1E      BCS $F04C  verzweige wenn ja
F02E   A9 10      LDA #$10   Bitwert für Timer starten
F030   8D 0E DD   STA $DD0E  Timer A starten
F033   AD 99 02   LDA $0299  Timer für
F036   8D 04 DD   STA $DD04  Sende-Baud-Rate
F039   AD 9A 02   LDA $029A  neu
F03C   8D 05 DD   STA $DD05  setzen
F03F   A9 81      LDA #$81   Code für Timer-Unterlauf NMI
                              Timer A
F041   20 3B EF   JSR $EF3B  in IC-Register schreiben
F044   20 06 EF   JSR $EF06  CTS und DSR prüfen und
                              Übertragung freigeben
F047   A9 11      LDA #$11   Bitwert Timer A starten
F049   8D 0E DD   STA $DD0E  Timer A starten
F04C   60         RTS        Rücksprung

**************************   RS-232 CHKIN, Eingabe auf
                              RS-232 setzen

Einsprung von $F227

F04D   85 99      STA $99    Gerätenummer speichern
F04F   AD 94 02   LDA $0294  RS 232 Befehlsregister laden
F052   4A         LSR  A     Bit 0 ins Carry schieben
F053   90 28      BCC $F07D  verzweige wenn 3-Line-
                              Handshake
F055   29 08      AND #$08   Bit für Dupex Mode isolieren
F057   F0 24      BEQ $F07D  verzweige wenn voll Dupex
```

```
F059   A9 02      LDA #$02    Maske für 'RTS OUT'
F05B   2C 01 DD   BIT $DD01   Data Set Ready abfragen
F05E   10 AD      BPL $F00D   verzweige wenn nein
F060   F0 22      BEQ $F084   Ready To Send abfragen
F062   AD A1 02   LDA $02A1   RS 232 NMI Status laden
F065   4A         LSR  A      Bit 0 ins Carry
                               (Sendebetrieb aktiv)
F066   B0 FA      BCS $F062   ja, warten bis beendet
F068   AD 01 DD   LDA $DD01   Port B laden
F06B   29 FD      AND #$FD    Request To Send
F06D   8D 01 DD   STA $DD01   und wieder speichern
F070   AD 01 DD   LDA $DD01   Port B holen
F073   29 04      AND #$04    Bit für Data Terminal Ready
F075   F0 F9      BEQ $F070   verzweige wenn nein, warten
F077   A9 90      LDA #$90    NMI-Maske für 'Flag' laden
F079   18         CLC         Carry löschen (ok Kennzeichen)
F07A   4C 3B EF   JMP $EF3B   NMI freigeben

**************************   RS-232 CHKIN bei 3-Line
                               Handshake

F07D   AD A1 02   LDA $02A1   RS-232 NMI Status laden
F080   29 12      AND #$12    wenn RS-232 nicht aktiv
F082   F0 F3      BEQ $F077   dann starten
F084   18         CLC         Carry löschen (ok
                               Kenneichen)
F085   60         RTS         Rücksprung

**************************   GET von RS-232

Einsprung von $F150

F086   AD 97 02   LDA $0297   RS-232 Status holen
F089   AC 9C 02   LDY $029C   Zeiger auf Ende des
                               Eingabepuffers
F08C   CC 9B 02   CPY $029B   mit Zeiger auf Anfang
                               vergleichen
F08F   F0 0B      BEQ $F09C   verzweige wenn gleich (Puffer
                               leer)
F091   29 F7      AND #$F7    Bit 3 (Puffer leer)
```

```
F093   8D 97 02   STA $0297    im Status löschen (Zeichen
                               im Puffer)
F096   B1 F7      LDA ($F7),Y Byte aus Puffer holen
F098   EE 9C 02   INC $029C    Pufferzeiger erhöhen
F09B   60         RTS          Rücksprung

F09C   09 08      ORA #$08     Bitwert für Puffer leer
F09E   8D 97 02   STA $0297    Status setzen
F0A1   A9 00      LDA #$00     Null übergeben
F0A3   60         RTS          Rücksprung

**************************     Ende der RS-232 Übertragung
                               abwarten

Einsprung von $ED0E, $F88A

F0A4   48         PHA          Akku auf Stack retten
F0A5   AD A1 02   LDA $02A1    RS-232 NMI Status laden
F0A8   F0 11      BEQ $F0BB    nicht gesetzt, dann ok
F0AA   AD A1 02   LDA $02A1    RS-232 NMI Status laden
F0AD   29 03      AND #$03     Bit 0 = senden und Bit 1 =
                               empfangen
F0AF   D0 F9      BNE $F0AA    warten bis beide Bits
                               gelöscht
F0B1   A9 10      LDA #$10     Bitwert für Interrupt durch
F0B3   8D 0D DD   STA $DD0D    'Flag'-Leitung setzen
F0B6   A9 00      LDA #$00     RS-232 NMI Status
F0B8   8D A1 02   STA $02A1    zurücksetzen
F0BB   68         PLA          Akku wieder holen
F0BC   60         RTS          Rücksprung

**************************     Systemmeldungen
F0BD   0D 49 2F 4F 20 45 52 4F  I/O ERROR #
F0C6   52 20 A3
F0C9   0D 53 45 41 52 43 48 49  SEARCHING
F0D1   4E 47 A0
F0D4   46 4F 52 A0                FOR
F0D8   0D 50 52 45 53 53 20 50  PRESS PLAY ON TAPE
F0D9   4C 41 59 20 4F 4E 20 54
F0DB   41 50 C5
```

```
F0EB   50 52 45 53 53 20 52 45  PRESS RECORD & PLAY ON TAPE
F0F3   43 4F 52 44 20 26 20 50
F0FB   4C 41 59 20 4F 4E 20 54
F103   41 50 C5
F106   0D 4C 4F 41 44 49 4E C7  LOADING
F10E   0D 53 41 56 49 4E 47 A0  SAVING
F116   0D 56 45 52 49 46 59 49  VERIFYING
F11E   4E C7
F120   0D 46 4F 55 4E 44 A0     FOUND
F127   0D 4F 4B 8D              OK

**************************   Systemmeldungen ausgeben

Einsprung von $F5DA

F12B   24 9D      BIT $9D      Direkt-Modus Flag
F12D   10 0D      BPL $F13C    Programm, dann überspringen

Einsprung von $F5B5, $F5BE, $F695, $F71F, $F752, $F81E
              $F82B

F12F   B9 BD F0   LDA $F0BD,Y Zeichen holen mit Offset der
                              Meldung in Y-Register
F132   08         PHP          Status-Register retten
F133   29 7F      AND #$7F     Bit 7 löschen
F135   20 D2 FF   JSR $FFD2    und Zeichen ausgeben
F138   C8         INY          Zeiger erhöhen
F139   28         PLP          Status wiederholen
F13A   10 F3      BPL $F12F    verzweige wenn noch weitere
                              Buchstaben
F13C   18         CLC          Carry löschen, ok
F13D   60         RTS          Rücksprung

**************************   GETIN

Einsprung von $FFE4

F13E   A5 99      LDA $99      Eingabegerät laden
F140   D0 08      BNE $F14A    verzweige wenn nicht Tastatur
```

```
F142   A5 C6      LDA $C6      Anzahl der Zeichen im
                                Tastaturpuffer laden
F144   F0 0F      BEQ $F155    verzweige wenn kein Zeichen
F146   78         SEI          Interruptflag setzen
F147   4C B4 E5   JMP $E5B4    Zeichen aus Tastaturpuffer
                                holen

F14A   C9 02      CMP #$02     Geräteadresse für RS-232
F14C   D0 18      BNE $F166    nein dann zur BASIN-Routine

Einsprung von $F1B8

F14E   84 97      STY $97      Y-Register merken
F150   20 86 F0   JSR $F086    Get von RS 232
F153   A4 97      LDY $97      Y-Register wiederholen
F155   18         CLC          Carry löschen, ok
F156   60         RTS          Rücksprung

**************************    BASIN  Eingabe eines
                                Zeichens

Einsprung von $FFCF

F157   A5 99      LDA $99      Gerätenummer laden
F159   D0 0B      BNE $F166    verzweige wenn nicht Tastatur
F15B   A5 D3      LDA $D3      Cursorposition holen
F15D   85 CA      STA $CA      und für
F15F   A5 D6      LDA $D6      Tastatureingabe
F161   85 C9      STA $C9      setzen
F163   4C 32 E6   JMP $E632    Eingabe vom Bildschirm

F166   C9 03      CMP #$03     Eingabekanal 3 = Bildschirm
F168   D0 09      BNE $F173    wenn nicht verzweige

**************************    vom Bildschirm
F16A   85 D0      STA $D0      Flag auf Eingabe von Bild-
                                schimrstelle
F16C   A5 D5      LDA $D5      Cursorzeile laden
F16E   85 C8      STA $C8      als Pointer für Ende der
                                Zeile speichern
```

```
F170   4C 32 E6   JMP $E632   zu Eingabe vom Bildschirm
F173   B0 38      BCS $F1AD   verzweige zu Eingabe vom
                              IEC-Bus
F175   C9 02      CMP #$02    Eingabe von RS-232 ?
F177   F0 3F      BEQ $F1B8   ja, so verzweige

**************************    Eingabe vom Band
F179   86 97      STX $97     X-Register merken
F17B   20 99 F1   JSR $F199   ein Zeichen vom Band holen
F17E   B0 16      BCS $F196   verzweige bei Fehler
F180   48         PHA         Akku retten
F181   20 99 F1   JSR $F199   ein Zeichen vom Band holen
F184   B0 0D      BCS $F193   verzweige bei Fehler
F186   D0 05      BNE $F18D   letzes Zeichen ?
F188   A9 40      LDA #$40    Code für 'End of Identify'
F18A   20 1C FE   JSR $FE1C   Status setzen
F18D   C6 A6      DEC $A6     Bandpuffer Zeiger erniedrigen
F18F   A6 97      LDX $97     X-Register zurückholen
F191   68         PLA         geholtes Zeichen in Akku
F192   60         RTS         Rücksprung

F193   AA         TAX         Fehlernummer ins X-Register
F194   68         PLA         Stack normalisieren
F195   8A         TXA         Fehlernummer in Akku
F196   A6 97      LDX $97     X-Register zurückholen
F198   60         RTS         Rücksprung

**************************    ein Zeichen vom Band holen

Einsprung von $F17B, $F181

F199   20 0D F8   JSR $F80D   Bandpuffer Zeiger erhöhen
F19C   D0 0B      BNE $F1A9   verzweige wenn noch Zeichen
                              im Puffer
F19E   20 41 F8   JSR $F841   sonst nächsten Block vom
                              Band holen
F1A1   B0 11      BCS $F1B4   STOP-Taste, dann Abbruch
F1A3   A9 00      LDA #$00    Pufferzeiger
F1A5   85 A6      STA $A6     auf Null
F1A7   F0 F0      BEQ $F199   unbedingter Sprung
```

```
F1A9   B1 B2      LDA ($B2),Y Zeichen aus Puffer lesen
F1AB   18         CLC         Carry =0 (ok Kennzeichen)
F1AC   60         RTS         Rücksprung

**************************    Eingabe vom IEC-Bus
F1AD   A5 90      LDA $90     Status testen
F1AF   F0 04      BEQ $F1B5   verzweige wenn ok
F1B1   A9 0D      LDA #$0D    'CR' Kode ausgeben
F1B3   18         CLC         Carry =0 (ok Kennzeichen)
F1B4   60         RTS         Rücksprung

F1B5   4C 13 EE   JMP $EE13   ein Byte vom IEC-Bus holen

**************************    RS 232 Eingabe
F1B8   20 4E F1   JSR $F14E   ein Byte von RS 232 holen
F1BB   B0 F7      BCS $F1B4   verzweige wenn Fehler
F1BD   C9 00      CMP #$00    vergleiche mit Nullbyte
F1BF   D0 F2      BNE $F1B3   nein, dann ok
F1C1   AD 97 02   LDA $0297   Status laden
F1C4   29 60      AND #$60    fehlt DSR ?
F1C6   D0 E9      BNE $F1B1   ja,'CR' zurückgeben
F1C8   F0 EE      BEQ $F1B8   nein, neuer Versuch

**************************    BSOUT  Ausgabe eines
                              Zeichens

Einsprung von $FFD2

F1CA   48         PHA         Datenbyte retten
F1CB   A5 9A      LDA $9A     Gerätenummer für Ausgabe
F1CD   C9 03      CMP #$03    vergleiche mit Bildschirm
F1CF   D0 04      BNE $F1D5   verzweige wenn nein
F1D1   68         PLA         Datenbyte wiederholen
F1D2   4C 16 E7   JMP $E716   ein Zeichen auf Bildschirm
                              ausgeben

F1D5   90 04      BCC $F1DB   verzweige wenn keine Ausgabe
                              IEC-Bus
```

```
**************************      Ausgabe auf IEC-Bus
F1D7   68          PLA          Datenbyte retten
F1D8   4C DD ED    JMP $EDDD    ein Byte auf IEC-Bus ausgeben

F1DB   4A          LSR  A       Bit 0 der Ausgabekanal-
                                Nummer ins Carry
F1DC   68          PLA          Datenbyte wiederholen

Einsprung von $F2D4

F1DD   85 9E       STA $9E      auszugebendes Zeichen merken
F1DF   8A          TXA          X-Register
F1E0   48          PHA          und Y-Register
F1E1   98          TYA          auf Stack
F1E2   48          PHA          retten
F1E3   90 23       BCC $F208    RS-232 Ausgabe

**************************      Ausgabe auf Band
F1E5   20 0D F8    JSR $F80D    Bandpuffer Zeiger erhöhen
F1E8   D0 0E       BNE $F1F8    verzweige wenn Puffer
                                nicht voll
F1EA   20 64 F8    JSR $F864    Puffer auf Band schreiben
F1ED   B0 0E       BCS $F1FD    STOP-Taste, dann Abbruch
F1EF   A9 02       LDA #$02     Kontrollbyte für Datenblock
F1F1   A0 00       LDY #$00     Pufferzeiger auf 0
F1F3   91 B2       STA ($B2),Y  Akku in Puffer schreiben
F1F5   C8          INY          Zeiger erhöhen
F1F6   84 A6       STY $A6      und merken
F1F8   A5 9E       LDA $9E      Datenbyte holen
F1FA   91 B2       STA ($B2),Y  Zeichen in Puffer schreiben

Einsprung von $F20B

F1FC   18          CLC          Carry =0 (ok Kennzeichen)
F1FD   68          PLA          X-Register
F1FE   A8          TAY          und Y-Register
F1FF   68          PLA          aus Stack
F200   AA          TAX          holen
F201   A5 9E       LDA $9E      Datenbyte zurückholen
F203   90 02       BCC $F207    verzweige wenn ok
```

```
F205   A9 00      LDA #$00    Flag für 'STOP-Taste
                              gedrückt'
F207   60         RTS         Rücksprung

**************************   RS-232 Ausgabe
F208   20 17 F0   JSR $F017   ein Zeichen in RS-232
                              Puffer schreiben
F20B   4C FC F1   JMP $F1FC   CHROUT

**************************   CHKIN  Eingabegerät setzen

Einsprung von $FFC6

F20E   20 0F F3   JSR $F30F   sucht logische Filenummer
F211   F0 03      BEQ $F216   verzweige wenn gefunden
F213   4C 01 F7   JMP $F701   sonst 'file not open'
F216   20 1F F3   JSR $F31F   setzt Fileparameter
F219   A5 BA      LDA $BA     Gerätenummer laden
F21B   F0 16      BEQ $F233   0, Tastatur
F21D   C9 03      CMP #$03    vergleiche mit Bildschirm
F21F   F0 12      BEQ $F233   verzweige zu Bildschirm
F221   B0 14      BCS $F237   verzweige zu IEC-Bus
F223   C9 02      CMP #$02    vergleiche mit RS-232
F225   D0 03      BNE $F22A   nein, dann Band
F227   4C 4D F0   JMP $F04D   ja, dann RS-232

**************************   Band als Eingabegerät setzen
F22A   A6 B9      LDX $B9     Sekundäradresse laden
F22C   E0 60      CPX #$60    vergleiche mit 'Null'
F22E   F0 03      BEQ $F233   verzweige wenn 'Null'
F230   4C 0A F7   JMP $F70A   sonst 'not input file'
F233   85 99      STA $99     Gerätenummer für Ausgabe
                              speichern
F235   18         CLC         Carry =0 (ok Kennzeichen)
F236   60         RTS         Rücksprung

**************************   IEC-Bus als Eingabegerät
F237   AA         TAX         Geräteadresse retten
F238   20 09 ED   JSR $ED09   TALK senden
F23B   A5 B9      LDA $B9     Sekundäradresse laden
```

```
F23D  10 06     BPL $F245   verzweige wenn kleiner 128
F23F  20 CC ED  JSR $EDCC   wartet auf Takt-Signal
F242  4C 48 F2  JMP $F248   nächsten Befehl überspringen
F245  20 C7 ED  JSR $EDC7   Sekundäradresse für TALK
                            senden

Einsprung von $F242

F248  8A        TXA         Geräteadresse wiederholen
F249  24 90     BIT $90     Status abfragen
F24B  10 E6     BPL $F233   verzweige wenn ok
F24D  4C 07 F7  JMP $F707   sonst 'DEVICE NOT PRESENT'

**************************  CKOUT  Ausgabegerät setzen

Einsprung von $FFC9

F250  20 0F F3  JSR $F30F   sucht logische Filenummer
F253  F0 03     BEQ $F258   verzweige wenn gefunden
F255  4C 01 F7  JMP $F701   sonst 'FILE NOT OPEN'
F258  20 1F F3  JSR $F31F   setzt Fileparameter
F25B  A5 BA     LDA $BA     Gerätenummer holen
F25D  D0 03     BNE $F262   verzweige wenn ungleich Null
F25F  4C 0D F7  JMP $F70D   sonst 'NOT INPUT FILE'
F262  C9 03     CMP #$03    vergleiche mit Bildschirm ?
F264  F0 0F     BEQ $F275   verzweige wenn Bildschirm
F266  B0 11     BCS $F279   verzweige wenn IEC-Bus
F268  C9 02     CMP #$02    vergleiche mit RS-232
F26A  D0 03     BNE $F26F   verzweige wenn nein
F26C  4C E1 EF  JMP $EFE1   Ausgabe auf RS-232
                            vorbereiten

**************************  Band als Ausgabegerät setzen
F26F  A6 B9     LDX $B9     Sekundäradresse laden
F271  E0 60     CPX #$60    mit 'Null' vergleichen
F273  F0 EA     BEQ $F25F   Bandfile zum Lesen, 'NOT
                            OUTPUT FILE'
F275  85 9A     STA $9A     Nummer des Ausgabegeräts
                            setzen
```

```
F277   18          CLC         Carry =0 (ok Kennzeichen)
F278   60          RTS         Rücksprung

**************************     Ausgabe auf IEC-Bus legen
F279   AA          TAX         Geräteadresse retten
F27A   20 0C ED    JSR $ED0C   LISTEN senden
F27D   A5 B9       LDA $B9     Sekundäradresse laden
F27F   10 05       BPL $F286   verzweige wenn kleiner 128
F281   20 BE ED    JSR $EDBE   ATN zurücksetzen
F284   D0 03       BNE $F289   unbedingter Sprung
F286   20 B9 ED    JSR $EDB9   Sekundäradresse für LISTEN
                               senden
F289   8A          TXA         Geräteadresse wiederholen
F28A   24 90       BIT $90     Status abfragen
F28C   10 E7       BPL $F275   verzweige wenn ok
F28E   4C 07 F7    JMP $F707   'device not present'

**************************     CLOSE logische Filenummer
                               im Akku

Einsprung von $FFC3

F291   20 14 F3    JSR $F314   sucht logische Filenummer
F294   F0 02       BEQ $F298   verzweige wenn gefunden
F296   18          CLC         File nicht vorhanden, dann
                               fertig
F297   60          RTS         Rücksprung

F298   20 1F F3    JSR $F31F   Fileparameter setzen
F29B   8A          TXA         Zeiger auf Parametereintrag
                               in Filetabelle
F29C   48          PHA         retten
F29D   A5 BA       LDA $BA     Geräteadresse laden
F29F   F0 50       BEQ $F2F1   verzweige wenn Tastatur
F2A1   C9 03       CMP #$03    vergleiche mit Bildschirm
F2A3   F0 4C       BEQ $F2F1   verzweige wenn Bildschirm
F2A5   B0 47       BCS $F2EE   verzweige wenn IEC-Bus
F2A7   C9 02       CMP #$02    vergleiche mit RS-232
F2A9   D0 1D       BNE $F2C8   nein, dann Band
```

```
**************************  RS-232 File schließen
F2AB  68        PLA         Zeiger auf Parametereintrag
F2AC  20 F2 F2  JSR $F2F2   Fileeintrag in Tabelle
                            löschen
F2AF  20 83 F4  JSR $F483   CIAs für I/O rücksetzen
F2B2  20 27 FE  JSR $FE27   Memory-Top holen
F2B5  A5 F8     LDA $F8     RS-232 Eingabepuffer
                            HIGH-Byte laden
F2B7  F0 01     BEQ $F2BA   verzweige wenn 0
F2B9  C8        INY         HIGH-Byte von Memory-Top
                            erhöhen
F2BA  A5 FA     LDA $FA     RS-232 Ausgabepuffer
                            HIGH-Byte laden
F2BC  F0 01     BEQ $F2BF   verzweige wenn 0
F2BE  C8        INY         sonst HIGH-Byte von Memory-
                            Top erhöhen
F2BF  A9 00     LDA #$00    0 laden
F2C1  85 F8     STA $F8     und Puffer
F2C3  85 FA     STA $FA     freigeben
F2C5  4C 7D F4  JMP $F47D   Memory Top neu setzen

**************************  Band File schließen
F2C8  A5 B9     LDA $B9     Sekundäradresse laden
F2CA  29 0F     AND #$0F    Bits 0 bis 3 isolieren
F2CC  F0 23     BEQ $F2F1   verzweige wenn File zum Lesen
F2CE  20 D0 F7  JSR $F7D0   Band-Puffer Startadresse
                            holen
F2D1  A9 00     LDA #$00    Markierung für letztes
                            Zeichen im Datenpuffer
F2D3  38        SEC         Flag für Ausgabe auf Recorder
F2D4  20 DD F1  JSR $F1DD   Zeichen in Kassettenpuffer
F2D7  20 64 F8  JSR $F864   Puffer auf Band schreiben
F2DA  90 04     BCC $F2E0   verzweige wenn alles ok
F2DC  68        PLA         Zeiger auf Fileeintrag holen
F2DD  A9 00     LDA #$00    0 für Break
F2DF  60        RTS         Rücksprung

F2E0  A5 B9     LDA $B9     Sekundäradresse laden
F2E2  C9 62     CMP #$62    vergleiche auf Open mit EOT
F2E4  D0 0B     BNE $F2F1   verzweige wenn kein EOT
```

```
F2E6   A9 05      LDA #$05    Kontrollbyte für EOT-Header
F2E8   20 6A F7   JSR $F76A   Block auf Band schreiben
F2EB   4C F1 F2   JMP $F2F1   Überspringe nächsten Befehl
F2EE   20 42 F6   JSR $F642   IEC-File schließen

Einsprung von $F2EB

F2F1   68         PLA         Zeiger auf Fileeintrag holen

Einsprung von $F2AC

F2F2   AA         TAX         ins X-Register schieben
F2F3   C6 98      DEC $98     Anzahl der offenen Files
                              erniedrigen
F2F5   E4 98      CPX $98     und mit Zeiger auf
                              Fileeintrag vergleichen
F2F7   F0 14      BEQ $F30D   gleich, dann fertig
F2F9   A4 98      LDY $98     Anzahl der offenen Files
F2FB   B9 59 02   LDA $0259,Y Letzten Fileeintrag
F2FE   9D 59 02   STA $0259,X an die
F301   B9 63 02   LDA $0263,Y freigewordene
F304   9D 63 02   STA $0263,X Stelle in der
F307   B9 6D 02   LDA $026D,Y Filetabelle
F30A   9D 6D 02   STA $026D,X schreiben
F30D   18         CLC         Carry =0 (ok Kennzeichnung)
F30E   60         RTS         Rücksprung

**************************    sucht logische Filenummer
                              (in X)

Einsprung von $F20E, $F250, $F351

F30F   A9 00      LDA #$00    Status
F311   85 90      STA $90     löschen
F313   8A         TXA         Filenummer in Akku schieben

Einsprung von $F291

F314   A6 98      LDX $98     Anzahl der offenen Files
F316   CA         DEX         Anzahl um eins verringern
```

```
F317   30 15      BMI $F32E    verzweige wenn kein File
                               offen oder Filenummer nicht
                               gefunden
F319   DD 59 02   CMP $0259,X sucht Eintrag in Tabelle
F31C   D0 F8      BNE $F316    verzweige wenn noch nicht
                               gefunden
F31E   60         RTS          Rücksprung

**************************     setzt Fileparameter

Einsprung von $F216, $F258, $F298

F31F   BD 59 02   LDA $0259,X logische Filenummer aus
F322   85 B8      STA $B8      Tabelle holen und speichern
F324   BD 63 02   LDA $0263,X Geräteadresse aus Tabelle
F327   85 BA      STA $BA      holen und speichern
F329   BD 6D 02   LDA $026D,X Sekundäradresse aus Tabelle
F32C   85 B9      STA $B9      holen und speichern
F32E   60         RTS          Rücksprung

**************************     CLALL  schließt alle
                               Ein-/Ausgabe Kanäle

Einsprung von $FFE7

F32F   A9 00      LDA #$00     Anzahl der offenen Files
F331   85 98      STA $98      auf Null stellen

**************************** CLRCH  schließt aktiven
                               I/O-Kanal

Einsprung von $FFCC

F333   A2 03      LDX #$03     Vergleichswert in X
F335   E4 9A      CPX $9A      vergleiche mit Nummer des
                               Ausgabegeräts
F337   B0 03      BCS $F33C    verzweige wenn kleiner als 3
F339   20 FE ED   JSR $EDFE    IEC, UNLISTEN senden
F33C   E4 99      CPX $99      vergleiche mit Nummer des
                               Eingabegeräts
```

```
F33E   B0 03      BCS $F343   verzweige wenn kleiner als 3
F340   20 EF ED   JSR $EDEF   IEC, UNTALK senden
F343   86 9A      STX $9A     Ausgabe wieder auf Bildschirm
F345   A9 00      LDA #$00    Eingabe wieder
F347   85 99      STA $99     von Tastatur
F349   60         RTS         Rücksprung

**************************** OPEN

Einsprung von $FFC0

F34A   A6 B8      LDX $B8     Filenummer in X
F34C   D0 03      BNE $F351   verzweige wenn ungleich Null
F34E   4C 0A F7   JMP $F70A   'not input file'   (??)
F351   20 0F F3   JSR $F30F   sucht logische Filenummer
F354   D0 03      BNE $F359   nicht gefunden, kann neu
                              angelegt werden
F356   4C FE F6   JMP $F6FE   sonst 'file open'
F359   A6 98      LDX $98     Anzahl der offenen Files
F35B   E0 0A      CPX #$0A    mit 10 vergleichen
F35D   90 03      BCC $F362   kleiner 10 dann ok
F35F   4C FB F6   JMP $F6FB   'too many files'
F362   E6 98      INC $98     Anzahl erhöhen
F364   A5 B8      LDA $B8     logische Filenummer laden
F366   9D 59 02   STA $0259,X und in die Tabelle schreiben
F369   A5 B9      LDA $B9     Sekundäradresse laden
F36B   09 60      ORA #$60    Bit 5 und 6 setzen
F36D   85 B9      STA $B9     wieder speichern
F36F   9D 6D 02   STA $026D,X und in die Tabelle schreiben
F372   A5 BA      LDA $BA     Gerätenummer laden
F374   9D 63 02   STA $0263,X und in die Tabelle schreiben
F377   F0 5A      BEQ $F3D3   verzweige wenn Gerätenummer
                              für Tastatur
F379   C9 03      CMP #$03    Code für Bildschirm
F37B   F0 56      BEQ $F3D3   ja, so verzweige
F37D   90 05      BCC $F384   verzweige wenn nicht IEC-Bus
F37F   20 D5 F3   JSR $F3D5   File auf IEC-Bus eröffnen
F382   90 4F      BCC $F3D3   unbedingter Sprung
F384   C9 02      CMP #$02    Code für Band
F386   D0 03      BNE $F38B   verzweige wenn nein
```

```
F388  4C 09 F4   JMP $F409     RS-232 open
F38B  20 D0 F7   JSR $F7D0     Bandpuffer Startadresse in X
                               und Y holen
F38E  B0 03      BCS $F393     verzweige wenn HIGH-Byte
                               größer als 2
F390  4C 13 F7   JMP $F713     'illegal device number'
F393  A5 B9      LDA $B9       Sekundäradresse laden
F395  29 0F      AND #$0F      Bits 0 bis 3 isolieren
F397  D0 1F      BNE $F3B8     ungleich Null dann schreiben
F399  20 17 F8   JSR $F817     wartet auf Play-Taste
F39C  B0 36      BCS $F3D4     verzweige wenn Play Taste
                               gedrückt
F39E  20 AF F5   JSR $F5AF     'SEARCHING' ('for name')
                               ausgeben
F3A1  A5 B7      LDA $B7       Länge des Filenamens
F3A3  F0 0A      BEQ $F3AF     kein Filename, dann weiter
F3A5  20 EA F7   JSR $F7EA     sucht gewünschen Bandheader
F3A8  90 18      BCC $F3C2     verzweige wenn gefunden
F3AA  F0 28      BEQ $F3D4     verzweige wenn  STOP-Taste
F3AC  4C 04 F7   JMP $F704     EOT, 'FILE NOT FOUND'
                               ausgeben
F3AF  20 2C F7   JSR $F72C     nächsten Bandheader suchen
F3B2  F0 20      BEQ $F3D4     EOT, Fehler
F3B4  90 0C      BCC $F3C2     verzweige wenn gefunden
F3B6  B0 F4      BCS $F3AC     sonst PRG-File, weiter suchen

F3B8  20 38 F8   JSR $F838     wartet auf Record & Play
                               Taste
F3BB  B0 17      BCS $F3D4     STOP-Taste, dann Abbruch
F3BD  A9 04      LDA #$04      Kontrollbyte für Datenheader
F3BF  20 6A F7   JSR $F76A     Header auf Band schreiben
F3C2  A9 BF      LDA #$BF      Zeiger auf Ende des
                               Bandpuffers
F3C4  A4 B9      LDY $B9       Sekundäradresse laden
F3C6  C0 60      CPY #$60      vergleiche mit $60 für Band
                               lesen
F3C8  F0 07      BEQ $F3D1     lesen, dann verzweige
F3CA  A0 00      LDY #$00      Zeiger auf 0 setzen
F3CC  A9 02      LDA #$02      Kontrollbyte für Datenblock
F3CE  91 B2      STA ($B2),Y in Bandpuffer schreiben
```

```
F3D0   98          TYA          Zeiger in Akku
F3D1   85 A6       STA $A6      Zeiger in Bandpuffer setzen
F3D3   18          CLC          Carry =0 (ok Kennzeichen)
F3D4   60          RTS          Rücksprung

**************************  File auf IEC-Bus eröffnen

Einsprung von $F37F, $F4C8, $F605

F3D5   A5 B9       LDA $B9      Sekundäradresse laden
F3D7   30 FA       BMI $F3D3    Rücksprung wenn größer,
                                gleich 128
F3D9   A4 B7       LDY $B7      Länge des Filenamens laden
F3DB   F0 F6       BEQ $F3D3    gleich Null, dann fertig
F3DD   A9 00       LDA #$00     Status
F3DF   85 90       STA $90      löschen
F3E1   A5 BA       LDA $BA      Geräteadressse laden
F3E3   20 0C ED    JSR $ED0C    LISTEN
F3E6   A5 B9       LDA $B9      Sekundäradresse laden
F3E8   09 F0       ORA #$F0     Bits 4 bis 7 setzen (Open
                                Kennzeichnung)
F3EA   20 B9 ED    JSR $EDB9    Sekundäradresse senden
F3ED   A5 90       LDA $90      Status testen
F3EF   10 05       BPL $F3F6    verzweige wenn ok
F3F1   68          PLA          Stack
F3F2   68          PLA          rücksetzen
F3F3   4C 07 F7    JMP $F707    'device not present'
F3F6   A5 B7       LDA $B7      Länge des Filenamens
F3F8   F0 0C       BEQ $F406    kein Filename, dann fertig
F3FA   A0 00       LDY #$00     Zeiger auf Null setzen
F3FC   B1 BB       LDA ($BB),Y  Filenamen holen
F3FE   20 DD ED    JSR $EDDD    auf IEC-Bus ausgeben
F401   C8          INY          Zeiger erhöhen
F402   C4 B7       CPY $B7      mit Länge des Filenamens
                                vergleichen
F404   D0 F6       BNE $F3FC    verzweige wenn noch nicht
                                alle Zeichen
F406   4C 54 F6    JMP $F654    UNLISTEN, return
```

```
**************************  RS-232 Open

Einsprung von $F388

F409   20 83 F4   JSR $F483   CIAs setzen
F40C   8C 97 02   STY $0297   RS-232 Status löschen
F40F   C4 B7      CPY $B7     Länge des "Filenamens"
F411   F0 0A      BEQ $F41D   verzweige wenn kein Filename
F413   B1 BB      LDA ($BB),Y die ersten
F415   99 93 02   STA $0293,Y vier
F418   C8         INY         Zeichen
F419   C0 04      CPY #$04    speichern
F41B   D0 F2      BNE $F40F   verzweige wenn noch nicht
                              alle vier Zeichen
F41D   20 4A EF   JSR $EF4A   Anzahl der Datenbits
                              berechnen
F420   8E 98 02   STX $0298   und speichern
F423   AD 93 02   LDA $0293   Kontrollregister holen
F426   29 0F      AND #$0F    Bits für Baud-Rate isolieren
F428   F0 1C      BEQ $F446   verzweige wenn User-Baud-Rate
F42A   0A         ASL  A      mal 2 für Tabelle
F42B   AA         TAX         als Zeiger merken
F42C   AD A6 02   LDA $02A6   NTSC-Version
F42F   D0 09      BNE $F43A   verzweige wenn nein
F431   BC C1 FE   LDY $FEC1,X Baud-Rate, HIGH für
                              NTSC-Timing
F434   BD C0 FE   LDA $FEC0,X Baud-Rate, LOW
F437   4C 40 F4   JMP $F440   überspringe zwei Befehle
F43A   BC EB E4   LDY $E4EB,X Baud-Rate, HIGH  für
                              PAL-Timing
F43D   BD EA E4   LDA $E4EA,X Baud-Rate, LOW

Einsprung von $F437

F440   8C 96 02   STY $0296   HIGH-Byte speichern
F443   8D 95 02   STA $0295   LOW-Byte speichern
F446   AD 95 02   LDA $0295   Timerwert = Baud-Rate *
                              zwei + $C8 (200)
F449   0A         ASL  A      Timer LOW * zwei
```

```
F44A  20 2E FF   JSR $FF2E   Timerwert für Baud-Rate
                             ermitteln
F44D  AD 94 02   LDA $0294   Kommandoregister laden
F450  4A         LSR  A      Prüfe ob 3-Line-Handshake
F451  90 09      BCC $F45C   verzweige wenn ja
F453  AD 01 DD   LDA $DD01   Prüfe ob Data Set Ready
F456  0A         ASL  A      Bit 7 ins Carry
F457  B0 03      BCS $F45C   verzweige wenn DSR vorhanden
F459  20 0D F0   JSR $F00D   Status für DSR setzen
F45C  AD 9B 02   LDA $029B   Anfang RS-232 Eingabepuffer
F45F  8D 9C 02   STA $029C   mit Ende des Eingabepuffers
                             gleichsetzen
F462  AD 9E 02   LDA $029E   Anfang des RS-232
                             Ausgabepuffers
F465  8D 9D 02   STA $029D   mit Ende des Ausgabepuffers
                             gleichsetzen
F468  20 27 FE   JSR $FE27   Memory Top holen
F46B  A5 F8      LDA $F8     HIGH-Byte des Zeigers auf
                             RS-232 Eingabepuffer
F46D  D0 05      BNE $F474   ungleich Null, so Eingabe-
                             puffer bereits angelegt
F46F  88         DEY         HIGH-Byte Memory Top -1
F470  84 F8      STY $F8     als Zeiger für RS-232
                             Eingabepuffer speichern
F472  86 F7      STX $F7     LOW-Byte Memory Top als LOW-
                             Byte Eingabepuffer setzen
F474  A5 FA      LDA $FA     HIGH-Byte des Zeigers auf
                             RS-232 Ausgabepuffer
F476  D0 05      BNE $F47D   verzweige wenn Ausgabepuffer
                             bereits angelegt
F478  88         DEY         HIGH-Byte des Memory Top -1
F479  84 FA      STY $FA     und als Zeiger für RS-232
                             Ausgabepuffer setzen
F47B  86 F9      STX $F9     LOW-Byte Memory Top als LOW-
                             Byte Ausgabepuffer setzen

Einsprung von $F2C5

F47D  38         SEC         Carry =1 (Fehlerkennzeichen)
```

```
F47E   A9 F0      LDA #$F0    Flag für Puffer schützen/
                              freigeben setzen
F480   4C 2D FE   JMP $FE2D   Memory-Top neu setzen

**************************  CIAs nach RS 232 rücksetzen

Einsprung von $F2AF, $F409

F483   A9 7F      LDA #$7F    Bitwert für alle
F485   8D 0D DD   STA $DD0D   NMIs blockieren setzen
F488   A9 06      LDA #$06    Bit 1 und 2 Ausgang
F48A   8D 03 DD   STA $DD03   PORT B Richtung
F48D   8D 01 DD   STA $DD01   PORT A Richtung
F490   A9 04      LDA #$04    Bit 2 setzen
F492   0D 00 DD   ORA $DD00   Bit 2 = TXD
F495   8D 00 DD   STA $DD00   Ausgeben
F498   A0 00      LDY #$00    RS-232
F49A   8C A1 02   STY $02A1   NMI-Flag löschen
F49D   60         RTS         Rücksprung

**************************  LOAD - Routine

Einsprung von $FFD5

F49E   86 C3      STX $C3     Startadresse
F4A0   84 C4      STY $C4     speichern
F4A2   6C 30 03   JMP ($0330) JMP $F4A5    LOAD-Vektor

Einsprung von $F4A2

F4A5   85 93      STA $93     Load/Verify Flag
F4A7   A9 00      LDA #$00    Status
F4A9   85 90      STA $90     löschen
F4AB   A5 BA      LDA $BA     Geräteadresse laden
F4AD   D0 03      BNE $F4B2   ungleich Null, dann weiter
F4AF   4C 13 F7   JMP $F713   'ILLEGAL DEVICE NUMBER'
F4B2   C9 03      CMP #$03    vergleiche mit Code für
                              Bildschirm
F4B4   F0 F9      BEQ $F4AF   verzweige wenn ja, Fehler
F4B6   90 7B      BCC $F533   kleiner 3, dann vom Band
```

```
**************************** IEC-Load
F4B8  A4 B7     LDY $B7     Länge des Filenamens laden
F4BA  D0 03     BNE $F4BF   ungleich Null, dann ok
F4BC  4C 10 F7  JMP $F710   'MISSING FILENAME'
F4BF  A6 B9     LDX $B9     Sekundäradresse laden
F4C1  20 AF F5  JSR $F5AF   'SEARCHING FOR' (filename)
F4C4  A9 60     LDA #$60    Sekundäradresse Null laden
                            (für OPEN)
F4C6  85 B9     STA $B9     und speichern
F4C8  20 D5 F3  JSR $F3D5   File auf IEC-Bus eröffnen
F4CB  A5 BA     LDA $BA     Gerätenummer laden
F4CD  20 09 ED  JSR $ED09   und TALK senden
F4D0  A5 B9     LDA $B9     Sekundäradresse laden
F4D2  20 C7 ED  JSR $EDC7   und senden
F4D5  20 13 EE  JSR $EE13   Byte vom IEC-Bus holen
F4D8  85 AE     STA $AE     als Startadresse LOW spei
                            chern
F4DA  A5 90     LDA $90     Status laden
F4DC  4A        LSR  A      Bit 1
F4DD  4A        LSR  A      ins Carry schieben
F4DE  B0 50     BCS $F530   falls gesetzt, dann Time out
                            (Fehler)
F4E0  20 13 EE  JSR $EE13   Startadresse HIGH holen
F4E3  85 AF     STA $AF     und speichern
F4E5  8A        TXA         Sekundäradresse laden
F4E6  D0 08     BNE $F4F0   verzweige falls ungleich Null
F4E8  A5 C3     LDA $C3     Startadresse LOW laden
F4EA  85 AE     STA $AE     und speichern
F4EC  A5 C4     LDA $C4     Startadresse HIGH laden
F4EE  85 AF     STA $AF     und speichern
F4F0  20 D2 F5  JSR $F5D2   'LOADING'/'VERIFYING'
                            ausgeben
F4F3  A9 FD     LDA #$FD    Time-out
F4F5  25 90     AND $90     Bit
F4F7  85 90     STA $90     löschen
F4F9  20 E1 FF  JSR $FFE1   Stop-Taste abfragen
F4FC  D0 03     BNE $F501   nicht gedrückt, dann weiter
F4FE  4C 33 F6  JMP $F633   File schließen
F501  20 13 EE  JSR $EE13   Programmbyte vom Bus holen
```

```
F504  AA        TAX          Akku in X-REG retten
F505  A5 90     LDA $90      Status testen
F507  4A        LSR  A       Time-out
F508  4A        LSR  A       Bit ins Carry schieben
F509  B0 E8     BCS $F4F3    falls Fehler, dann abbrechen
F50B  8A        TXA          ansonsten Akku wiederholen
F50C  A4 93     LDY $93      Load/Verify Flag testen
F50E  F0 0C     BEQ $F51C    gleich Null, dann LOAD
F510  A0 00     LDY #$00     Zähler auf Null setzen
F512  D1 AE     CMP ($AE),Y  Verify, Vergleich
F514  F0 08     BEQ $F51E    verzweige falls gleich
F516  A9 10     LDA #$10     Bit 4 für Status setzen
F518  20 1C FE  JSR $FE1C    Status setzen
F51B  2C        .BYTE $2C    Skip nach $F51E
F51C  91 AE     STA ($AE),Y  Byte abspeichern
F51E  E6 AE     INC $AE      LOW-Byte der Adresse erhöhen
F520  D0 02     BNE $F524    verzweige falls kein Übertrag
F522  E6 AF     INC $AF      ansonsten HIGH-Byte erhöhen
F524  24 90     BIT $90      Status prüfen
F526  50 CB     BVC $F4F3    verzweige wenn noch kein EOI
F528  20 EF ED  JSR $EDEF    UNTALK senden
F52B  20 42 F6  JSR $F642    File schließen
F52E  90 79     BCC $F5A9    vezweige wenn kein Fehler
F530  4C 04 F7  JMP $F704    'FILE NOT FOUND'

****************************
F533  4A        LSR  A       Gerätenummer feststellen
F534  B0 03     BCS $F539    eins (Band) , dann weiter
F536  4C 13 F7  JMP $F713    RS 232, 'ILLEGAL DEVICE
                             NUMBER'
F539  20 D0 F7  JSR $F7D0    Bandpuffer Startadresse holen
F53C  B0 03     BCS $F541    verzweige wenn HIGH-Byte der
                             Bandpufferstartadresse größer/
                             gleich 2
F53E  4C 13 F7  JMP $F713    sonst 'ILLEGAL DEVICE NUMBER'
F541  20 17 F8  JSR $F817    wartet auf Play-Taste
F544  B0 68     BCS $F5AE    STOP-Taste, dann Abbruch
F546  20 AF F5  JSR $F5AF    'SEARCHING' ( 'for name')
                             ausgeben
F549  A5 B7     LDA $B7      Länge des Filenamens laden
```

```
F54B  F0 09     BEQ $F556   verzweige wenn Null
F54D  20 EA F7  JSR $F7EA   gewünschten Bandheader suchen
F550  90 0B     BCC $F55D   verzweige wenn gefunden
F552  F0 5A     BEQ $F5AE   STOP-Taste, dann Abbruch
F554  B0 DA     BCS $F530   EOT, dann 'FILE NOT FOUND'
F556  20 2C F7  JSR $F72C   nächsten Bandheader suchen
F559  F0 53     BEQ $F5AE   STOP-Taste, dann Abbruch
F55B  B0 D3     BCS $F530   'EOT', dann 'FILE NOT FOUND'
F55D  A5 90     LDA $90     Status holen
F55F  29 10     AND #$10    EOF-Bit ausblenden
F561  38        SEC         Carry =1 (Fehlerkennzeichen)
F562  D0 4A     BNE $F5AE   verzweige falls Fehler
F564  E0 01     CPX #$01    Header-Typ  1 = BASIC-
                            Programm (verschiebbar)
F566  F0 11     BEQ $F579   verzweige wenn Header-Typ =1
F568  E0 03     CPX #$03    3 = Maschinen-Programm
                            (absolut)
F56A  D0 DD     BNE $F549   verzweige wenn nicht 3
                            (falscher Header)
F56C  A0 01     LDY #$01    Zeiger setzen
F56E  B1 B2     LDA ($B2),Y LOW-Byte Startadresse holen
F570  85 C3     STA $C3     und speichern
F572  C8        INY         Zeiger erhöhen
F573  B1 B2     LDA ($B2),Y HIGH-Byte Startadresse holen
F575  85 C4     STA $C4     und speichern
F577  B0 04     BCS $F57D   unbedingter Sprung
F579  A5 B9     LDA $B9     Sekundär-Adresse
F57B  D0 EF     BNE $F56C   ungleich Null, dann nicht
                            verschiebbar laden
F57D  A0 03     LDY #$03    Zeiger setzen
F57F  B1 B2     LDA ($B2),Y LOW-Byte der Endadresse+1 des
                            Programms holen
F581  A0 01     LDY #$01    Zeiger auf LOW-Byte Anfangs
                            adresse setzen
F583  F1 B2     SBC ($B2),Y von Endadresse subtrahieren
F585  AA        TAX         Ergebnis ins X-REG schieben
F586  A0 04     LDY #$04    Zeiger auf HIGH-Byte der
                            Endadresse setzen
F588  B1 B2     LDA ($B2),Y Endadresse holen
```

```
F58A   A0 02      LDY #$02     Zeiger auf Startadresse
                               setzen
F58C   F1 B2      SBC ($B2),Y  und von Endadresse subtrahie
                               ren
F58E   A8         TAY          Ergebnis ins Y-REG schieben
F58F   18         CLC          Carry für Addition löschen
F590   8A         TXA          LOW-Byte der Programmlänge
                               in Akku schieben
F591   65 C3      ADC $C3      mit LOW-Byte der Anfangs
                               adresse addieren
F593   85 AE      STA $AE      als LOW-Byte der Endadresse
                               speichern
F595   98         TYA          HIGH-Byte der Programmlänge
                               in Akku schieben
F596   65 C4      ADC $C4      mit HIGH-Byte Anfangsadresse
                               addieren
F598   85 AF      STA $AF      als HIGH-Byte Endadresse
                               speichern
F59A   A5 C3      LDA $C3      Startadresse
F59C   85 C1      STA $C1      nach $C1
F59E   A5 C4      LDA $C4      und $C2
F5A0   85 C2      STA $C2      bringen
F5A2   20 D2 F5   JSR $F5D2    'LOADING' / 'VERIFYING'
                               ausgeben
F5A5   20 4A F8   JSR $F84A    Programm vom Band laden
F5A8   24         .BYTE $24    Skip nach $F5AA
F5A9   18         CLC          Carry =0 (ok Kennzeichen)
F5AA   A6 AE      LDX $AE      Endadresse
F5AC   A4 AF      LDY $AF      nach X/Y
F5AE   60         RTS          Rücksprung

**************************     'SEARCHING FOR' (Filename)
                               ausgeben

Einsprung von $F39E, $F4C1, $F546

F5AF   A5 9D      LDA $9D      Direkt-Modus-Flag laden
F5B1   10 1E      BPL $F5D1    verzweige wenn Bit 7 =0
                               (Programm-Mode)
F5B3   A0 0C      LDY #$0C     Offset für 'SEARCHING'
```

```
F5B5   20 2F F1   JSR $F12F    Meldung ausgeben
F5B8   A5 B7      LDA $B7      Länge des Filenamens
F5BA   F0 15      BEQ $F5D1    gleich Null, dann fertig
F5BC   A0 17      LDY #$17     Offset für 'FOR'
F5BE   20 2F F1   JSR $F12F    Meldung ausgeben

Einsprung von $F698

F5C1   A4 B7      LDY $B7      Länge des Filenamens
F5C3   F0 0C      BEQ $F5D1    gleich Null, dann fertig
F5C5   A0 00      LDY #$00     Zähler setzen
F5C7   B1 BB      LDA ($BB),Y  Filenamen holen
F5C9   20 D2 FF   JSR $FFD2    und ausgeben
F5CC   C8         INY          Zähler erhöhen
F5CD   C4 B7      CPY $B7      mit Länge des Filenamens ver-
                               gleichen
F5CF   D0 F6      BNE $F5C7    verzweige wenn noch nicht
                               alle Buchstaben
F5D1   60         RTS          Rücksprung

**************************     'LOADING/VERIFYING' ausgeben

Einsprung von $F4F0, $F5A2

F5D2   A0 49      LDY #$49     Offset für 'LOADING'
F5D4   A5 93      LDA $93      Load/Verify-Flag laden
F5D6   F0 02      BEQ $F5DA    Load wenn 0, dann ausgeben
F5D8   A0 59      LDY #$59     sonst Offset für 'VERIFYING'
F5DA   4C 2B F1   JMP $F12B    Meldung ausgeben, Rücksprung

**************************     SAVE - Routine

Einsprung von $FFD8

F5DD   86 AE      STX $AE      LOW-Byte der Endadresse
                               speichern
F5DF   84 AF      STY $AF      High-Byte der Endadresse
                               speichern
F5E1   AA         TAX          Zeiger auf Anfangsadress-
                               tabelle ins X-REG schieben
```

```
F5E2   B5 00      LDA $00,X   LOW-Byte der Startadresse
F5E4   85 C1      STA $C1     holen und speichern
F5E6   B5 01      LDA $01,X   HIGH-Byte der Startadresse
F5E8   85 C2      STA $C2     holen und speichern
F5EA   6C 32 03   JMP ($0332) SAVE-Vektor,  JMP $F5ED

Einsprung von $F5EA

F5ED   A5 BA      LDA $BA     Geräteadresse laden
F5EF   D0 03      BNE $F5F4   verzweige wenn nicht gleich 0
F5F1   4C 13 F7   JMP $F713   sonst 'ILLEGAL DEVICE NUMBER'
F5F4   C9 03      CMP #$03    mit Code für Bildschirm
                              vergleichen
F5F6   F0 F9      BEQ $F5F1   wenn Bildschirm, dann Fehler
F5F8   90 5F      BCC $F659   kleiner 3, dann verzweige

**************************    Speichern auf IEC-Bus
F5FA   A9 61      LDA #$61    Sekundäradresse 1
F5FC   85 B9      STA $B9     setzen
F5FE   A4 B7      LDY $B7     Länge des Filenamens laden
F600   D0 03      BNE $F605   ungleich Null, dann ok
F602   4C 10 F7   JMP $F710   sonst 'MISSING FILENAME'
F605   20 D5 F3   JSR $F3D5   Filenamen auf IEC-Bus
F608   20 8F F6   JSR $F68F   'SAVING' ausgeben
F60B   A5 BA      LDA $BA     Geräteadresse laden
F60D   20 0C ED   JSR $ED0C   und LISTEN senden
F610   A5 B9      LDA $B9     Sekundäradresse laden
F612   20 B9 ED   JSR $EDB9   und für LISTEN senden
F615   A0 00      LDY #$00    Zähler auf Null setzen
F617   20 8E FB   JSR $FB8E   Startadresse nach $AC/$AD
F61A   A5 AC      LDA $AC     Startadresse LOW-
F61C   20 DD ED   JSR $EDDD   Byte senden
F61E   A5 AD      LDA $AD     und HIGH-
F621   20 DD ED   JSR $EDDD   senden
F624   20 D1 FC   JSR $FCD1   Endadresse schon erreicht ?
F627   B0 16      BCS $F63F   ja, dann fertig
F629   B1 AC      LDA ($AC),Y Programmbyte laden
F62B   20 DD ED   JSR $EDDD   auf IEC-Bus ausgeben
```

```
F62E   20 E1 FF   JSR $FFE1   STOP-Taste abfragen
F631   D0 07      BNE $F63A   nicht gedrückt, dann
                              weitermachen

Einsprung von $F4FE

F633   20 42 F6   JSR $F642   IEC-Bus Kanal schließen
F636   A9 00      LDA #$00    Kennzeichnung für 'BREAK'
F638   38         SEC         Carry =1 (Fehlerkennzeichen)
F639   60         RTS         Rücksprung

F63A   20 DB FC   JSR $FCDB   laufende Adresse erhöhen
F63D   D0 E5      BNE $F624   unbedingter Sprung
F63F   20 FE ED   JSR $EDFE   UNLISTEN senden

**************************   File auf IEC-Bus schließen

Einsprung von $F2EE, $F52B, $F633

F642   24 B9      BIT $B9     Sekundäradresse testen
F644   30 11      BMI $F657   verzweige falls keine
                              Sekundäradresse
F646   A5 BA      LDA $BA     Geräteadresse laden
F648   20 0C ED   JSR $ED0C   und LISTEN senden
F64B   A5 B9      LDA $B9     Sekundäradresse laden
F64D   29 EF      AND #$EF    Sekundäradresse
F64F   09 E0      ORA #$E0    für CLOSE berechnen
F651   20 B9 ED   JSR $EDB9   und ausgeben

Einsprung von $F406

F654   20 FE ED   JSR $EDFE   UNLISTEN senden
F657   18         CLC         Carry =0 (ok Kennzeichen)
F658   60         RTS         Rücksprung
F659   4A         LSR  A      Bit 0 ins Carry schieben
F65A   B0 03      BCS $F65F   falls gesetzt, dann zu Band
F65C   4C 13 F7   JMP $F713   sonst RS-232, 'ILLEGAL DIVICE
                              NUMBER'
F65F   20 D0 F7   JSR $F7D0   Bandpuffer Startadresse holen
```

```
F662   90 8D      BCC $F5F1   falls HIGH-Byte der Band
                              Pufferstartadresse kleiner 2
                              dann 'ILLEGAL DEVICE NUMBER'
F664   20 38 F8   JSR $F838   wartet auf Record & Play-
                              Taste
F667   B0 25      BCS $F68E   STOP, dann Abbruch
F669   20 8F F6   JSR $F68F   'SAVING' (Name) ausgeben
F66C   A2 03      LDX #$03    Header-Typ 3 = Maschinen
                              programm (absolut)
F66E   A5 B9      LDA $B9     Sekundäradresse laden
F670   29 01      AND #$01    Bit 0 gesetzt  (1 oder 3)
F672   D0 02      BNE $F676   falls ja, dann Maschinen
                              programm
F674   A2 01      LDX #$01    Header-Typ 1 = BASIC-
                              Programm (verschiebbar)
F676   8A         TXA         Header in Akku schieben
F677   20 6A F7   JSR $F76A   Header auf Band schreiben
F67A   B0 12      BCS $F68E   Aussprung bei Stop-Taste
F67C   20 67 F8   JSR $F867   Programm auf Band schreiben
F67F   B0 0D      BCS $F68E   Aussprung bei Stop-Taste
F681   A5 B9      LDA $B9     Sekundäradresse laden
F683   29 02      AND #$02    Bit 1 gesetzt  (2 oder 3)
F685   F0 06      BEQ $F68D   falls nicht, dann fertig
F687   A9 05      LDA #$05    EOT Kontrollbyte
F689   20 6A F7   JSR $F76A   Block auf Band schreiben
F68C   24         .BYTE $24   Skip zu $F68E
F68D   18         CLC         Carry =0 (ok Kennzeichen)
F68E   60         RTS         Rücksprung

**************************   'SAVING' ausgeben

Einsprung von $F608, $F669

F68F   A5 9D      LDA $9D     Flag für Direktmodus laden
F691   10 FB      BPL $F68E   Bit 7 gelöscht, dann
                              Programm-Mode
F693   A0 51      LDY #$51    Offset für 'SAVING'
F695   20 2F F1   JSR $F12F   Meldung ausgeben
F698   4C C1 F5   JMP $F5C1   Filenamen ausgeben,
                              Rücksprung
```

```
**************************  UDTIM  Time erhöhen und
                            STOP-Taste abfragen

Einsprung von $FFEA

F69B   A2 00      LDX #$00    X-REG auf Null setzen
F69D   E6 A2      INC $A2     Sekundenzeiger erhöhen
F69F   D0 06      BNE $F6A7   verzweige falls kein Überlauf
F6A1   E6 A1      INC $A1     Minutenzeiger erhöhen
F6A3   D0 02      BNE $F6A7   verzweige falls kein Überlauf
F6A5   E6 A0      INC $A0     Stundenzeiger erhöhen
F6A7   38         SEC         Carry für Subtraktion löschen
F6A8   A5 A2      LDA $A2     Stundenzeiger laden
F6AA   E9 01      SBC #$01    feststellen
F6AC   A5 A1      LDA $A1     ob
F6AE   E9 1A      SBC #$1A    24
F6B0   A5 A0      LDA $A0     Stunden
F6B2   E9 4F      SBC #$4F    erreicht
F6B4   90 06      BCC $F6BC   falls kleiner, dann verzweige
F6B6   86 A0      STX $A0     alle
F6B8   86 A1      STX $A1     Zeiger
F6BA   86 A2      STX $A2     auf Null setzen

**************************  Abfrage auf STOP-Taste direkt
                            vom Port

Einsprung von $F8CA, $FE5E

F6BC   AD 01 DC   LDA $DC01   Port B laden
F6BF   CD 01 DC   CMP $DC01   und
F6C2   D0 F8      BNE $F6BC   entprellen
F6C4   AA         TAX         Wert ins X-REG schieben
F6C5   30 13      BMI $F6DA   verzweige falls STOP-Taste
                              nicht gedrückt
F6C7   A2 BD      LDX #$BD    Bitmuster zur Abrage der
                              Reihe mit SHIFT-Tasten
F6C9   8E 00 DC   STX $DC00   in Port A schreiben
F6CC   AE 01 DC   LDX $DC01   Port B laden
F6CF   EC 01 DC   CPX $DC01   und
```

```
F6D2   D0 F8      BNE $F6CC   entprellen
F6D4   8D 00 DC   STA $DC00   Akku in Port A schreiben
F6D7   E8         INX         inhalt von Port B erhöhen
F6D8   D0 02      BNE $F6DC   verzweige falls ungleich Null
                              (SHIFT-Taste gedrückt)
F6DA   85 91      STA $91     Flag für Stop-Taste setzen
F6DC   60         RTS         Rücksprung

**************************    TIME holen

Einsprung von $FFDE

F6DD   78         SEI         Interrupt verhindern um Uhr
                              anzuhalten
F6DE   A5 A2      LDA $A2     Stunden
F6E0   A6 A1      LDX $A1     Minuten
F6E2   A4 A0      LDY $A0     Sekunden holen

**************************    TIME setzen

Einsprung von $FFDB

F6E4   78         SEI         Interrupt verhindern um Uhr
                              anzuhalten
F6E5   85 A2      STA $A2     Stunden
F6E7   86 A1      STX $A1     Minuten
F6E9   84 A0      STY $A0     Sekunden schreiben
F6EB   58         CLI         Interrupt wieder ermöglichen
F6EC   60         RTS         Rücksprung

**************************    STOP-Taste abfragen

Einsprung von $FFE1

F6ED   A5 91      LDA $91     STOP-Flag laden
F6EF   C9 7F      CMP #$7F    auf Code für STOP testen
F6F1   D0 07      BNE $F6FA   verzweige falls nicht
F6F3   08         PHP         Statusregister retten
F6F4   20 CC FF   JSR $FFCC   Ein-Ausgabe zurücksetzen
                              CLRCH
```

```
F6F7   85 C6      STA $C6     Anzahl der gedrückten Tasten
F6F9   28         PLP         Statusregister holen
F6FA   60         RTS         Rücksprung

**************************    Meldungen des Betriebs
                              systems ausgeben

Einsprung von $F35F

F6FB   A9 01      LDA #$01    'TOO MANY FILES'
F6FD   2C         .BYTE $2C   Skip zu $F700

Einsprung von $F356

F6FE   A9 02      LDA #$02    'FILE OPEN'
F700   2C         .BYTE $2C   Skip zu $F703

Einsprung von $F213, $F255

F701   A9 03      LDA #$03    'FILE NOT OPEN'
F703   2C         .BYTE $2C   Skip zu $F706

Einsprung von $F3AC, $F530

F704   A9 04      LDA #$04    'FILE NOT FOUND'
F706   2C         .BYTE $2C   Skip zu $F709

Einsprung von $F24D, $F28E, $F3F3

F707   A9 05      LDA #$05    'DIVICE NOT PRESENT'
F709   2C         .BYTE $2C   Skip zu $F70C

Einsprung von $F230, $F34E

F70A   A9 06      LDA #$06    'NOT INPUT FILE'
F70C   2C         .BYTE $2C   Skip zu $F70F
```

```
Einsprung von $F25F

F70D   A9 07      LDA #$07    'NOT OUTPUT FILE'
F70F   2C         .BYTE $2C   Skip zu $F712

Einsprung von $F4BC, $F602

F710   A9 08      LDA #$08    'MISSING FILENAME'
F712   2C         .BYTE $2C   Skip zu $F715

Einsprung von $F390, $F4AF, $F536, $F53E, $F5F1, $F65C

F713   A9 09      LDA #$09    'ILLEGAL DEVICE NUMBER'
F715   48         PHA         Fehlernummer merken
F716   20 CC FF   JSR $FFCC   Ein-Ausgabe zurücksetzen
                              CLRCH
F719   A0 00      LDY #$00
F71B   24 9D      BIT $9D     Flag auf Direkt-Mode testen
F71D   50 0A      BVC $F729   nicht gesetzt, dann übergehen
F71F   20 2F F1   JSR $F12F   'I/O ERROR #' ausgeben
F722   68         PLA         Fehlernummer holen
F723   48         PHA         und wieder merken
F724   09 30      ORA #$30    nach ASCII wandeln
F726   20 D2 FF   JSR $FFD2   und ausgeben
F729   68         PLA         Fehlernummer holen
F72A   38         SEC         Carry =1 (Fehlerkennzeichen)
F72B   60         RTS         Rücksprung

**************************    Programm Header vom Band
                              lesen

Einsprung von $F3AF, $F556, $F7EA

F72C   A5 93      LDA $93     Load/Verify Flag laden
F72E   48         PHA         und retten
F72F   20 41 F8   JSR $F841   Block vom Band lesen
F732   68         PLA         L/V Flag wiederholen
F733   85 93      STA $93     und speichern
F735   B0 32      BCS $F769   Fehler, dann beenden
F737   A0 00      LDY #$00    Zähler auf Null stellen
```

```
F739   B1 B2      LDA ($B2),Y Header-Typ testen
F73B   C9 05      CMP #$05    EOT ?
F73D   F0 2A      BEQ $F769   verzweige falls ja
F73F   C9 01      CMP #$01    BASIC-Programm ?
F741   F0 08      BEQ $F74B   verzweige falls ja
F743   C9 03      CMP #$03    Maschinenprogramm ?
F745   F0 04      BEQ $F74B   verzweige falls ja
F747   C9 04      CMP #$04    Daten-Header ?
F749   D0 E1      BNE $F72C   kein Header gefunden, dann
                               erneut suchen
F74B   AA         TAX         Kennzeichen merken
F74C   24 9D      BIT $9D     Direktmodus ?
F74E   10 17      BPL $F767   nein, dann weiter
F750   A0 63      LDY #$63    Offset für 'FOUND'
F752   20 2F F1   JSR $F12F   Meldung ausgeben
F755   A0 05      LDY #$05    Zeiger auf Filenamen
F757   B1 B2      LDA ($B2),Y Filenamen holen
F759   20 D2 FF   JSR $FFD2   und ausgeben
F75C   C8         INY         Zeiger erhöhen
F75D   C0 15      CPY #$15    schon alle Buchstaben
F75F   D0 F6      BNE $F757   verzweige wenn nein
F761   A5 A1      LDA $A1     Akku mit mittelwertigem
                               Time-Byte laden
F763   20 E0 E4   JSR $E4E0   wartet auf Commodore-Taste
                               oder Zeitschleife
F766   EA         NOP         no operation
F767   18         CLC         Carry =0 (ok Kennzeichen)
F768   88         DEY         Y-REG auf $FF zur Kennzeich
                               nung, daß kein EOT
F769   60         RTS         Rücksprung

**************************   Header generieren und auf
                               Band schreiben

Einsprung von $F2E8, $F3BF, $F677, $F689

F76A   85 9E      STA $9E     Header-Typ speichern
F76C   20 D0 F7   JSR $F7D0   Bandpufferadresse holen
F76F   90 5E      BCC $F7CF   verzweige falls Adresse
                               ungültig
```

```
F771  A5 C2     LDA $C2      Startadresse
F773  48        PHA          laden
F774  A5 C1     LDA $C1      und in
F776  48        PHA          Stack schreiben
F777  A5 AF     LDA $AF      Endadresse
F779  48        PHA          laden
F77A  A5 AE     LDA $AE      und in
F77C  48        PHA          Stack schreiben
F77D  A0 BF     LDY #$BF     Pufferlänge für Schleife
                             holen
F77F  A9 20     LDA #$20     Code für ' ' laden
F781  91 B2     STA ($B2),Y  und speichern
F783  88        DEY          Zähler verringern
F784  D0 FB     BNE $F781    verzweige falls Puffer noch
                             nicht alles gelöscht
F786  A5 9E     LDA $9E      gespeicherten Header-Typ
                             holen
F788  91 B2     STA ($B2),Y  und in Puffer schreiben
F78A  C8        INY          Zähler erhöhen
F78B  A5 C1     LDA $C1      Startadresse LOW holen
F78D  91 B2     STA ($B2),Y  und in Puffer schreiben
F78F  C8        INY          Zähler erhöhen
F790  A5 C2     LDA $C2      Startadesse HIGH holen
F792  91 B2     STA ($B2),Y  und in Puffer schreiben
F794  C8        INY          Zähler erhöhen
F795  A5 AE     LDA $AE      Endadresse LOW holen
F797  91 B2     STA ($B2),Y  und in Puffer schreiben
A799  C8        INY          Zähler erhöhen
F79A  A5 AF     LDA $AF      Endadresse HIGH holen
F79C  91 B2     STA ($B2),Y  und in Puffer schreiben
F79E  C8        INY          Zähler erhöhen
F79F  84 9F     STY $9F      Zähler speichern
F7A1  A0 00     LDY #$00     Zähler für Filenamen auf Null
                             setzen
F7A3  84 9E     STY $9E      und speichern
F7A5  A4 9E     LDY $9E      Zähler holen
F7A7  C4 B7     CPY $B7      und mit Länge des Filenamens
                             vergleichen
F7A9  F0 0C     BEQ $F7B7    verzweige falls alle Buchsta-
                             ben geholt
```

```
F7AB   B1 BB      LDA ($BB),Y Filenamen holen
F7AD   A4 9F      LDY $9F     Pufferzeiger laden
F7AF   91 B2      STA ($B2),Y und Zeichen in Puffer schrei-
                              ben
F7B1   E6 9E      INC $9E     Zähler für Filenamen erhöhen
F7B3   E6 9F      INC $9F     Zeiger auf Bandpuffer erhöhen
F7B5   D0 EE      BNE $F7A5   unbedingter Sprung

F7B7   20 D7 F7   JSR $F7D7   Start- und Endadresse auf
                              Bandpuffer holen
F7BA   A9 69      LDA #$69
F7BC   85 AB      STA $AB     Checksumme für Header bzw.
                              Datenblock = $69
F7BE   20 6B F8   JSR $F86B   Block auf Band schreiben
F7C1   A8         TAY         Akku retten
F7C2   68         PLA         Endadresse
F7C3   85 AE      STA $AE     vom Stack
F7C5   68         PLA         holen und
F7C6   85 AF      STA $AF     in $AE/$AF speichern
F7C8   68         PLA         Startadresse
F7C9   85 C1      STA $C1     vom Stack
F7CB   68         PLA         holen und
F7CC   85 C2      STA $C2     in $C1/C2 speichern
F7CE   98         TYA         Akku wiederholen
F7CF   60         RTS         Rücksprung

**************************    Bandpuffer Startadresse holen
                              und prüfen ob gültig

Einsprung von $F2CE, $F38B, $F539 ,$F65F, $F76C, $F7D7
             $F80D

F7D0   A6 B2      LDX $B2     Anfang Bandpuffer LOW in X
F7D2   A4 B3      LDY $B3     Anfang Bandpuffer HIGH in Y
F7D4   C0 02      CPY #$02    Adresse kleiner $200 ?
F7D6   60         RTS         Rücksprung
```

```
**************************  Bandpufferendadresse = Puf-
                            ferstartadresse + $C0 (192)

Einsprung von $F7B7, $F847, $F864

F7D7  20 D0 F7  JSR $F7D0   BandpufferaAdresse holen
F7DA  8A        TXA         Pufferanfang LOW in Akku
F7DB  85 C1     STA $C1     und speichern
F7DD  18        CLC         Carry für Addition löschen
F7DE  69 C0     ADC #$C0    Endadresse = Startadresse +
                            Länge $C0 (192)
F7E0  85 AE     STA $AE     und Endadresse speichern
F7E2  98        TYA         Pufferanfang HIGH in Akku
F7E3  85 C2     STA $C2     und speichern
F7E5  69 00     ADC #$00    mit Übertrag addieren
F7E7  85 AF     STA $AF     und speichern
F7E9  60        RTS         Rücksprung

**************************  Bandheader nach Namen suchen

Einsprung von $F3A5, $F54D

F7EA  20 2C F7  JSR $F72C   nächsten Bandheader suchen
F7ED  B0 1D     BCS $F80C   verzweige falls EOT (fertig)
F7EF  A0 05     LDY #$05    Offset für Filenamen im
                            Header
F7F1  84 9F     STY $9F     und speichern
F7F3  A0 00     LDY #$00    Zähler für Länge des Filena-
                            mens auf Null setzen
F7F5  84 9E     STY $9E     und Zähler speichern
F7F7  C4 B7     CPY $B7     mit Länge des gesuchten
                            Namens vergleichen
F7F9  F0 10     BEQ $F80B   gleich, dann gefunden
F7FB  B1 BB     LDA ($BB),Y Buchstaben des Filenamens
F7FD  A4 9F     LDY $9F     Position im Header laden
F7FF  D1 B2     CMP ($B2),Y mit Filenamen im Header
                            vergleichen
F801  D0 E7     BNE $F7EA   verzweige falls ungleich,
                            dann nächsten Header testen
F803  E6 9E     INC $9E     Zähler für Filenamen erhöhen
```

```
F805   E6 9F      INC $9F     Zeiger auf Position im Header
                              erhöhen
F807   A4 9E      LDY $9E     Zähler für Filenamen laden
F809   D0 EC      BNE $F7F7   unbedingter Sprung
F80B   18         CLC         Carry =0 (ok Kennzeichen)
F80C   60         RTS         Rücksprung

**************************** Bandpufferzeiger erhöhen

Einsprung von $F199, $F1E5

F80D   20 D0 F7   JSR $F7D0   Bandpufferadresse holen
F810   E6 A6      INC $A6     Zeiger erhöhen
F812   A4 A6      LDY $A6     und laden um
F814   C0 C0      CPY #$C0    mit Maximalwert (192) zu
                              vergleichen
F816   60         RTS         Rücksprung

**************************** Wartet auf Bandtaste

Einsprung von $F399, $F541, $F84A

F817   20 2E F8   JSR $F82E   fragt BandtTaste ab
F81A   F0 1A      BEQ $F836   gedrückt, dann fertig
F81C   A0 1B      LDY #$1B    Offset für 'PRESS PLAY ON
                              TAPE'
F81E   20 2F F1   JSR $F12F   und ausgeben
F821   20 D0 F8   JSR $F8D0   testet auf STOP-Taste
F824   20 2E F8   JSR $F82E   fragt BandtTaste ab
F827   D0 F8      BNE $F821   nicht gedrückt so erneut
                              abfragen
F829   A0 6A      LDY #$6A    Offset für 'OK'
F82B   4C 2F F1   JMP $F12F   und ausgeben, Rücksprung

**************************** Abfrage ob Band-Taste
                              gedrückt

Einsprung von $F817 ,$F824, $F838

F82E   A9 10      LDA #$10    Bit 4 testen
```

```
F830   24 01      BIT $01     mit Port vergleichen
F832   D0 02      BNE $F836   verzweige wenn Bandtaste
                               nicht gedrückt
F834   24 01      BIT $01     nochmal abfragen (Entprellen)
F836   18         CLC         Carry =0 (ok Kennzeichen)
F837   60         RTS         Rücksprung

**************************   Wartet auf Bandtaste für
                               Schreiben

Einsprung von $F3B8, $F664, $F86B

F838   20 2E F8   JSR $F82E   fragt Bandtaste ab
F83B   F0 F9      BEQ $F836   gedrückt, dann fertig
F83D   A0 2E      LDY #$2E    Offset für 'PRESS RECORD &
                               PLAY ON TAPE'
F83F   D0 DD      BNE $F81E   unbedingter Sprung

**************************   Block vom Band lesen

Einsprung von $F19E, $F72F

F841   A9 00      LDA #$00    Status
F843   85 90      STA $90     und Verify-Flag
F845   85 93      STA $93     löschen
F847   20 D7 F7   JSR $F7D7   Bandpufferadresse holen

**************************   Programm vom Band laden

Einsprung von $F5A5

F84A   20 17 F8   JSR $F817   wartet auf Play-Taste
F84D   B0 1F      BCS $F86E   STOP-Taste gedrückt ?
F84F   78         SEI         Interrupt verhindern
F850   A9 00      LDA #$00    Arbeitsspeicher für IRQ-
                               Routine löschen
F852   85 AA      STA $AA     Eingabebytespeicher (read)
F854   85 B4      STA $B4     Band Hilfszeiger
F856   85 B0      STA $B0     Kassetten Zeitkonstante
F858   85 9E      STA $9E     Korrekturzähler Pass 1
```

```
F85A   85 9F      STA $9F     Korrekturzähler Pass 2
F85C   85 9C      STA $9C     Flag für Byte emfngen
F85E   A9 90      LDA #$90    Bitwert IRQ an Pin 'Flag'
F860   A2 0E      LDX #$0E    Nummer des IRQ-Vektors, $F92C
F862   D0 11      BNE $F875   unbedingter Sprung

**************************    Bandpuffer auf Band schreiben

Einsprung von $F1EA, $F2D7

F864   20 D7 F7   JSR $F7D7   Bandpufferadresse holen

Einsprung von $F67C

F867   A9 14      LDA #$14    Länge des Vorspanns vor WRITE
F869   85 AB      STA $AB     speichern

**************************    Block bzw. Programm auf Band
                              schreiben

Einsprung von $F7BE

F86B   20 38 F8   JSR $F838   wartet auf Record & Play
                              Taste
F86E   B0 6C      BCS $F8DC   verzweige falls STOP-Taste
                              gedrückt
F870   78         SEI         Interrupt verhindern
F871   A9 82      LDA #$82    Bitwert für IRQ bei Unterlauf
                              von Timer B
F873   A2 08      LDX #$08    Nummer des IRQ-Vektors, $FC6A
F875   A0 7F      LDY #$7F    Bitwert für alle IRQs sperren
F877   8C 0D DC   STY $DC0D   Wert schreiben
F87A   8D 0D DC   STA $DC0D   und neu setzen
F87D   AD 0E DC   LDA $DC0E   Control Register A laden
F880   09 19      ORA #$19    Bitwert für one shot, starten
F882   8D 0F DC   STA $DC0F   und ins Steuerregister für
                              Timer B
F885   29 91      AND #$91    Vergleichszeiger für Bandope-
F887   8D A2 02   STA $02A2   rationen entsprechend setzen
```

```
F88A   20 A4 F0   JSR $F0A4   auf Ende RS-232 Übertragung
                              warten
F88D   AD 11 D0   LDA $D011   Bildschirm
F890   29 EF      AND #$EF    dunkel
F892   8D 11 D0   STA $D011   Tasten
F895   AD 14 03   LDA $0314   IRQ-Vector
F898   8D 9F 02   STA $029F   nach $029F
F89B   AD 15 03   LDA $0315   und $02A0
F89E   8D A0 02   STA $02A0   speichern
F8A1   20 BD FC   JSR $FCBD   IRQ-Vektor für Band I/O
                              setzen (X-indiziert)
F8A4   A9 02      LDA #$02    Anzahl der
F8A6   85 BE      STA $BE     zu lesenden Blöcke
F8A8   20 97 FB   JSR $FB97   serielle Ausgabe vorbereiten
                              Bit-Zähler setzen
F8AB   A5 01      LDA $01     Prozessorport laden
F8AD   29 1F      AND #$1F    Bandmotor einschalten
F8AF   85 01      STA $01     und wieder speichern
F8B1   85 C0      STA $C0     Flag für Bandmotor setzen
F8B3   A2 FF      LDX #$FF    HIGH-Byte für Zähler
F8B5   A0 FF      LDY #$FF    LOW-Byte für Zähler
F8B7   88         DEY         Verzögerungsschleife
F8B8   D0 FD      BNE $F8B7   für Bandhochlaufzeit
F8BA   CA         DEX         HIGH-Byte veringern
F8BB   D0 F8      BNE $F8B5   verzweige falls nicht Null
F8BD   58         CLI         Interrupt für Band I/O
                              freigeben

**************************    I/O Abschluß abwarten

Einsprung von $F8CD

F8BE   AD A0 02   LDA $02A0   Band IRQ Vector mit normalem
F8C1   CD 15 03   CMP $0315   IRQ Vector vergleichen
F8C4   18         CLC         Carry =0 (ok Kennzeichen)
F8C5   F0 15      BEQ $F8DC   verzweige falls ja (fertig)
F8C7   20 D0 F8   JSR $F8D0   Testen auf Stop-Taste
F8CA   20 BC F6   JSR $F6BC   bei gedrückter Stop-Taste
                              Flag setzen
F8CD   4C BE F8   JMP $F8BE   weiter warten
```

```
**************************   testet auf Stop-Taste

Einsprung von $F821, $F8C7

F8D0   20 E1 FF   JSR $FFE1   Stop-Taste abfragen
F8D3   18         CLC         Carry =0 (ok Kennzeichen)
F8D4   D0 0B      BNE $F8E1   verzweige wenn Taste nein
                              gedrückt
F8D6   20 93 FC   JSR $FC93   Band-Motor aus, normalen
                              IRQ wiederherstellen
F8D9   38         SEC         Kennzeichen für Abbruch
F8DA   68         PLA         Rücksprung
F8DB   68         PLA         Adresse löschen
F8DC   A9 00      LDA #$00    Kennzeichen für normalen
F8DE   8D A0 02   STA $02A0   IRQ setzen
F8E1   60         RTS         Rücksprung

**************************   Band für Lesen vorbereiten

Einsprung von $F9CB, $FA0A, $FA2A, $FA67

F8E2   86 B1      STX $B1     X-Register speichern
F8E4   A5 B0      LDA $B0     Timing-Konstante laden
F8E6   0A         ASL  A      mit vier
F8E7   0A         ASL  A      multiplizieren
F8E8   18         CLC         zur Addition Carry löschen
F8E9   65 B0      ADC $B0     mit altem Wert addieren (*5)
F8EB   18         CLC         zur Addition Carry löschen
F8EC   65 B1      ADC $B1     alten X Wert dazuaddieren
F8EE   85 B1      STA $B1     und im Hilfszeiger speichern
F8F0   A9 00      LDA #$00    Akku löschen
F8F2   24 B0      BIT $B0     prüfe Timing-Konstante
F8F4   30 01      BMI $F8F7   verzweige, falls größer 128
F8F6   2A         ROL  A      Carry in die unterste
                              Position des Akkus schieben
F8F7   06 B1      ASL $B1     und Timer A
F8F9   2A         ROL  A      Initialisierung
F8FA   06 B1      ASL $B1     mit vier
F8FC   2A         ROL  A      multiplizieren
```

```
F8FD  AA          TAX         Akku ins X-Register
F8FE  AD 06 DC    LDA $DC06   LOW-Byte Timer B laden
F901  C9 16       CMP #$16    mit $16 vergleichen
F903  90 F9       BCC $F8FE   verzweige, wenn kleiner
F905  65 B1       ADC $B1     LOW-Byte für Initialisierung
                              addieren
F907  8D 04 DC    STA $DC04   Timer A LOW speichern
F90A  8A          TXA         HIGH-Byte für Initialisierung
F90B  6D 07 DC    ADC $DC07   zu  Timer B HIGH addieren
F90E  8D 05 DC    STA $DC05   und in Timer A HIGH schreiben
F911  AD A2 02    LDA $02A2   Init. Wert für Band Zeitkon.
F914  8D 0E DC    STA $DC0E   zum Starten von Timer A
F917  8D A4 02    STA $02A4   Timer A Flag zurücksetzten
F91A  AD 0D DC    LDA $DC0D   ICR laden
F91D  29 10       AND #$10    Bit isolieren
F91F  F0 09       BEQ $F92A   verzweige wenn IRQ nicht vom
                              Pin Flag
F921  A9 F9       LDA #$F9    Rücksprungadresse
F923  48          PHA         auf
F924  A9 2A       LDA #$2A    Stack
F926  48          PHA         schieben
F927  4C 43 FF    JMP $FF43   zum Interrupt
F92A  58          CLI         alle Interrupts freigeben
F92B  60          RTS         Rücksprung

**************************    Interrupt-Routine für Band
                              lesen
F92C  AE 07 DC    LDX $DC07   Timer B HIGH laden
F92F  A0 FF       LDY #$FF    Y-Register mit $FF laden (für
                              Timer)
F931  98          TYA         in Akku schieben
F932  ED 06 DC    SBC $DC06   Timer B von $FF abziehen
F935  EC 07 DC    CPX $DC07   Timer B mit altem Wert
                              vergleichen
F938  D0 F2       BNE $F92C   verzweige, falls vermindert
F93A  86 B1       STX $B1     Timer B HIGH ablegen
F93C  AA          TAX         und in Akku schieben
F93D  8C 06 DC    STY $DC06   Timer B LOW und
F940  8C 07 DC    STY $DC07   Timer B HIGH auf $FF setzen
F943  A9 19       LDA #$19    Arbeitsmodus für Timer B
```

```
F945  8D 0F DC   STA $DC0F   festlegen und starten
F948  AD 0D DC   LDA $DC0D   Interrupt Control Register
F94B  8D A3 02   STA $02A3   laden und nach $02A3
F94E  98         TYA         Y-REG in Akku ($FF)
F94F  E5 B1      SBC $B1     Errechnung von vergangener
                             Zeit seit letzter Flanke
F951  86 B1      STX $B1     vergangene Zeit LOW nach $B1
F953  4A         LSR  A      vergangene Zeit
F954  66 B1      ROR $B1     HIGH
F956  4A         LSR  A      geteilt
F957  66 B1      ROR $B1     durch vier
F959  A5 B0      LDA $B0     Timingkonstante laden
F95B  18         CLC         und mit
F95C  69 3C      ADC #$3C    $3C addiert
F95E  C5 B1      CMP $B1     errechnete Zeit größer als
                             die Zeit bei letzten Flanken
F960  B0 4A      BCS $F9AC   verzweige, wenn größer
F962  A6 9C      LDX $9C     Flag für empfangenes Byte
                             laden
F964  F0 03      BEQ $F969   verzweige, falls Null (Byte
                             nicht geladen)
F966  4C 60 FA   JMP $FA60   ansonsten nach $FA60

F969  A6 A3      LDX $A3     Byte vollständig gelesen
F96B  30 1B      BMI $F988   verzweige, falls ja
F96D  A2 00      LDX #$00    Code für kurzer Impuls (X=0)
F96F  69 30      ADC #$30    zu errechneter Zeit mit $30
F971  65 B0      ADC $B0     und mit Zeitkonstante
                             addieren
F973  C5 B1      CMP $B1     größer als Zeit beim letztem
                             Flanken ?
F975  B0 1C      BCS $F993   verzweige wenn größer
F977  E8         INX         sonst langer Impuls (X=1)
F978  69 26      ADC #$26    und wieder $26 und
F97A  65 B0      ADC $B0     Zeitkonstanten addieren
F97C  C5 B1      CMP $B1     jetzt größer ?
F97E  B0 17      BCS $F997   verzweige, falls ja
F980  69 2C      ADC #$2C    sonst wieder $2C und
F982  65 B0      ADC $B0     Zeitkonstante addieren
F984  C5 B1      CMP $B1     vergangene Zeit noch länger ?
```

```
F986  90 03     BCC $F98B   verzweige, wenn jetzt kürzer
F988  4C 10 FA  JMP $FA10   zu empfangenes Byte verarbeiten

F98B  A5 B4     LDA $B4     Flag für Timer A laden
F98D  F0 1D     BEQ $F9AC   verzweige, wenn Timer A nicht
                            freigegeben
F98F  85 A8     STA $A8     Zeiger auf 'READ ERROR'
                            setzen
F991  D0 19     BNE $F9AC   unbedingter Sprung

F993  E6 A9     INC $A9     Zeiger auf Impulswechsel +1
F995  B0 02     BCS $F999   unbedingter Sprung

Einsprung von $FA1C

F997  C6 A9     DEC $A9     Zeiger auf Impulswechsel -1
F999  38        SEC         Carry für Subtraktion setzen
F99A  E9 13     SBC #$13    Anfangswert ($13) und
F99C  E5 B1     SBC $B1     vergangene Zeit subtrahieren
F99E  65 92     ADC $92     und mit Flag für Timing
                            Korrektur addieren
F9A0  85 92     STA $92     Ergebnis dort speichern
F9A2  A5 A4     LDA $A4     Flag für Empfang beider
F9A4  49 01     EOR #$01    Impulse invertieren
F9A6  85 A4     STA $A4     und abspeichern
F9A8  F0 2B     BEQ $F9D5   verzweige wenn beide Impulse
                            empfangen
F9AA  86 D7     STX $D7     empfangenes Signal speichern
F9AC  A5 B4     LDA $B4     Flag für Timer A laden
F9AE  F0 22     BEQ $F9D2   verzweige wenn Timer gesperrt
F9B0  AD A3 02  LDA $02A3   ICR in Akku
F9B3  29 01     AND #$01    Bit 0 isolieren
F9B5  D0 05     BNE $F9BC   verzweige wenn Interrupt von
                            Timer A
F9B7  AD A4 02  LDA $02A4   Timer A abgelaufen
F9BA  D0 16     BNE $F9D2   nein, dann zum Interruptende
F9BC  A9 00     LDA #$00    Impulszähler
F9BE  85 A4     STA $A4     löschen und
F9C0  8D A4 02  STA $02A4   Zeiger auf Timeout setzen
```

```
F9C3  A5 A3      LDA $A3     prüfe ob Byte vollständig
                             gelesen
F9C5  10 30      BPL $F9F7   verzweige falls nein
F9C7  30 BF      BMI $F988   unbedingter Sprung

F9C9  A2 A6      LDX #$A6    Initialisierungswert für
                             Timer A
F9CB  20 E2 F8   JSR $F8E2   Band zum Lesen vorbereiten
F9CE  A5 9B      LDA $9B     Paritätsbyte in Akku
F9D0  D0 B9      BNE $F98B   verzweige falls parit. Fehler
F9D2  4C BC FE   JMP $FEBC   Rückkehr vom Interrupt

F9D5  A5 92      LDA $92     Timing Korrekturzeiger laden
F9D7  F0 07      BEQ $F9E0   verzweige wenn Flag gelöscht
F9D9  30 03      BMI $F9DE   verzweige wenn kleiner Null
F9DB  C6 B0      DEC $B0     Timing Konstante -1
F9DD  2C         .BYTE $2C   Skip zu $F9E0
F9DE  E6 B0      INC $B0     Timing Konstante +1
F9E0  A9 00      LDA #$00    Timing
F9E2  85 92      STA $92     Korrekturzeiger löschen
F9E4  E4 D7      CPX $D7     Vergleiche empfangenen Impuls
                             mit vorherigem
F9E6  D0 0F      BNE $F9F7   verzweige falls ungleich
F9E8  8A         TXA         Prüfe ob kurzer Impuls
                             empfangen
F9E9  D0 A0      BNE $F98B   falls nein, verzweige
F9EB  A5 A9      LDA $A9     Impulswechselzeiger laden
F9ED  30 BD      BMI $F9AC   verzweige wenn negativ
F9EF  C9 10      CMP #$10    vergleiche mit $10
F9F1  90 B9      BCC $F9AC   verzweige wenn kleiner $10
F9F3  85 96      STA $96     sonst EOB Flag empfangen
F9F5  B0 B5      BCS $F9AC   unbedingter Sprung

F9F7  8A         TXA         Empfangenes Bit in Akku
F9F8  45 9B      EOR $9B     mit Band-Parität verknüpfen
F9FA  85 9B      STA $9B     in Band-Parität speichern
F9FC  A5 B4      LDA $B4     Flag für Timer A laden
F9FE  F0 D2      BEQ $F9D2   verzweige wenn nicht freige-
                             geben
FA00  C6 A3      DEC $A3     Speicher für Bitzähler -1
```

```
FA02  30 C5      BMI $F9C9   verzweige wenn Paritätsbit
                             empfangen
FA04  46 D7      LSR $D7     gelesenes Bit ins Carry und
FA06  66 BF      ROR $BF     dann in $BF rollen
FA08  A2 DA      LDX #$DA    Initialisierungswert für
                             Timer A ins X-Register
FA0A  20 E2 F8   JSR $F8E2   zur Kassettensynchronisation
FA0D  4C BC FE   JMP $FEBC   Rückkehr vom Interrupt

Einsprung von $F988

FA10  A5 96      LDA $96     Prüfe ob EOB empfangen
FA12  F0 04      BEQ $FA18   falls nein, verzweige
FA14  A5 B4      LDA $B4     Prüfe ob Timer A freige.
FA16  F0 07      BEQ $FA1F   wenn nein, überspringe Bit
                             Zähler Test
FA18  A5 A3      LDA $A3     Bitzähler laden
FA1A  30 03      BMI $FA1F   verzweige falls negatv
FA1C  4C 97 F9   JMP $F997   langen Impuls verarbeiten

FA1F  46 B1      LSR $B1     vergangene Zeit seit letztem
                             Flangen halbieren
FA21  A9 93      LDA #$93    und diesen Wert
FA23  38         SEC         von $93
FA24  E5 B1      SBC $B1     abziehen
FA26  65 B0      ADC $B0     dazu dann Timing-Konstante
                             addieren
FA28  0A         ASL  A      und Ergebnis verdoppeln
FA29  AA         TAX         Ergebnis ins X-Register
FA2A  20 E2 F8   JSR $F8E2   Timing initialisieren
FA2D  E6 9C      INC $9C     Flag für Byte empfangen
                             setzen
FA2F  A5 B4      LDA $B4     Flag für Timer A laden
FA31  D0 11      BNE $FA44   verzweige falls freigegeben
FA33  A5 96      LDA $96     wurde EOB emfangen ?
FA35  F0 26      BEQ $FA5D   verzweige wenn nicht
                             empfangen
FA37  85 A8      STA $A8     Flag für Lesefehler setzen
FA39  A9 00      LDA #$00    Flag für
FA3B  85 96      STA $96     EOB rücksetzen
```

```
FA3D  A9 81     LDA #$81    Interrupt für
FA3F  8D 0D DC  STA $DC0D   Timer A freigeben
FA42  85 B4     STA $B4     und Flag für Timer A setzen
FA44  A5 96     LDA $96     Flag für EOB laden
FA46  85 B5     STA $B5     und nach $B5 kopieren
FA48  F0 09     BEQ $FA53   verzweige wenn kein EOB
FA4A  A9 00     LDA #$00    Flag für Timer A
FA4C  85 B4     STA $B4     löschen und auch
FA4E  A9 01     LDA #$01    Interruptflag
FA50  8D 0D DC  STA $DC0D   wieder löschen
FA53  A5 BF     LDA $BF     Shift Register für Read laden
FA55  85 BD     STA $BD     und nach $BD bringen
FA57  A5 A8     LDA $A8     Flag für Lesefehler laden
FA59  05 A9     ORA $A9     mit Impulswechselzeiger
FA5B  85 B6     STA $B6     verknüpfen und in Fehlercode
                            des Bytes ablegen
FA5D  4C BC FE  JMP $FEBC   Rückkehr vom Interrupt

Einsprung von $F966

FA60  20 97 FB  JSR $FB97   Bitzähler für serielle
                            Ausgabe setzen
FA63  85 9C     STA $9C     Zeiger auf Byte empfangen
                            rücksetzen
FA65  A2 DA     LDX #$DA    Initialisierungswert Timer A
FA67  20 E2 F8  JSR $F8E2   Kassettensynchronisation
FA6A  A5 BE     LDA $BE     Anzahl der verbliebenen
                            Blöcke laden
FA6C  F0 02     BEQ $FA70   verzweige wenn Null
FA6E  85 A7     STA $A7     Blockanzahl neu setzen
FA70  A9 0F     LDA #$0F    Maskenwert für Zählung vor
                            dem Lesen
FA72  24 AA     BIT $AA     Prüfe Zeiger für Lesen von
                            Band
FA74  10 17     BPL $FA8D   verzweige wenn alle Zeichen
                            empfangen (Ende)
FA76  A5 B5     LDA $B5     Flag für EOB laden
FA78  D0 0C     BNE $FA86   verzweige wenn gültiges EOB
                            empfangen
```

```
FA7A  A6 BE     LDX $BE     Anzahl der verbliebenen
                            Blöcke laden
FA7C  CA        DEX         Anzahl -1
FA7D  D0 0B     BNE $FA8A   verzweige wenn nicht Null
FA7F  A9 08     LDA #$08    'LONG BLOCK' error
FA81  20 1C FE  JSR $FE1C   Status setzen
FA84  D0 04     BNE $FA8A   unbedingter Sprung zum
                            normalen IRQ
FA86  A9 00     LDA #$00    Flag für Lesen vom Band auf
FA88  85 AA     STA $AA     Abtastung setzen
FA8A  4C BC FE  JMP $FEBC   Rückkehr vom Interrupt

FA8D  70 31     BVS $FAC0   verzweige wenn Bandzeiger auf
                            lesen
FA8F  D0 18     BNE $FAA9   verzweige wenn Bandzeiger
                            auf Zählen
FA91  A5 B5     LDA $B5     Flag für EOB laden
FA93  D0 F5     BNE $FA8A   verzweige wenn EOB empfangen
FA95  A5 B6     LDA $B6     Flag für Lesefehler laden
FA97  D0 F1     BNE $FA8A   verzweige falls Fehler
                            aufgetreten
FA99  A5 A7     LDA $A7     Anzahl der noch zu lesenden
                            Blöcke holen
FA9B  4A        LSR  A      Bit 0 ins Carry schieben
FA9C  A5 BD     LDA $BD     hole gelesenes Byte
FA9E  30 03     BMI $FAA3   verzweige wenn es Zählbyte
                            ist
FAA0  90 18     BCC $FABA   verzweige wenn mehr als ein
                            Block zu lesen
FAA2  18        CLC         lösche Carry um nicht zu
                            verzweigen
FAA3  B0 15     BCS $FABA   verzweige falls nur ein Block
                            zu lesen
FAA5  29 0F     AND #$0F    Bits 0 bis 3 isolieren
FAA7  85 AA     STA $AA     und für Zählung speichern
FAA9  C6 AA     DEC $AA     alle Synchrrnisationsbytes
                            empfangen
FAAB  D0 DD     BNE $FA8A   wenn nein verzweige
FAAD  A9 40     LDA #$40    Bandzeiger auf
FAAF  85 AA     STA $AA     lesen stellen
```

```
FAB1  20 8E FB   JSR $FB8E   Ein/Ausgabe Adresse kopieren
FAB4  A9 00      LDA #$00    Flag für
FAB6  85 AB      STA $AB     Leseprüfsumme löschen
FAB8  F0 D0      BEQ $FA8A   unbedingter Sprung

FABA  A9 80      LDA #$80    Bandzeiger
FABC  85 AA      STA $AA     auf Ende stellen
FABE  D0 CA      BNE $FA8A   unbedingter Sprung

FAC0  A5 B5      LDA $B5     Flag für EOB laden
FAC2  F0 0A      BEQ $FACE   verzweige wenn nicht gesetzt
FAC4  A9 04      LDA #$04    'SHORT BLOCK' error
FAC6  20 1C FE   JSR $FE1C   Status setzen
FAC9  A9 00      LDA #$00    Code für Lesezeiger auf
                             "Abtasten"
FACB  4C 4A FB   JMP $FB4A   setzen, unbedingter Sprung

FACE  20 D1 FC   JSR $FCD1   Endadresse schon erreicht ?
FAD1  90 03      BCC $FAD6   nein dann verzweige
FAD3  4C 48 FB   JMP $FB48   zu Read Ende für Block
FAD6  A6 A7      LDX $A7     nur noch
FAD8  CA         DEX         ein Block zu lesen
FAD9  F0 2D      BEQ $FB08   verzweige wenn ja (Pass 2)
FADB  A5 93      LDA $93     Load/Verify-Flag
FADD  F0 0C      BEQ $FAEB   verzweige wenn Load
FADF  A0 00      LDY #$00    Zähler auf Null setzen
FAE1  A5 BD      LDA $BD     gelesenes Byte
FAE3  D1 AC      CMP ($AC),Y vergleichen
FAE5  F0 04      BEQ $FAEB   verzweige wenn Übereinstim-
                             mung
FAE7  A9 01      LDA #$01    Fehlerflag
FAE9  85 B6      STA $B6     setzen
FAEB  A5 B6      LDA $B6     Fehlerflag laden
FAED  F0 4B      BEQ $FB3A   verzweige wenn kein Fehler
                             aufgetreten
FAEF  A2 3D      LDX #$3D    bereits 31 Fehler
FAF1  E4 9E      CPX $9E     aufgetreten
FAF3  90 3E      BCC $FB33   verzweige wenn weniger Fehler
FAF5  A6 9E      LDX $9E     Index für Lesefehler
FAF7  A5 AD      LDA $AD     laufender Adressbyte HIGH
```

```
FAF9  9D 01 01   STA $0101,X im Stack speichern
FAFC  A5 AC      LDA $AC     Adressbyte LOW
FAFE  9D 00 01   STA $0100,X für spätere Korrektur
                             ebenfalls im Stack speichern
FB01  E8         INX         Zeiger auf nachfolgende
FB02  E8         INX         freie Stelle setzen
FB03  86 9E      STX $9E     und abspeichern
FB05  4C 3A FB   JMP $FB3A   weitermachen

FB08  A6 9F      LDX $9F     bereits alle Lesefehler
FB0A  E4 9E      CPX $9E     korrigiert ?
FB0C  F0 35      BEQ $FB43   verzweige falls ja
FB0E  A5 AC      LDA $AC     Adressbyte LOW laden
FB10  DD 00 01   CMP $0100,X mit fehlerhaftem Adressbyte
                             LOW vergleichen
FB13  D0 2E      BNE $FB43   verzweige falls nicht
                             gefunden
FB15  A5 AD      LDA $AD     Adressbyte HIGH laden
FB17  DD 01 01   CMP $0101,X mit fehlerhaftem Adressbyte
                             HIGH vergleichen
FB1A  D0 27      BNE $FB43   verzweige wenn nicht gefunden
FB1C  E6 9F      INC $9F     Korrekturzähler
FB1E  E6 9F      INC $9F     Pass 2 um zwei erhöhen
FB20  A5 93      LDA $93     Verify-Flag gesetzt
FB22  F0 0B      BEQ $FB2F   verzweige wenn nicht gesetzt
FB24  A5 BD      LDA $BD     gelesenes Byte laden
FB26  A0 00      LDY #$00    Zähler auf Null setzen
FB28  D1 AC      CMP ($AC),Y mit Speicherinhalt verglei-
                             chen
FB2A  F0 17      BEQ $FB43   verzweige wenn gleich, dann
                             nächstes Byte
FB2C  C8         INY         Flag für
FB2D  84 B6      STY $B6     Fehler setzen
FB2F  A5 B6      LDA $B6     Fehlerflag testen
FB31  F0 07      BEQ $FB3A   verzweige wenn kein Fehler
FB33  A9 10      LDA #$10    'SECOND PASS' error
FB35  20 1C FE   JSR $FE1C   Status setzen
FB38  D0 09      BNE $FB43   und nächstes Byte verarbeiten
```

```
Einsprung von $FB05

FB3A   A5 93      LDA $93      Verify-Flag laden
FB3C   D0 05      BNE $FB43    verzweige wenn gesetzt
FB3E   A8         TAY          Zeiger löschen
FB3F   A5 BD      LDA $BD      gelesenes Byte
FB41   91 AC      STA ($AC),Y  speichern
FB43   20 DB FC   JSR $FCDB    Adresszeiger erhöhen
FB46   D0 43      BNE $FB8B    Rückkehr vom Interrupt

Einsprung von $FAD3

FB48   A9 80      LDA #$80     Flag für Lesen

Einsprung von $FACB

FB4A   85 AA      STA $AA      auf Ende
FB4C   78         SEI          Interrupt verhindern
FB4D   A2 01      LDX #$01     IRQ vom
FB4F   8E 0D DC   STX $DC0D    Timer A verhindern
FB52   AE 0D DC   LDX $DC0D    IRQ-Flag löschen
FB55   A6 BE      LDX $BE      Pass-Zähler
FB57   CA         DEX          erniedrigen
FB58   30 02      BMI $FB5C    verzweige wenn Null gewesen
FB5A   86 BE      STX $BE      Passzähler merken
FB5C   C6 A7      DEC $A7      Blockzähler vermindern
FB5E   F0 08      BEQ $FB68    verzweige wenn Null
FB60   A5 9E      LDA $9E      Fehler in Pass 1 aufgetre-
                               ten ?
FB62   D0 27      BNE $FB8B    ja, Rückkehr vom Interrupt
FB64   85 BE      STA $BE      kein Block mehr zu verarbei-
                               ten
FB66   F0 23      BEQ $FB8B    Rückkehr vom Interrupt

FB68   20 93 FC   JSR $FC93    ein Pass beendet
FB6B   20 8E FB   JSR $FB8E    Adresse wieder auf Programm-
                               anfang
FB6E   A0 00      LDY #$00     Zähler auf Null setzen
FB70   84 AB      STY $AB      Checksumme löschen
FB72   B1 AC      LDA ($AC),Y  Programm
```

```
FB74   45 AB      EOR $AB     Checksumme berechnen
FB76   85 AB      STA $AB     und speichern
FB78   20 DB FC   JSR $FCDB   Adresszeiger erhöhen
FB7B   20 D1 FC   JSR $FCD1   Endadresse schon erreicht ?
FB7E   90 F2      BCC $FB72   nein, weiter vergleichen
FB80   A5 AB      LDA $AB     berechnete Checksumme
FB82   45 BD      EOR $BD     mit Checksumme vom Band
                              vergleichen
FB84   F0 05      BEQ $FB8B   Checksumme gleich , dann ok
FB86   A9 20      LDA #$20    'CHECKSUM' error
FB88   20 1C FE   JSR $FE1C   Status setzen
FB8B   4C BC FE   JMP $FEBC   Rückkehr vom Interrupt

**************************    laufenden Zeiger auf
                              Programmstart

Einsprung von $F617, $FAB1, $FB6B, $FC88

FB8E   A5 C2      LDA $C2     Startadresse
FB90   85 AD      STA $AD     $C1/$C2
FB92   A5 C1      LDA $C1     nach $AC/$AD
FB94   85 AC      STA $AC     speichern
FB96   60         RTS         Rücksprung

**************************    Bitzähler für serielle
                              Ausgabe setzen

Einsprung von $F8A8, $FA60, $FC16, $FC75

FB97   A9 08      LDA #$08    Zähler für 8 Bits
FB99   85 A3      STA $A3     Nach $A3
FB9B   A9 00      LDA #$00    Akku mit $00 laden
FB9D   85 A4      STA $A4     Bit-Impuls-Flag löschen
FB9F   85 A8      STA $A8     Lesefehler Byte löschen
FBA1   85 9B      STA $9B     Parity-Bit löschen
FBA3   85 A9      STA $A9     Impulswechsel-Flag löschen
FBA5   60         RTS         Rücksprung
```

```
**************************** Ein Bit auf Band schreiben

Einsprung von $FBF0

FBA6   A5 BD      LDA $BD     Bit in $BD
FBA8   4A         LSR  A      Bit 0 in Carry
FBA9   A9 60      LDA #$60    Zeit für '0' Bit
FBAB   90 02      BCC $FBAF   verzweige falls Carry=0

Einsprung von $FBE7

FBAD   A9 B0      LDA #$B0    Zeit für '1' Bit

Einsprung von $FC6C

FBAF   A2 00      LDX #$00    HIGH-Byte Timerwert laden

Einsprung von $FBD5

FBB1   8D 06 DC   STA $DC06   Timer B LOW
FBB4   8E 07 DC   STX $DC07   Timer B HIGH
FBB7   AD 0D DC   LDA $DC0D   Interrupt-Flag löschen
FBBA   A9 19      LDA #$19    Timer
FBBC   8D 0F DC   STA $DC0F   B starten
FBBF   A5 01      LDA $01     Tape-Write-Bit laden
FBC1   49 08      EOR #$08    Ausgabe-Bit für Band
                              invertieren
FBC3   85 01      STA $01     und speichern
FBC5   29 08      AND #$08    augenblicklichen Pegel merken
FBC7   60         RTS

FBC8   38         SEC         Block-Write-Flag
FBC9   66 B6      ROR $B6     Negativ
FBCB   30 3C      BMI $FC09   Rückkehr vom Interrupt

**************************** Interrupt-Routine für Band
                              schreiben
FBCD   A5 A8      LDA $A8     falls 'Byte'-Impuls ge-
FBCF   D0 12      BNE $FBE3   schrieben, dann verzweige
FBD1   A9 10      LDA #$10    Timer auf
```

```
FBD3  A2 01     LDX #$01    $110 (272)
FBD5  20 B1 FB  JSR $FBB1   Takt auf Band schreiben
FBD8  D0 2F     BNE $FC09   Rückkehr vom Interrupt
FBDA  E6 A8     INC $A8     '1' Byte-Write-Flag setzen
FBDC  A5 B6     LDA $B6     falls Block-Write-Flag
                            positiv, dann
FBDE  10 29     BPL $FC09   Rückkehr vom Interrupt
FBE0  4C 57 FC  JMP $FC57   zweiten Block schreiben
FBE3  A5 A9     LDA $A9     falls '1' Bit gesezt
FBE5  D0 09     BNE $FBF0   dann verzweige
FBE7  20 AD FB  JSR $FBAD   '1' Bit schreiben
FBEA  D0 1D     BNE $FC09   Rückkehr vom Interrupt
FBEC  E6 A9     INC $A9     '1' Bit-Flag setzen
FBEE  D0 19     BNE $FC09   Rückkehr vom Interrupt
FBF0  20 A6 FB  JSR $FBA6   Bit auf Band schreiben
FBF3  D0 14     BNE $FC09   Rückkehr vom Interrupt
FBF5  A5 A4     LDA $A4     Bit-Impulsflag laden
FBF7  49 01     EOR #$01    Bit 0 invertieren
FBF9  85 A4     STA $A4     und speichern
FBFB  F0 0F     BEQ $FC0C   falls null, dann verzweige
FBFD  A5 BD     LDA $BD     Bit-SHIFT-Register laden
FBFF  49 01     EOR #$01    Bit für Ausgabe invertieren
FC01  85 BD     STA $BD     und speichern
FC03  29 01     AND #$01    Bit holen und mit
FC05  45 9B     EOR $9B     Parity-Bit verknüpfen
FC07  85 9B     STA $9B     und speichern
FC09  4C BC FE  JMP $FEBC   Rückkehr vom Interrupt
FC0C  46 BD     LSR $BD     nächstes Bit in Position 0
FC0E  C6 A3     DEC $A3     Bitzähler erniedrigen
FC10  A5 A3     LDA $A3     und laden
FC12  F0 3A     BEQ $FC4E   nächstes Bit ausgeben
FC14  10 F3     BPL $FC09   Rückkehr vom Interrupt
FC16  20 97 FB  JSR $FB97   Bitzähler wieder auf 8 setzen
FC19  58        CLI         Interrupt freigeben
FC1A  A5 A5     LDA $A5     Falls Synchronbytes geschrie-
                            ben
FC1C  F0 12     BEQ $FC30   dann verzweige
FC1E  A2 00     LDX #$00    Prüfsumme
FC20  86 D7     STX $D7     löschen
FC22  C6 A5     DEC $A5     Zähler vermindern
```

```
FC24   A6 BE      LDX $BE      noch zu schreibende
                               Blockanzahl laden
FC26   E0 02      CPX #$02     falls erster Block nicht
FC28   D0 02      BNE $FC2C    geschrieben, dann verzweige
FC2A   09 80      ORA #$80     Bit 7 setzen
FC2C   85 BD      STA $BD      und speichern
FC2E   D0 D9      BNE $FC09    Rückkehr vom Interrupt
FC30   20 D1 FC   JSR $FCD1    Endadresse schon erreicht ?
FC33   90 0A      BCC $FC3F    falls kleiner, dann
                               weiterschreiben
FC35   D0 91      BNE $FBC8    falls ungleich, dann
                               Block-Write-Flag setzen
FC37   E6 AD      INC $AD      HIGH-Byte ungleich machen
FC39   A5 D7      LDA $D7      Prüfsumme laden
FC3B   85 BD      STA $BD      und in SHIFT-Flag speichern
FC3D   B0 CA      BCS $FC09    Rückkehr vom Interrupt
FC3F   A0 00      LDY #$00     Zähler auf Null
FC41   B1 AC      LDA ($AC),Y  zu schreibendes Byte laden
FC43   85 BD      STA $BD      in SHIFT-Flag bringen
FC45   45 D7      EOR $D7      Prüfsumme
FC47   85 D7      STA $D7      bilden
FC49   20 DB FC   JSR $FCDB    Adresszeiger erhöhen
FC4C   D0 BB      BNE $FC09    Rückkehr vom Interrupt
FC4E   A5 9B      LDA $9B      Parity-Bit
FC50   49 01      EOR #$01     invertieren

FC52   85 BD      STA $BD      und ins SHIFT-Flag speichern
FC54   4C BC FE   JMP $FEBC    Rückkehr vom Interrupt

Einsprung von $FBE0

FC57   C6 BE      DEC $BE      Zähler für Blocks erniedrigen
FC59   D0 03      BNE $FC5E    falls noch ein Block,
FC5B   20 CA FC   JSR $FCCA    dann Bandmotor aus
FC5E   A9 50      LDA #$50     80
FC60   85 A7      STA $A7      Zähler für Impulse
FC62   A2 08      LDX #$08     Offset für IRQ
FC64   78         SEI          Interrupt verhindern
FC65   20 BD FC   JSR $FCBD    IRQ auf $FC6A
FC68   D0 EA      BNE $FC54    Rückkehr vom Interrupt
```

```
**************************      Interrupt-Routine für Band
                                schreiben
FC6A   A9 78      LDA #$78      120
FC6C   20 AF FB   JSR $FBAF     Bit auf Band schreiben
FC6F   D0 E3      BNE $FC54     Rückkehr vom Interrupt
FC71   C6 A7      DEC $A7       Zähler erniedrigen
FC73   D0 DF      BNE $FC54     nicht null, dann Rückkehr
                                vom Interrupt
FC75   20 97 FB   JSR $FB97     Bitzähler für serielle
                                Ausgabe setzen
FC78   C6 AB      DEC $AB       falls Datenende nicht er-
                                reicht, dann
FC7A   10 D8      BPL $FC54     Rückkehr vom Interrupt
FC7C   A2 0A      LDX #$0A      IRQ
FC7E   20 BD FC   JSR $FCBD     IRQ auf $FBCD
FC81   58         CLI           Interrupt ermöglichen
FC82   E6 AB      INC $AB       Shortdauer
FC84   A5 BE      LDA $BE       Zähler für Anzahl der Blocks
FC86   F0 30      BEQ $FCB8     alle Blocks geschrieben ?
FC88   20 8E FB   JSR $FB8E     Adresse wieder auf Anfang
                                setzen
FC8B   A2 09      LDX #$09      Zähler für
FC8D   86 A5      STX $A5       Synchronisation
FC8F   86 B6      STX $B6       Flag für Block geschrieben
FC91   D0 83      BNE $FC16     unbedingter Sprung

**************************      Rekorderbetrieb beenden

Einsprung von $F8D6, $FB68, $FCB8

FC93   08         PHP           Status merken
FC94   78         SEI           Interrupt verhindern
FC95   AD 11 D0   LDA $D011     Bildschirm
FC98   09 10      ORA #$10      wieder
FC9A   8D 11 D0   STA $D011     einschalten
FC9D   20 CA FC   JSR $FCCA     Rekordermotor ausschalten
FCA0   A9 7F      LDA #$7F      Interruptmöglichkeiten
FCA2   8D 0D DC   STA $DC0D     löschen
```

```
FCA5   20 DD FD   JSR $FDDD    CIA wieder auf Standardwerte,
                               1/60 s Timing
FCA8   AD A0 02   LDA $02A0    Interruptvektor schon auf
                               Standardwert ?
FCAB   F0 09      BEQ $FCB6    falls ja, dann fertig
FCAD   8D 15 03   STA $0315    ansonsten zurücksetzen
FCB0   AD 9F 02   LDA $029F    geretteten IRQ zurückholen
FCB3   8D 14 03   STA $0314    und speichern
FCB6   28         PLP          Status zurückholen
FCB7   60         RTS          Rücksprung

****************************   IRQ-Vektor setzen,
                               X-indiziert
FCB8   20 93 FC   JSR $FC93    IRQ auf Standard
FCBB   F0 97      BEQ $FC54    Abschluß IRQ

Einsprung von $F8A1, $FC65, $FC7E

FCBD   BD 93 FD   LDA $FD93,X IRQ-Vektor
FCC0   8D 14 03   STA $0314    aus Tabelle setzen
FCC3   BD 94 FD   LDA $FD94,X IRQ-Vektor
FCC6   8D 15 03   STA $0315    aus Tabelle setzen
FCC9   60         RTS          Rücksprung

****************************

Einsprung von $FC5B, $FC9D

FCCA   A5 01      LDA $01      Rekorder-
FCCC   09 20      ORA #$20     motor
FCCE   85 01      STA $01      ausschalten
FCD0   60         RTS          Rücksprung

****************************   prüft auf Erreichen der
                               Endadresse

Einsprung von $F624, $FACE, $FB7B, $FC30

FCD1   38         SEC          Carry für Subtraktion
                               vorbereiten
```

```
FCD2   A5 AC      LDA $AC     laufende Adresse
FCD4   E5 AE      SBC $AE     $AC/$AD
FCD6   A5 AD      LDA $AD     Endadresse
FCD8   E5 AF      SBC $AF     $AE/$AF
FCDA   60         RTS         Rücksprung

****************************

Einsprung von $F63A, $FB43, $FB78, $FC49

FCDB   E6 AC      INC $AC     Adreßzeiger
FCDD   D0 02      BNE $FCE1   er-
FCDF   E6 AD      INC $AD     höhen
FCE1   60         RTS         Rücksprung


****************************  RESET
FCE2   A2 FF      LDX #$FF    Wert für Stapelzeiger
FCE4   78         SEI         Interrupt setzen
FCE5   9A         TXS         Stapelzeiger initialisieren
FCE6   D8         CLD         Dezimalflag zurücksetzen
FCE7   20 02 FD   JSR $FD02   prüft auf ROM in $8000
FCEA   D0 03      BNE $FCEF   kein Autostart-Modul ?
FCEC   6C 00 80   JMP ($8000) Sprung auf Modul-Start

FCEF   8E 16 D0   STX $D016   Videocontroller Steuerreg. 2
FCF2   20 A3 FD   JSR $FDA3   Interrupt vorbereiten
FCF5   20 50 FD   JSR $FD50   Arbeitsspeicher initialisieren
FCF8   20 15 FD   JSR $FD15   Hardware und I/O Vekt. setzen
FCFB   20 5B FF   JSR $FF5B   Video-Reset
FCFE   58         CLI
FCFF   6C 00 A0   JMP ($A000) zum BASIC Kaltstart

****************************  prüft auf ROM in $8000

Einsprung von $FCE7, $FE56

FD02   A2 05      LDX #$05    Zeiger setzen
FD04   BD 0F FD   LDA $FD0F,X Wert aus Tabelle holen und
FD07   DD 03 80   CMP $8003,X ab $8000 vergleichen (CBM/80)
```

```
FD0A   D0 03      BNE $FD0F   verzweige wenn ungleich
FD0C   CA         DEX         Zeiger vermindern
FD0D   D0 F5      BNE $FD04   weiter wenn nicht 5 Bytes
FD0F   60         RTS         Rücksprung

**************************** ROM-Modul Identifizierung
FD10   C3 C2 CD 38 30        'CBM80'

**************************** Hardware und I/O Vektoren
                             setzen/holen

Einsprung von $FCF8, $FE66, $FF8A

FD15   A2 30      LDX #$30    LOW- und HIGH-Byte des
FD17   A0 FD      LDY #$FD    Zeigers auf Tabelle $FD30
FD19   18         CLC         Flag für 'Vektoren setzen'

Einsprung von $FF8D

FD1A   86 C3      STX $C3     LOW- und HIGH-Byte
FD1C   84 C4      STY $C4     des Zeigers setzen
FD1E   A0 1F      LDY #$1F    Zeiger setzen (16 Vektoren)
FD20   B9 14 03   LDA $0314,Y Wert aus Tabelle holen
FD23   B0 02      BCS $FD27   C=1 holen,C=0 setzen
FD25   B1 C3      LDA ($C3),Y Tabellenwert holen
FD27   91 C3      STA ($C3),Y Tabellenwert setzen
FD29   99 14 03   STA $0314,Y Wert in Tabelle ablegen
FD2C   88         DEY         Zähler vermindern
FD2D   10 F1      BPL $FD20   Fertig? nein: nächster Wert
FD2F   60         RTS         Rücksprung

**************************** Tabelle der Hardware
                             und I/O-Vektoren
FD30   31 EA 66 FE 47 FE 4A F3
FD38   91 F2 0E F2 50 F2 33 F3
FD40   57 F1 CA F1 ED F6 3E F1
FD48   2F F3 66 FE A5 F4 ED F5
```

```
**************************** Arbeitsspei. initialisieren

Einsprung von $FCF5, $FF87

FD50   A9 00      LDA #$00    Wert zum Löschen laden
FD52   A8         TAY         als Zähler nach Y
FD53   99 02 00   STA $0002,Y Zeropage,
FD56   99 00 02   STA $0200,Y Page 2 und
FD59   99 00 03   STA $0300,Y Page 3 löschen
FD5C   C8         INY         Zähler vermindern
FD5D   D0 F4      BNE $FD53   weiter wenn nicht fertig
FD5F   A2 3C      LDX #$3C    Werte für Startadresse
FD61   A0 03      LDY #$03    des Bandpuffers laden
FD63   86 B2      STX $B2     Bandpuffer Zeiger
FD65   84 B3      STY $B3     auf $033C setzen
FD67   A8         TAY         Zeiger in Y auf 0 setzen
FD68   A9 03      LDA #$03    Wert für RAM testen ($04-1)
FD6A   85 C2      STA $C2     Startadresse (HIGH) des RAM
FD6C   E6 C2      INC $C2     setzen und auf $0400 erhöhen
FD6E   B1 C1      LDA ($C1),Y Wert holen
FD70   AA         TAX         Wert merken
FD71   A9 55      LDA #$55    %01010101 ($55)
FD73   91 C1      STA ($C1),Y abspeichern und über-
FD75   D1 C1      CMP ($C1),Y prüfen, ob Wert drin ist
FD77   D0 0F      BNE $FD88   ungleich dann kein RAM
FD79   2A         ROL         %10101010
FD7A   91 C1      STA ($C1),Y Wert abspeichern und
FD7C   D1 C1      CMP ($C1),Y überprüfen, ob Wert drin ist
FD7E   D0 08      BNE $FD88   ungleich dann kein RAM
FD80   8A         TXA         Wert wieder zurückholen
FD81   91 C1      STA ($C1),Y und wieder zurückschreiben
FD83   C8         INY         Zeiger erhöhen
FD84   D0 E8      BNE $FD6E   Pageende? nein: weiter
FD86   F0 E4      BEQ $FD6C   sonst Zeiger-HIGH erhöhen
FD88   98         TYA         Zeiger-LOW ins
FD89   AA         TAX         X-Register bringen
FD8A   A4 C2      LDY $C2     Zeiger-HIGH holen
FD8C   18         CLC         C=0 (Flag für setzen)
FD8D   20 2D FE   JSR $FE2D   Memory (RAM) Top setzen
FD90   A9 08      LDA #$08    HIGH-Byte der Startadresse
```

```
FD92   8D 82 02   STA $0282   Memory (RAM) Start auf $800
FD95   A9 04      LDA #$04    HIGH-Byte der Startadresse
FD97   8D 88 02   STA $0288   Video-RAM auf $400
FD9A   60         RTS         Rücksprung

**************************** IRQ Vektoren
FD9B   6A FC CD FB 31 EA 2C F9  $FC6A, $FBCD, $EA31, $F92C

**************************** Interrupt Initialisierung

Einsprung von $FCF2, $FE69, $FF84

FDA3   A9 7F      LDA #$7F    Interrupt löschen
FDA5   8D 0D DC   STA $DC0D   ICR CIA 1
FDA8   8D 0D DD   STA $DD0D   ICR CIA 2
FDAB   8D 00 DC   STA $DC00   Port A  CIA 1
                              Tastatur Matrixzeile 0
FDAE   A9 08      LDA #$08    Wert laden
FDB0   8D 0E DC   STA $DC0E   CRA CIA 1 Timer A 'one shot'
FDB3   8D 0E DD   STA $DD0E   CRA CIA 2 Timer A 'one shot'
FDB6   8D 0F DC   STA $DC0F   CRB CIA 1 Timer B 'one shot'
FDB9   8D 0F DD   STA $DD0F   CRB CIA 2 Timer B 'one shot'
FDBC   A2 00      LDX #$00    Eingangs-Modus
FDBE   8E 03 DC   STX $DC03   Datenrichtungsreg. B CIA 1
FDC1   8E 03 DD   STX $DD03   Datenrichtungsreg. B CIA 2
FDC4   8E 18 D4   STX $D418   Lautstärke für SID auf Null
FDC7   CA         DEX         Ausgabe-Modus
FDC8   8E 02 DC   STX $DC02   Datenrichtungsreg. A  CIA 1
FDCB   A9 07      LDA #$07    Videocontroller auf
                              unterste 16 K
FDCD   8D 00 DD   STA $DD00   Port A  CIA 2, ATN löschen
FDD0   A9 3F      LDA #$3F    Bit 0 bis 5 auf Ausgabe
FDD2   8D 02 DD   STA $DD02   Datenrichtungsreg. A  CIA 2
FDD5   A9 E7      LDA #$E7    Normalwert laden und
FDD7   85 01      STA $01     Speicheraufteilung neu setzen
FDD9   A9 2F      LDA #$2F    Bit 0-3 und 5 Ausgang,
                              Bit 4 Eingang
FDDB   85 00      STA $00     Datenrichtung Prozessorport
```

```
Einsprung von $FCA5, $FF6B

FDDD   AD A6 02   LDA $02A6   NTSC-Version ?
FDE0   F0 0A      BEQ $FDEC   ja
FDE2   A9 25      LDA #$25    Wert für PAL-Version
FDE4   8D 04 DC   STA $DC04   Timer für PAL-Version setzen
FDE7   A9 40      LDA #$40    $4025 = 16421 Zyklen
FDE9   4C F3 FD   JMP $FDF3   NTSC-Version übergehen
FDEC   A9 95      LDA #$95    Wert für NTSC-Version
FDEE   8D 04 DC   STA $DC04   Timer für NTSC-Version setzen
FDF1   A9 42      LDA #$42    $4295 = 17045 Zyklen

Einsprung von $FDE9

FDF3   8D 05 DC   STA $DC05   Timer-HIGH setzen
FDF6   4C 6E FF   JMP $FF6E   Interrupt durch Timer setzen

****************************  Parameter f. Filenamen setzen

Einsprung von $FFBD

FDF9   85 B7      STA $B7     Länge speichern
FDFB   86 BB      STX $BB     Adresse-LOW speichern
FDFD   84 BC      STY $BC     Adresse-HIGH speichern
FDFF   60         RTS         Rücksprung

****************************  Parameter für aktives
                              File setzen

Einsprung von $FFBA

FE00   85 B8      STA $B8     logische Filenummer
FE02   86 BA      STX $BA     Geräteadresse
FE04   84 B9      STY $B9     Sekundäradresse
FE06   60         RTS         Rücksprung
```

```
****************************** Status holen

Einsprung von $FFB7

FE07   A5 BA      LDA $BA     Gerätenummer holen
FE09   C9 02      CMP #$02    gleich 2 ? (RS 232)
FE0B   D0 0D      BNE $FE1A   nein
FE0D   AD 97 02   LDA $0297   RS 232-Status holen
FE10   48         PHA         und auf Stapel retten
FE11   A9 00      LDA #$00    Status
FE13   8D 97 02   STA $0297   löschen
FE16   68         PLA         und Statuswert zurückholen
FE17   60         RTS         Rücksprung

****************************** Flag für Betriebssystem-
                               meldungen setzen

Einsprung von $FF90

FE18   85 9D      STA $9D     Ausgabeflag (Direktmodus)
FE1A   A5 90      LDA $90     Statusflag holen

****************************** Status setzen

Einsprung von $EDB2, $EE4F, $F18A, $F518, $FA81, $FAC6
              $FB35, $FB88

FE1C   05 90      ORA $90     Statusflag testen und
FE1E   85 90      STA $90     wieder abspeichern
FE20   60         RTS         Rücksprung

****************************** Timeout-Flag für IEC setzen

Einsprung von $FFA2

FE21   8D 85 02   STA $0285   Timeout-disable
FE24   60         RTS         Rücksprung
```

```
****************************** MEMTOP, Obergrenze des
                                BASIC-RAM holen/setzen

Einsprung von $FF99

FE25   90 06      BCC $FE2D   C=0: Adresse setzen

Einsprung von $F282, $F468

FE27   AE 83 02   LDX $0283   Carry gesetzt
FE2A   AC 84 02   LDY $0284   Adresse nach X/Y holen

Einsprung von $F480, $FD8D

FE2D   8E 83 02   STX $0283   Carry gelöscht
FE30   8C 84 02   STY $0284   X/Y nach Adresse setzen
FE33   60         RTS         Rücksprung

***************************** MEMBOT, Untergrenze des
                                BASIC-RAM holen/setzen

Einsprung von $FF9C

FE34   90 06      BCC $FE3C   C=0: Adresse setzen
FE36   AE 81 02   LDX $0281   Carry gesetzt
FE39   AC 82 02   LDY $0282   Adresse nach X/Y holen
FE3C   8E 81 02   STX $0281   Carry gelöscht
FE3F   8C 82 02   STY $0282   Adresse aus X/Y setzen
FE42   60         RTS         Rücksprung

***************************** NMI Einsprung
FE43   78         SEI         Interrupt setzen
FE44   6C 18 03   JMP ($0318) JMP $FE47, NMI-Vektor

Einsprung von $FE44

FE47   48         PHA         Akku auf Stapel retten
FE48   8A         TXA         X nach Akku
FE49   48         PHA         X retten
FE4A   98         TYA         Y nach Akku
```

```
FE4B  48        PHA          Y retten
FE4C  A9 7F     LDA #$7F     Wert laden
FE4E  8D 0D DD  STA $DD0D    NMI-Möglichkeiten löschen
FE51  AC 0D DD  LDY $DD0D    Flags lesen und löschen
FE54  30 1C     BMI $FE72    RS 232 aktiv ?
FE56  20 02 FD  JSR $FD02    Prüft auf ROM-Modul in $8000
FE59  D0 03     BNE $FE5E    nein: weiter
FE5B  6C 02 80  JMP ($8002)  ja: Sprung auf Modul-NMI

FE5E  20 BC F6  JSR $F6BC    Flag für Stop-Taste setzen
FE61  20 E1 FF  JSR $FFE1    Stop-Taste abfragen
FE64  D0 0C     BNE $FE72    nicht gedrückt ?

Einsprung von $FF55

FE66  20 15 FD  JSR $FD15    Standard-Vektoren für
                             Interrupt und I/O setzen
FE69  20 A3 FD  JSR $FDA3    I/O initialisieren
FE6C  20 18 E5  JSR $E518    Bildschirmreset
FE6F  6C 02 A0  JMP ($A002)  zum BASIC-Warmstart

**************************** NMI-Routine für RS 232
FE72  98        TYA          ICR-Register
FE73  2D A1 02  AND $02A1    mit RS 232 NMI-Flag verknüp.
FE76  AA        TAX          nach X retten
FE77  29 01     AND #$01     Sendebetrieb aktiv ?
FE79  F0 28     BEQ $FEA3    nein
FE7B  AD 00 DD  LDA $DD00    Datenport lesen
FE7E  29 FB     AND #$FB     Bit 2 TXD löschen
FE80  05 B5     ORA $B5      zu sendendes Bit übergeben
FE82  8D 00 DD  STA $DD00    und wieder in Datenport spei.
FE85  AD A1 02  LDA $02A1    RS-232 NMI-Flag
FE88  8D 0D DD  STA $DD0D    wieder in ICR schreiben
FE8B  8A        TXA          Wert aus X zurückholen
FE8C  29 12     AND #$12     Bit 1 und 4 isolieren
FE8E  F0 0D     BEQ $FE9D    Bit 1 und 4=0: Bit empfangen
FE90  29 02     AND #$02     Bit 1, Aufruf von Timer B
FE92  F0 06     BEQ $FE9A    nein: verzweige zu Startbit
FE94  20 D6 FE  JSR $FED6    empfangenes Bit verarbeiten
```

```
FE97   4C 9D FE   JMP $FE9D   Vorbereitung für Byte umgehen
FE9A   20 07 FF   JSR $FF07   Vorbereitung für Empfang
                              des nächsten Bytes

Einsprung von $FE97

FE9D   20 BB EE   JSR $EEBB   Empfang des nächsten Bits v.
FEA0   4C B6 FE   JMP $FEB6   Rückkehr vom Interrupt
FEA3   8A         TXA         X nach Akku
FEA4   29 02      AND #$02    Datenempfang ?
FEA6   F0 06      BEQ $FEAE   verzweige wenn kein Empfang
FEA8   20 D6 FE   JSR $FED6   empfangenes Bit verarbeiten
FEAB   4C B6 FE   JMP $FEB6   Rückkehr vom Interrupt
FEAE   8A         TXA         X nach Akku
FEAF   29 10      AND #$10    warten auf Startbit ?
FEB1   F0 03      BEQ $FEB6   verzweige wenn kein Startbit
FEB3   20 07 FF   JSR $FF07   Vorbereitung für Empfang
                              des nächsten Bytes

Einsprung von $FEA0, $FEAB

FEB6   AD A1 02   LDA $02A1   RS-232 NMI-Flag
FEB9   8D 0D DD   STA $DD0D   wieder in ICR
FEBC   68         PLA         Y-Register vom Stapel
FEBD   A8         TAY         zurückholen
FEBE   68         PLA         X-Register
FEBF   AA         TAX         zurückholen
FEC0   68         PLA         Akku zurückholen
FEC1   40         RTI         Rücksprung

**********        Timerkonstanten für RS 232 Baud-Rate,
                  NTSC-Version
FEC2   C1 27      $27C1  = 10177     50 Baud
FEC4   3A 1A      $1A3E  =  6718     75 Baud
FEC6   C5 11      $11C5  =  4549    110 Baud
FEC8   74 0E      $0E74  =  3700    134.5 Baud
FECA   ED 0C      $0CED  =  3309    150 Baud
FECC   45 06      $0645  =  1605    300 Baud
FECE   F0 02      $02F0  =   752    600 Baud
FED0   46 01      $0146  =   326   1200 Baud
```

```
FED2   B8 00        $00B8  =   184     1800 Baud
FED4   71 00        $0071  =   113     2400 Baud

**************************   NMI-Routine für RS-232
                             Eingabe

Einsprung von $FE94, $FEA8

FED6   AD 01 DD   LDA $DD01   Port Register B
FED9   29 01      AND #$01    Bit für Receive Data isolie-
                              ren
FEDB   85 A7      STA $A7     und speichern
FEDD   AD 06 DD   LDA $DD06   Timer B LOW
FEE0   E9 1C      SBC #$1C    minus 28
FEE2   6D 99 02   ADC $0299   + LOW-Byte der Baudrate
FEE5   8D 06 DD   STA $DD06   wieder abspeichern
FEE8   AD 07 DD   LDA $DD07   RS 232 Timerkon. für Baudrate
FEEB   6D 9A 02   ADC $029A   HIGH-Byte addieren
FEEE   8D 07 DD   STA $DD07   in Timer schreiben
FEF1   A9 11      LDA #$11    Timer B starten
FEF3   8D 0F DD   STA $DD0F   Control Register B
FEF6   AD A1 02   LDA $02A1   CIA 2 NMI-Flag holen
FEF9   8D 0D DD   STA $DD0D   Interrupt Control Register
FEFC   A9 FF      LDA #$FF    Wert laden
FEFE   8D 06 DD   STA $DD06   und damit
FF01   8D 07 DD   STA $DD07   Timer setzen
FF04   4C 59 EF   JMP $EF59   Bit holen

**************************   NMI-Routine RS 232 Ausgabe

Einsprung von $FE9A, $FEB3

FF07   AD 95 02   LDA $0295   LOW- und HIGH-Byte
FF0A   8D 06 DD   STA $DD06   holen und in
FF0D   AD 96 02   LDA $0296   RS 232 Timerkonstanten für
FF10   8D 07 DD   STA $DD07   Baudrate
FF13   A9 11      LDA #$11    Timer B starten
FF15   8D 0F DD   STA $DD0F   Control Register B
FF18   A9 12      LDA #$12    Bit 1 und 4 für Verknüpfung
FF1A   4D A1 02   EOR $02A1   mit NMI-Flag für CIA 2
```

```
FF1D  8D A1 02   STA $02A1   Wert wieder speichern
FF20  A9 FF      LDA #$FF    höchsten Wert laden
FF22  8D 06 DD   STA $DD06   und in Latch von
FF25  8D 07 DD   STA $DD07   Timer B laden
FF28  AE 98 02   LDX $0298   Anzahl der zu sendenden Bits
FF2B  86 A8      STX $A8     in Zähler für Wortlänge
FF2D  60         RTS         Rücksprung

**************************   Timerwert für Sendebaudrate
                             ermitteln

Einsprung von $F44A

FF2E  AA         TAX         Baudrate aus Tabelle nach X
FF2F  AD 96 02   LDA $0296   HIGH-Byte holen
FF32  2A         ROL         mal 2
FF33  A8         TAY         nach Y retten
FF34  8A         TXA         LOW-Byte holen
FF35  69 C8      ADC #$C8    plus 200
FF37  8D 99 02   STA $0299   nach Timerwert LOW
FF3A  98         TYA         HIGH-Byte zurückholen
FF3B  69 00      ADC #$00    Übertrag addieren
FF3D  8D 9A 02   STA $029A   nach Timerwert HIGH
FF40  60         RTS         Rücksprung

FF41  EA         NOP         No OPeration
FF42  EA         NOP         No OPeration

**************************   Einsprung aus Bandroutine

Einsprung von $F927

FF43  08         PHP         Statusregister auf Stapel
FF44  68         PLA         Statusregister in Akku
FF45  29 EF      AND #$EF    Break-Flag löschen
FF47  48         PHA         und wieder auf Stapel legen

**************************   IRQ-Einsprung
FF48  48         PHA         Akku auf Stapel retten
FF49  8A         TXA         X nach Akku
```

```
FF4A   48         PHA          X-Register retten
FF4B   98         TYA          Y nach Akku
FF4C   48         PHA          Y-Register retten
FF4D   BA         TSX          Stapelzeiger als Zähler in X
FF4E   BD 04 01   LDA $0104,X  Break-Flag vom Stapel holen
FF51   29 10      AND #$10     und testen
FF53   F0 03      BEQ $FF58    nicht gesetzt
FF55   6C 16 03   JMP ($0316)  BREAK - Routine
FF58   6C 14 03   JMP ($0314)  Interrupt - Routine

**************************   Video-Reset

Einsprung von $FCFB, $FF81

FF5B   20 18 E5   JSR $E518    Videocontroller initialisie-
                               ren
FF5E   AD 12 D0   LDA $D012    Rasterzeile
FF61   D0 FB      BNE $FF5E    wartet auf Ende Videozeile
FF63   AD 19 D0   LDA $D019    Interrupt durch Rasterzeile?
FF66   29 01      AND #$01     Bit 0 isolieren und als Flag
FF68   8D A6 02   STA $02A6    PAL/NTSC-Version merken
FF6B   4C DD FD   JMP $FDDD    Interrupttimer setzen

**************************   Timer für Interrupt setzen

Einsprung von $FDF6

FF6E   A9 81      LDA #$81     Timer A Unterlauf
FF70   8D 0D DC   STA $DC0D    Interrupt Control Register
FF73   AD 0E DC   LDA $DC0E    Control Register A
FF76   29 80      AND #$80     Bit 7 retten
                               Uhrzeittrigger (50/60 Hz)
FF78   09 11      ORA #$11     Timer A starten
FF7A   8D 0E DC   STA $DC0E    Control Register A
FF7D   4C 8E EE   JMP $EE8E    seriellen Takt aus
FF80   00         BRK          BReaK
```

```
**************************  Sprungtabelle für
                            Betriebssystem-Routinen

FF81  4C 5B FF  JMP $FF5B   Video-Reset

FF84  4C A3 FD  JMP $FDA3   CIAs initialisieren

FF87  4C 50 FD  JMP $FD50   RAM löschen bzw. testen

FF8A  4C 15 FD  JMP $FD15   I/O initialisieren

FF8D  4C 1A FD  JMP $FD1A   I/O Vektoren initialisieren

**************************  Einsprung von $A47D, $A874

FF90  4C 18 FE  JMP $FE18   Status setzen

FF93  4C B9 ED  JMP $EDB9   Sekundäradresse
                            nach LISTEN senden

FF96  4C C7 ED  JMP $EDC7   Sekundäradresse
                            nach TALK senden

**************************  Einsprung von $E40B

FF99  4C 25 FE  JMP $FE25   RAM-Ende setzen/holen

**************************  Einsprung von $E403

FF9C  4C 34 FE  JMP $FE34   RAM-Anfang setzen/holen

FF9F  4C 87 EA  JMP $EA87   Tastatur abfragen

FFA2  4C 21 FE  JMP $FE21   Time-out-Flag für
                            IEC-Bus setzen

FFA5  4C 13 EE  JMP $EE13   Eingabe vom IEC-Bus

FFA8  4C DD ED  JMP $EDDD   Ausgabe vom IEC-Bus
```

```
FFAB   4C EF ED   JMP $EDEF   UNTALK senden

FFAE   4C FE ED   JMP $EDFE   UNLISTEN senden

FFB1   4C 0C ED   JMP $ED0C   LISTEN senden

FFB4   4C 09 ED   JMP $ED09   TALK senden

**************************   Einsprung von $ABDD, $AF9A, $E180, $E195

FFB7   4C 07 FE   JMP $FE07   Status holen

**************************   Einsprung von $E1DD, $E1F0, $E1FD, $E22B,
                              $E23F, $E24E

FFBA   4C 00 FE   JMP $FE00   Fileparameter setzen

**************************   Einsprung von $E1D6, $E21B, $E261

FFBD   4C F9 FD   JMP $FDF9   Filenamenparameter setzen

**************************   Einsprung von $E1C1

FFC0   6C 1A 03   JMP ($031A) $F34A OPEN

**************************   Einsprung von $E1CC

FFC3   6C 1C 03   JMP ($031C) $F291 CLOSE

**************************   Einsprung von $E11E

FFC6   6C 1E 03   JMP ($031E) $F20E CHKIN Eingabeg. setzen

**************************   Einsprung von $E4AE

FFC9   6C 20 03   JMP ($0320) $F250 CKOUT Ausgabegerät set.

**************************   Einsprung von $A447, $ABB7, $E37B, $F6F4,
                              $F716
```

```
FFCC   6C 22 03   JMP ($0322) $F333 CLRCH Ein-Ausgabe
                              zurücksetzen
**************************   Einsprung von $E112

FFCF   6C 24 03   JMP ($0324) $F157 BASIN  Eingabe
                              eines Zeichens

**************************    Einsprung von $E10C, $F135, $F5C9, $F726,
                              $F759

FFD2   6C 26 03   JMP ($0326) $F1CA BSOUT  Ausgabe
                              eines Zeichens

**************************   Einsprung von $E175

FFD5   4C 9E F4   JMP $F49E   LOAD

**************************   Einsprung von $E15F

FFD8   4C DD F5   JMP $F5DD   SAVE

**************************   Einsprung von $AA1A

FFDB   4C E4 F6   JMP $F6E4   Time setzen

**************************   Einsprung von $AF84

FFDE   4C DD F6   JMP $F6DD   Time holen

**************************    Einsprung von $A82C, $F4F9, $F62E, $F8D0,
                              $FE61

FFE1   6C 28 03   JMP ($0328) $F6ED STOP-Taste abfragen
**************************   Einsprung von $E124

FFE4   6C 2A 03   JMP ($032A) $F13E GET
**************************   Einsprung von $A660

FFE7   6C 2C 03   JMP ($032C) $F32F CLALL
```

```
*****************************  Einsprung von $EA31

FFEA   4C 9B F6   JMP $F69B   Time erhöhen

FFED   4C 05 E5   JMP $E505   SCREEN  Anzahl Zeilen
                              und Spalten holen

*****************************  Einsprung von $AAE9, $AAFA, $B39F

FFF0   4C 0A E5   JMP $E50A   Cursor setzen /
                              Cursorposition holen

*****************************  Einsprung von $E09E

FFF3   4C 00 E5   JMP $E500   Startadresse des
                              I/O-Bausteins holen

FFF6   52 52 42 59z

*****************************  Hardware Vektorenz

FFFA   43 FE      $FE43       NMI Vektor

FFFC   E2 FC      $FCE2       RESET Vektor

FFFE   48 FF      $FF48       IRQ Vektor
```

SX-64 Betriebssystem

Im folgenden sind die Abweichungen des SX 64 Betriebssystems gegenüber dem normalen C64 aufgelistet. Sie beziehen sich hauptsächlich auf geänderte Einschaltmeldung, andere Farbkombinationen nach dem Einschalten sowie auf das Fehlen des Anschlusses für die Datasette (Geräteadresse 1)

```
*****************************  Einschaltmeldung
E479 20 2A 2A 2A 2A 2A 20 20   *****
E481 53 58 2D 36 34 20 42 41   SX-64 BA
E489 53 49 43 20 56 32 2E 30   SIC V2.0
```

```
E491 20 20 2A 2A 2A 2A 2A 0D    *****
E4AC B3
****************************  RS-232 Routinen
E4D3   85 A9      STA $A9     in Speicher für Band
E4D5   A9 01      LDA #$01    Wert laden und
E4D7   85 AB      STA $AB     in Speicher für Band
E4D9   60         RTS         Rücksprung

****************************  Hintergrundfarbe setzen
E4DA   AD 86 02   LDA $0286   aktuelle Textfarbe
E534   A9 06      LDA #$06    Einschaltfarbe blau

****************************  Cursorposition berechnen
E57C   20 F0 E9   JSR $E9F0   Zeiger auf Videoram setzen
E57F   A9 27      LDA #$27    39
E581   E8         INX         Zeiger erhöhen
E582   B4 D9      LDY $D9,X   MSB der Startadresse der
                               Zeile
E584   30 06      BMI $E58C   N=1 : $E58C
E586   18         CLC         Carry löschen (Addition)
E587   69 28      ADC #$28    40 addieren
E589   E8         INX         Zeiger erhöhen
E58A   10 F6      BPL $E582   Fertig? nein: nächste Zeile
E58C   85 D5      STA $D5     Länge der Zeile speichern
E58E   4C 24 EA   JMP $EA24   Zeiger auf Farbram berechnen
E591   E4 C9      CPX $C9     mit Cursorzeile für Eingabe
E593   F0 03      BEQ $E598   vergleichen, gleich: RTS
E595   4C ED E6   JMP $E6ED   MSB's für Zeile berechnen
E598   60         RTS         Rücksprung
E599   EA         NOP         No OPeration

****************************  Text nach Drücken von
                               Shift RUN/STOP
E5EE   A2 0F      LDX #$0F    15 Zeichen
E5F0   78         SEI         Interrupt setzen
E5F1   86 C6      STX $C6     Anzahl der Zeichen im
                               Tastaturpuffer
E5F3   BD D7 F0   LDA $F0D7,X Text LOAD":*",8 <CR> RUN <CR>
E5F6   9D 76 02   STA $0276,X in Tastaturpuffer schreiben
```

```
*****************************
E621   20 91 E5   JSR $E591   Cursorposition Bildschirm

*****************************  Bildschirmzeile löschen
EA07   20 DA E4   JSR $E4DA   Cursorfarbe setzen
EA0A   A9 20      LDA #$20    ' ' Leerzeichen
EA0C   91 D1      STA ($D1),Y in Videoram schreiben
EA0E   88         DEY         Zähler vermindern
EA0F   10 F6      BPL $EA07   Fertig? nein: weiter
EA11   60         RTS         Rücksprung
EA12   EA         NOP         No OPeration

*****************************
ECD9 03 01

*****************************
EF94   4C D3 E4   JMP $E4D3   zur geänderten RS 232-Routine

*****************************  Text nach Shift RUN/STOP
F0D8 4C 4F 41 44 22 3A 2A 22  LOAD":*"
F0E0 2C 38 0D 52 55 4E 0D     ,8 RUN

*****************************  Ignorieren der Geräte-
                              adresse 1 (Datasette)
F386   D0 08      BNE $F390   zu 'ILLEGAL DEVICE NUMBER'

F4F6   90 85      BCC $F47D   Memory top setzen

F5F8   90 F7      BCC $F5F1   zu 'ILLEGAL DEVICE NUMBER'

FF80 43
```

7. C64 Pflegen und Warten

7.1 Allgemeines zu diesem Kapitel

Das Ziel dieses Kapitels ist es, Ihren C64 im bescheidenen Maßstab aufzurüsten und im Falle eines Fehlers den Fehler selbst zu lokalisieren und ihn gegebenenfalls zu reparieren.

Falls die Garantie Ihres Geräts noch nicht abgelaufen ist, sollten sie es sich vorher gut überlegen, ob Sie Ihr Gerät aufschrauben, und damit Ihren Garantieanspruch verlieren.

Bevor Sie nach den, in diesem Kapitel gegebenen Kurzanleitungen, Geräte aufschrauben, ICs austauschen oder sonstiges unternehmen, untersuchen Sie als erstes, ob Sie wirklich alle nötigen Geräte ordnugsgemäß angeschlossen und eingeschaltet haben. Wenn sich bei einem Gerät überhaupt nichts tut, kontrollieren Sie als erstes die Kabel und Sicherungen. Wenn Sie im Herausnehmen von ICs noch keine Erfahrung haben, und dies in der entsprechenden Anleitung empfohlen wird, so lesen Sie sich vorher unbedingt Kapitel 7.9 durch. Nehmen Sie auch keine Eingriffe vor, die Sie sich nicht selbst zutrauen, oder die Ihren Erfahrungsbereich weit übersteigen. Überschätzen Sie sich nicht, denn das kann Sie weitaus mehr kosten als die Reparatur bei einem Fachmann.

In den folgenden Absätzen werden wir auf die häufigsten Fehlertypen näher eingehen:

7.2 Der Bildausfall

Unter Bildausfall verstehen wir, daß nach dem Einschalten des Computers und des entsprechenden Datensichtgeräts (Monitor, Fernseher) kein Bild erscheint, obwohl die LED am Computer leuchtet.

Falls dies der Fall ist, so ist mit größter Warscheinlichkeit ein Teil der Stromzuleitung unterbrochen, welches meistens auf eine

defekte Sicherung zurückzuführen ist. Um dies festzustellen, lösen Sie die drei Schrauben an der Unterseite des Computers und klappen den oberen Teil vorsichtig nach hinten weg. Den Standort der Sicherung können Sie in den nachfolgenden Bildern ermitteln (die Stärke der Sicherung ist entweder 1 oder 1.25 Ampere). Anschließend nehmen Sie die Sicherung vorsichtig aus der Fassung und ersetzen sie gegebenenfalls durch eine 1.25 Ampere tragende Sicherung. Es ist häufig der Fall, daß die Sicherung ohne besonderen Grund durchbrennt. Der Ersatz durch eine 1.25 Ampere starke Sicherung ist völlig ohne Risiko, dagegen kann eine Überbrückung der Sicherung durch ein Stück Draht zu großen Schäden führen.

Ist die Sicherung in Ordnung, liegt der Fehler wahrscheinlich beim Transformator. Falls Sie über das entsprechende Meßgerät verfügen, können Sie in der DIN-Buchse, die im Kapitel 7.9 beschrieben werden, nachmessen. Oft fehlt die 5 VOLT Gleichspannung, die von der Stabilisierungsschaltung im Netzteil geliefert werden soll. Aber auch die 9 VOLT Wechselspannung ist vereinzelt nicht vorhanden.

Wenn Sie allerdings nicht die Möglichkeit haben, das Netzteil durchzumessen, tauschen Sie Ihr Netzteil durch das einens Freundes aus. Sollte sich herausstellen, daß der Fehler im Netzteil lag, so bringen Sie dieses zu Ihrem Fachhändler.

Sollte sich herausstellen, daß auch das Netzteil funktioniert, so überprüfen Sie ihr Anschlußkabel an den Monitor oder Fernseher. Sind die Kabel in Ordnung, so könnte der Modulator defekt sein. Um dieses zu überprüfen, bedarf es eines kleinen Tricks:

Schließen sie Ihre Floppy oder Ihre Datasette an den Computer an. Im Falle der Datasette drücken Sie SHIFT/RUN-STOP und daraufhin die PLAY-Taste. Beginnt der Motor zu laufen, so wird der Modulator defekt sein und Ihr Computer muß in die Werkstatt. Ist eine Floppy angeschlossen, so tippen Sie blind LOAD"$",8. Beginnt der Laufwerksmotor sich zu drehen, so gilt das oben gesagte.

Wenn nicht einmal die LED an Ihrem Computer leuchtet, so sollten Sie die Sicherung an Ihrem Netzteil überprüfen und gegebenenfalls durch eine gleicher Stärke ersetzen. Es ist auch möglich, daß Ihre Sicherung sich nur gelockert hat. Trifft beides nicht zu, so sollten Sie den Stecker für die Steckdose überprüfen.

7.3 Nur Bildschirm und Rahmen

Häufiger aber auch schwerwiegender sind die Fehler, wenn nach dem Einschalten der Rahmen und die Hintergrundfarbe erscheinen, danach jedoch jede Aktivität stoppt. In diesem Fall sollten Sie zuerst einmal die IRQ-Leitung messen. Im Normalfall, also bei funktionierendem Gerät, tritt an diesem Anschluß (Pin 3 des Prozessors) alle 16 Millisekunden ein negativer Impuls von ca. 200 Microsekunden auf. Mit einem Vielfachmeßgerät ist nur eine Gleichspannung von ca. 4.5 VOLT zu messen. Mit einer LED und einem Widerstand von 200 Ohm kann man oben sehen, ob die Interrupts ausgelöst werden. Dazu wird der Widerstand mit 5 VOLT verbunden und als Vorwiderstand für die LED benutzt. Die Kathode der LED wird an Pin 3 des Prozessors gehalten. Jetzt muß die LED soeben sichtbar leuchten und etwas flackern. Besser geeignet ist natürlich ein Logik-Tester oder ein Oszilloskop.

Was aber, wenn die Interrupts ausbleiben? Wenn die IRQ-Leitung nach dem Einschalten HIGH ist, dann nach kurzer Zeit LOW wird, kommen als Fehlerquellen das Betriebssystem-ROM, das RAM, die CIAs oder der Prozessor in Frage. Ein Fall für eine gut ausgerüstete Werkstatt, wenn Sie nicht die Möglichkeit haben, die ICs aus einem anderen Rechner zu probieren.

Sind die CIAs defekt, so ist dieser Fehler leicht festzustellen. Der C64 kann ohne CIA 2 arbeiten, und deshalb ist es möglich, die beiden ICs, deren Lage anhand der Bilder ermittelt werden kann, auszutauschen. Falls der Computer nun wieder funktioniert, müssen Sie das defekte IC, das vorher im Sockel von CIA 1 steckte, austauschen.

Hat diese Operation auch nicht den gewünschten Erfolg, können Sie untersuchen, ob der Prozessorport defekt ist.

Um dieses feststellen zu können, sollten Sie jedoch schon einige Erfahrung im Umgang mit ICs haben.

Dazu messen Sie die Leitungen -LORAM, -HIRAM und -CHAREN an den Pins 27, 28 und 29 des Prozessors. Diese Leitungen sollten nach dem Einschalten auf HIGH liegen. Stellt sich nach dem Einschalten aber ein LOW-Pegel an einem dieser Pins ein, so kann der Rechner das Betriebsystem-ROM nicht ordnungsgemäß adressieren. Es wird ausgeblendet und das darunterliegende RAM kommt zum Vorschein. Folglich kann der Computer die RESET-Routine nicht anspringen, was unweigerlich zum 'Absturz' führt.

Stellt sich also ein LOW-Pegel ein, so können Sie jetzt den Prozessorport untersuchen. Dafür löten Sie den Prozessor aus und löten eine 40-beinige Fassung an diese Stelle. Nun müssen Sie die Pins 27, 28 und 29 des Prozessors rechtwinklig abbiegen und ihn zurück in die Fassung stecken, so daß diese Pins keinen Kontakt zur Platine haben. Auf diese Art wird dem Ardeß-Manager U17 vorgegaukelt, daß alles in Ordnung ist, da ein offener Eingang an TTL-ICs als HIGH gewertet wird. Wenn der Rechner nach dieser Prozedur arbeitet, so ist ein neuer Prozessor nötig.

Schlägt diese Methode fehl, so ist es unumgänglich, den Computer in eine Werkstatt zu geben, da der Fehler mit kaum mit normalen Mitteln zu lokalisieren ist.

7.4 Farbige Zeichen auf dem Bildschirm

Wenn Sie zum Beispiel beim Listen einer Directory oder eines BASIC-Programms ungewollt lauter farbige Zeichen auf dem Bildschirm erhalten, so muß dies nicht unbedingt am VIC liegen. Der Fehler kann auch an dem Netzteil liegen, welches zu wenig Spannung liefert. Um dieses zu überprüfen, messen Sie die Spannung nach, oder tauschen Sie Ihr Netzteil aus.

7.5 Die Tastatur funktioniert nicht richtig!

In diesem Fall liegt bestimmt ein Fehler der CIA 1 vor, die für die Tastaturabfrage zuständig ist. Gewißheit darüber können Sie sich jedoch mit dem Testprogramm verschaffen.

7.6 Der Joystick funktioniert nicht!

Hier liegt der Fehler, sofern keine Störungen der Tastatur vorliegen, ausschließlich am Joystick selbst. Mit Hilfe des Testprogramms können Sie die Joystickfunktionen überprüfen.

7.7 Wenn er nicht richtig lädt!

Hier muß man zwischen Datasette und Floppy unterscheiden. Beim Bandbetrieb gibt es drei Fehlermöglichkeiten:

In der Datasette selbst:

Der Tonkopf ist dejustiert. Mit einem kleinen Schraubenzieher kann man versuchen, die Stellung des Tonkopfs anhand der Justageschraube zu justieren (merken Sie sich die ursprüngliche Einstellung, damit Sie, falls dies nicht die Fehlerquelle war, sich nicht die Datasette selbst dejustieren). Eine weitere Möglichkeit ist der verschmutzte Tonkopf, den man mit einer Reinigungskassette oder einem Tonkopfspray säubern kann.

Im Computer:

Auch hier muß zwischen zwei Fällen unterschieden werden:

1. Es wurde nicht richtig gesavet, was auf einen defekten Prozessorport zurückzuführen ist. In diesem Fall müssen Sie den Computer in die Reparatur schicken.

2. Es wird nicht richtig geladen, was auf eine defekte CIA 1 schließen läßt. In diesem Fall können Sie, wie im Kapitel 7.3 ab Absatz 3 erläutert wird, die beiden CIAs vertauschen.

Die Floppy lädt nicht!

Falls der Fehler im Computer liegt, so kann dies nur bei der CIA 2 der Fall sein, da diese für die Kommunikation mit der Floppy zuständig ist. Hier ist es ratsam, die CIA 2 mit der eines Freundes auszutauschen, um sich Gewißheit zu verschaffen. Wenn dies nicht hilft, so müßte der Fehler in der Floppy liegen, die man, wenn es nicht auf eine defekte Sicherung in der Floppy zurückzuführen ist, zu einer Reparaturwerkstatt bringen muß.

7.8 Fehler, die nach längerem Betrieb auftreten

Zeitweise oder spontan auftretende Fehler sind in einer Werkstatt immer die unbeliebtesten Defekte. Da kann es dann ohne weiteres vorkommen, daß ein Gerät mehrere Tage im Test ordnungsgemäß funktioniert, obwohl der Kunde als Fehler angegeben hat, daß sich das Gerät nach 1/2 Stunde 'verabschiedet'. Häufige Ursache dieser Fehler ist ein thermischer Defekt in einem Bauteil. Sollte Ihr C64 irgendwann einmal solche Symptome zeigen, so empfiehlt sich die Anschaffung einer großen Dose Kältespray. Dieses Spray ist im Elektronikfachhandel für ein paar DM zu erhalten. Durch einfaches Ansprühen lassen sich die Bauteile auf bis zu -40 Grad abkühlen. Dabei hat sich das folgende Prinzip bewährt:

Nachdem die Tastatur entfernt ist, wird die Rechnerplatine mit einem normalen Haarfön gleichmäßig erhitzt. Sobald der Rechner einen Fehler zeigt, werden die einzelnen Bauteile systematisch mit dem Spray gekühlt und der Rechner nach jedem Bauteil aus- und wieder eingeschaltet. Sobald nach dem Einschalten der Rechner wieder funktioniert, werden die zuletzt abgekühlten Bauteile nocheinmal erwärmt, um zu sehen, ob wirklich eines

dieser Bauteile den Defekt verursachte. Auf diese Weise kann man nun das defekte Bauteil immer weiter einkreisen und zuletzt austauschen.

Leider sind nicht nur die Bauteile als Fehlerursache möglich. Auch die Leiterplatte kann als Ursache in Frage kommen. Durch die Erwärmung dehnt sie das Material aus, und obwohl diese Ausdehnung sehr gering ist, können winzige Haarrisse entstehen, die dann zu Versagen des Gerätes führen. Diese Haarrise zu finden ist nur mit viel Glück möglich. Glücklicherweise ist das im C64 verwendete Leiterplattenmaterial von sehr guter Qualität, so daß diese Fehler ausgesprochen selten sind.

Eine Ursache für sporadisch auftretende Fehler können auch die RAMs sein. Mit dem in diesem Kapitel abgedrucktem Testprogramm ist es möglich, das gesamte RAM des C64 zu prüfen, und das vermutlich fehlerhafte RAM-IC anzuzeigen.

7.9 Das Herausnehmen von ICs

Diese Anleitung gilt nur für ICs, die gesockelt sind. Sollte dies nicht der Fall sein, überlassen sie diese Arbeit einem Fachmann.

Nehmen Sie einen kleinen Schraubenzieher und schieben Sie ihn vorsichtig unter die eine der kurzen Seiten des ICs. Drücken Sie den Schraubenzieher nun allmählich nach unten, um den IC an der Stelle etwas aus der Fassung zu heben. Genauso verfahren Sie an der anderen Seite. Achten Sie dabei darauf, daß Sie das IC nicht einseitig zuweit anheben, weil dies zum Verbiegen der noch steckenden Beinchen auf der anderen Seite des ICs zur Folge haben könnte. Wenn Sie das IC nun soweit aus der Fassung gehoben haben, daß Sie es mühelos mit Daumen und Zeigefinger, die die beiden Enden des ICs halten, herausziehen können, dann nehmen Sie es an beiden Enden gleichmäßig ziehend aus der Fassung heraus.

Fassen Sie die Beinchen des ICs nicht an, weil Sie statisch aufgeladen sein können, was eine Zerstörung des ICs zur Folge haben könnte.

7.10 Wie stelle ich mir meine Tastatur strammer?

Dieser kleine Trick ist sehr nützlich, wenn man die Tastatur satt hat. Alle Tasten des C64 sind mit Federn ausgestattet, die die Tasten vom Kontakt auf der Tastaturplatine wegdrücken. Die Tasten sind auf einem kleinen Stäbchen festgesteckt. Mit Hilfe eines Schraubenziehers kann man die Tasten vom Keyboard lösen.

Man schiebt den Schraubenzieher unter eine Taste und drückt dann vorsichtig den Schraubenzieher nach unten, um die Taste aus der Halterung zu hebeln. Gleichzeitig zieht man die Taste mit Daumen und Zeigefinger nach oben. Nachdem Sie die Taste entfernt haben, nehmen Sie die Feder heraus, die ca. 1 cm lang ist. Sie können sie jetzt vorsichtig auseinanderziehen. Es ist nicht ratsam, sie länger als 1.5 cm zu ziehen, da der Anschlag sonst zu hart wird.

Nachdem Sie die Feder präpariert haben, setzen Sie sie wieder auf ihren alten Platz. Dann stecken Sie die Taste auf den Pin und pressen sie nach unten, bis sie einrastet.

Die Taste hat nun einen härteren Anschlag. Am besten stellt man sich die Tasten härter, die am meisten benutzt werden, zum Beispiel die RETURN-, RUN-STOP- und RESTORE-Tasten.

7.11 Wie baue ich einen RESET-Taster ein?

Bevor wir in die Praxis übergehen, wollen wir Ihnen zuvor erst die Arbeitsweise eines RESETs erklären. Beim Auslösen eines Hardware-RESETs geschieht das gleiche wie beim Einschalten des Computers. Die RESET-Leitung des Computers, die mit den wichtigsten Bauteilen verbunden ist, wird kurzzeitig auf LOW (MASSE) gelegt. Daraufhin springt der Prozessor die RESET-Routine, deren Vektor in Speicheradresse $FFFC und $FFFD liegt, an. In dieser Routine werden die wichtigsten Bausteine initialisiert, wärend der Speicherinhalt zum größten Teil erhalten

bleibt. In unserer Anleitung haben wir die einfachste Möglichkeit zum Bau eines RESET-Tasters gewählt. Es werden die Leitungen RESET und MASSE des USER-PORTs per Taster miteinander verbunden.

Zum Bau benötigen Sie folgende Bauteile:

Einen Taster und zwei dünne Kabel

Falls Sie nichts an Ihren Computer anlöten wollen, dann besorgen Sie sich noch einen USER-PORT-Stecker und löten die beiden Kabel nicht direkt an den USER-PORT, sondern an den Stecker.

Jetzt löten Sie die beiden Kabelenden an die Pin 1 und 3 des USER-Ports oder des Steckers. Die Belegung des USER-Ports finden sie am Ende des Kapitels 5.

Drücken Sie nun auf den Taster, und Sie werden feststellen, daß Ihr Rechner einen RESET ausführt. Den gleichen Vorgang können Sie auch mit SYS64738 vom BASIC aus erzielen.

7.12 Das Testprogramm

Ans Ende dieses Kapitels haben wir noch ein kleines Testprogramm gehängt, mit dessen Hilfe Sie das RAM, den Soundchip und Ihren Joystick überprüfen können. Außerdem enthält es noch ein Testbild, mit dem Sie die Farben und den Kontrast Ihres Fernsehers oder Monitors einstellen können. Dieses Progamm ist in Assembler geschrieben und liegt hier als BASIC-Lader vor.

```
5 N=49152
10 READX:IFX=-1THEN30
20 S=S+X:POKEN,X:N=N+1:GOTO 10
30 IF S<>124998 OR N<>50359 THEN PRINT"FEHLER IN DATAS":END
40 SYS49152
```

```
101 DATA 169,0,141,32,208,141,33,208,170,169,5,141,134,2,189,58,19
2,32,210
102 DATA 255,232,201,0,208,245,32,62,241,240,251,201,49,208,3,76,1
54,192,201
103 DATA 50,208,3,76,125,195,201,51,208,3,76,59,194,201,52,208,201
,76,56,193
104 DATA 147,13,32,32,32,32,32,32,32,67,72,69,67,75,80,82,79,71,82
,65,77,77
105 DATA 32,13,13,13,13,32,32,18,32,49,32,146,32,84,69,83,84,66,73
,76,68,13
106 DATA 13,32,32,18,32,50,32,146,32,83,79,85,78,68,13,13,32,32,18
,32,51,32
107 DATA 146,32,82,65,77,84,69,83,84,13,13,32,32,18,32,52,32,146,3
2,74,79
108 DATA 89,83,84,73,67,75,13,13,0,169,147,32,210,255,160,28,162,0
,189,223
109 DATA 192,32,210,255,232,201,0,208,245,136,208,240,162,0,160,40
,189,255
110 DATA 192,32,210,255,136,208,250,232,224,20,208,240,162,0,189,2
1,193,240
111 DATA 7,32,210,255,232,76,197,192,162,18,160,12,24,32,10,229,32
,86,195
112 DATA 76,0,192,18,154,32,32,5,32,32,28,32,32,159,32,32,156,32,3
2,30,32
113 DATA 32,31,32,32,158,32,32,152,32,32,153,32,32,0,18,154,32,5,3
2,28,32
114 DATA 159,32,156,32,30,32,31,32,158,32,152,32,153,32,0,19,5,18,
29,29,29
115 DATA 29,29,29,29,29,29,17,84,69,83,84,66,73,76,68,32,68,69,83,
32,68,65
116 DATA 84,65,83,65,84,13,0,162,0,189,211,193,32,210,255,232,201,
0,208,245
117 DATA 162,13,160,16,24,32,240,255,162,0,189,44,194,32,210,255,2
32,201,0
118 DATA 208,245,162,13,160,16,24,32,240,255,162,0,173,0,220,41,1,
208,13,189
119 DATA 9,194,32,210,255,232,201,0,208,245,240,205,173,0,220,41,2
,208,13
120 DATA 189,16,194,32,210,255,232,201,0,208,245,240,185,173,0,220
,41,4,208
```

```
121 DATA 13,189,23,194,32,210,255,232,201,0,208,245,240,165,173,0,
220,41,8
122 DATA 208,13,189,30,194,32,210,255,232,201,0,208,245,240,25,173
,0,220,41
123 DATA 16,208,13,189,37,194,32,210,255,232,201,0,208,245,240,5,3
2,62,241
124 DATA 208,3,76,69,193,76,0,192,147,13,83,84,69,67,75,69,78,32,8
3,73,69
125 DATA 32,68,69,78,32,74,79,89,83,84,73,67,75,32,73,78,32,80,79,
82,84,32
126 DATA 35,50,13,17,18,69,78,68,69,146,32,61,32,84,65,83,84,69,0,
79,66,69
127 DATA 78,32,32,0,85,78,84,69,78,32,0,76,73,78,75,83,32,0,82,69,
67,72,84
128 DATA 83,0,70,69,85,69,82,32,0,32,32,32,32,32,32,32,157,157,157
,157,157
129 DATA 157,157,0,120,162,0,189,7,195,32,210,255,232,201,0,208,24
5,162,0
130 DATA 181,0,141,144,4,160,0,200,208,253,213,0,240,3,76,235,194,
232,208
131 DATA 236,162,0,189,53,195,32,210,255,232,201,0,208,245,162,0,1
89,64,195
132 DATA 32,210,255,232,201,0,208,245,162,0,189,0,1,160,0,200,208,
253,141
133 DATA 144,4,157,0,1,221,0,1,208,93,232,208,234,162,0,189,53,195
,32,210
134 DATA 255,232,201,0,208,245,162,0,189,75,195,32,210,255,232,201
,0,208,245
135 DATA 120,169,48,133,1,162,0,160,4,134,251,132,252,162,0,160,25
1,161,251
136 DATA 129,251,141,144,4,193,251,208,36,232,208,242,230,252,136,
208,237
137 DATA 169,55,133,1,162,0,189,53,195,32,210,255,232,201,0,208,24
5,169,13
138 DATA 32,210,255,32,86,195,76,0,192,169,55,133,1,162,0,189,40,1
95,32,210
139 DATA 255,232,201,0,208,245,169,13,32,210,255,32,86,195,76,0,19
2,147,13
140 DATA 32,32,32,32,32,32,32,32,82,65,77,84,69,83,84,17,17,17,17,
17,13,13
```

```
141 DATA 90,69,82,79,80,65,71,69,0,32,32,32,32,32,32,70,69,72,76,6
9,82,0,32
142 DATA 32,32,32,32,32,32,32,79,75,0,13,13,83,84,65,80,69,76,32,3
2,0,13,13
143 DATA 54,52,75,32,82,65,77,32,0,162,0,189,105,195,32,210,255,23
2,201,0
144 DATA 208,245,32,62,241,240,251,96,18,17,17,17,84,65,83,84,69,3
2,68,82
145 DATA 85,69,67,75,69,78,146,0,32,68,229,162,0,134,124,169,8,133
,125,32
146 DATA 49,196,166,124,169,15,141,24,212,169,1,157,0,212,133,126,
169,22,157
147 DATA 1,212,169,34,157,5,212,169,133,157,6,212,169,15,157,3,212
,157,2,212
148 DATA 6,125,32,2,196,165,125,9,1,166,124,157,4,212,32,232,195,1
65,125,157
149 DATA 4,212,32,232,195,32,243,195,176,231,36,125,48,3,76,176,19
5,169,0
150 DATA 157,4,212,165,124,24,105,7,133,124,201,21,144,159,76,0,19
2,160,0
151 DATA 162,5,136,208,253,202,208,250,96,166,124,230,126,165,126,
240,5,157
152 DATA 1,212,56,96,24,96,32,87,196,169,32,36,125,112,14,48,17,20
8,5,162
153 DATA 0,76,32,196,162,14,76,32,196,162,30,76,32,196,162,44,189,
105,196
154 DATA 240,7,32,210,255,232,76,32,196,169,13,76,210,255,32,68,22
9,32,97
155 DATA 196,165,124,240,10,201,7,240,3,162,59,44,162,56,44,162,53
,32,32,196
156 DATA 162,62,32,32,196,169,13,32,210,255,76,210,255,56,32,240,2
55,160,7
157 DATA 24,76,240,255,162,5,160,7,24,76,240,255,68,82,69,73,69,67
,75,45,87
158 DATA 69,76,76,69,0,83,69,65,71,69,90,65,72,78,45,87,69,76,76,6
9,0,82,69
159 DATA 67,72,69,67,75,45,87,69,76,76,69,0,82,65,85,83,67,72,69,7
8,0,49,46
160 DATA 0,50,46,0,51,46,0,32,32,84,79,78,71,69,78,69,82,65,84,79,
82,0,9,-1
```

8. Vergleich der Rechner

Der neue C64 II hat sich optisch zu seinem Vorläufermodell recht stark verändert. Das neue Gehäuse entspricht nicht mehr dem alten etwas klobig wirkenden "Buckelgehäuse".

Die Designer haben sich bei ihrer Arbeit eher am Commodore 128 orientiert, wohl in der Hoffnung, wenigstens optisch einen PC zu schaffen. Eine Bestätigung dafür findet sich auch in der neuen Aufschrift, die der Rechnerjetzt trägt.

Aus dem simplen 'Commodore 64' ist jetzt ein respektschaffender 'Commodore 64 PERSONAL COMPUTER' geworden.

Insgesamt ist das Gerät flacher geworden und die Tastatur etwas tiefer und schräger angebracht als vorher. Dadurch wird das Eintippen von Programmen oder Texten erheblich erleichtert. Außerdem sind die Sonderzeichen, die vorher unter den Buchstaben gedruckt waren, neben die Buchstaben gesetzt worden, was jedoch kein Nachteil ist.

Die Mechanik der Tastatur ist identisch mit der alten, nur die Farbe der Tasten hat sich geändert. Auch die einzelnen Anschlüsse des Rechners sind gleich geblieben. Nicht einmal der so dringend benötigte RESET-Schalter hat an dem neuen Rechner einen Platz gefunden.

Durch das veränderte Gehäuse steht im Computer kein Platz zur Verfügung, um zum Beispiel ein zweites Betriebsystem einzubauen. Schraubt man das Gehäuse auf, so ist die Platine noch gar nicht sichtbar, weil diese unter der Tastatur und einer dicken Isolation aus Blech verborgen ist.

Um an die Platine heranzukommen, müssen also erst sämtliche Schrauben und Stecker, die mit der Platine verbunden sind, gelöst werden, um das Isolationsblech abnehmen zu können. Erst jetzt wird einem ein Blick auf die stark veränderte Platine gewährt, die ca. 40 Prozent kleiner ist als die alte. Die Platine ist von der Größe her ungefähr mit einer Steckkarte für den IBM-

AT zu vergleichen. In ihr stecken weitaus weniger Bausteine und elektronische Teile als in der alten Platine. Sie beinhaltet nämlich nur noch 17 Cips, wogegen die alte Platine noch ca. 30 hatte.

Die CIAs 6526 sind gleich geblieben, sie liegen nur an unterschiedlichen Stellen, eine davon links außen und die andere direkt neben der Tastatur, woraus sich schießen läßt, daß die linke zum Steuern des USER-Ports und der RS232 und die andere zur Abfrage von Tastatur und Control-Port dient.

An den RAM-Bausteinen wurde ebenfalls gespart. Wo vorher acht 8-KByte-RAM-Bausteine waren, befinden sich nur noch zwei 32-KByte-RAM-Chips des Types 41464.

Die Taktbausteine des Types 8701 beinhalten quasi alle vier Bausteine der alten Platine, die vorher in dem abgeschirmten Teil neben dem VIC lagen.

Der Modulator ist erheblich kleiner geworden, wobei er nichts von seiner Leistungsfähigkeit eingebüßt hat.

Der VIC ist nicht mehr der alte 6569, der gesondert abgeschirmt in einem Blechgehäuse auf der Platine lag, sondern er ist durch einen neuen mit der Typenbezeichnung 8565 ausgetauscht worden, der im Gegensatz zum alten nur noch mit 5V auskommt. Er liegt außerdem wie die anderen Chips frei auf der Platine, was aber keinen Unterschied macht, weil die Platine auch so ausreichend abgeschirmt ist.

Der neue Prozessor mit der Typenbezeichnung 8500 ist mit dem alten 6510 vollkommen kompatibel, so daß auf jeden Fall alle Programme auch mit dem neuen Baustein laufen.

Das Zeichensatz-ROM ist in beiden Fällen identisch und befindet sich sogar noch an der gleichen Stelle wie vorher.

Leider hat man bei dem neuen Rechner das BASIC- und KERNAL-ROM in einem Baustein zusammengefaßt, was den Nachteil hat, daß man nicht so einfach eine Betriebs-

systemumschaltplatine einbauen kann, da man jetzt immer das BASIC mitbrennen muß. Außerdem ist dieses Vorhaben sowieso wegen des Platzmangels stark eingeschränkt.

Das Farb-RAM ist zwar bei dem neuen C64 II an anderer Stelle, ist aber immer noch das alte.

Der neue SID mit der Typenbezeichnung 8580 ist mit dem alten 6581 vollkommen kompatibel. Er liegt lediglich an einer anderen Stelle.

Die Multiplexer, die zur Adressierung der RAM-Bausteine nötig sind, sind bei der neuen Version alle in einem Chip vom Typ 251715 untergebracht. Er übernimmt somit alle Funktionen der Speicherverwaltung.

Trotz aller Änderungen läßt sich mit Bestimmtheit sagen, daß der neue C64 II 100% kompatibel mit seinem Vorläufermodell ist, denn alle Programme oder Module arbeiten beim 'neuen' genausogut wie beim guten alten C64.

Anhang A

Der folgende BASIC-Loader entspricht dem bereits im Kapitel 2.1 beschriebenen Maschinensprachemonitor, der im Speicherbereich $9000 (36864) bis $9D10 (40208) liegt.

```
5 N=9*16^3
10 READ X:IF X=-1 THEN 30
12 POKE N,X
15 S=S+X:N=N+1:GOTO 10
30 IF S<>363070 OR N<>40208 THEN PRINT"FEHLER IN DATAS":END
35 SYS 9*16^3
36 DATA120,169,39,141,22,3,169,152,141,23,3,169,0,133,103,76,139,1
52,162
37 DATA255,160,155,132,99,160,0,240,11,169,168,133,98,32,17,145,32
,141,145
38 DATA96,177,96,232,224,160,240,238,221,171,155,208,246,138,56,24
9,216
39 DATA156,144,3,200,208,247,132,2,152,10,24,101,2,133,98,32,17,14
5,189
40 DATA65,156,208,8,169,35,32,240,145,76,95,145,201,1,208,3,76,113
,145,201
41 DATA2,208,6,32,113,145,76,156,145,201,3,208,6,32,113,145,76,166
,145,201
42 DATA4,208,3,76,95,145,201,5,208,6,32,95,145,76,156,145,201,6,20
8,6,32
43 DATA95,145,76,166,145,201,7,208,16,169,40,32,240,145,32,95,145,
32,156
44 DATA145,169,41,76,240,145,201,8,208,16,169,40,32,240,145,32,95,
145,169
45 DATA41,32,240,145,76,166,145,201,9,208,58,32,141,145,169,36,32,
240,145
46 DATA160,0,177,96,48,28,56,101,96,72,144,10,164,97,200,152,32,58
,145,24
47 DATA144,5,165,97,32,58,145,104,32,58,145,76,141,145,56,101,96,7
2,176
48 DATA238,164,97,136,152,32,58,145,24,144,233,201,10,208,13,169,4
0,32,240
```

```
49 DATA145,32,113,145,169,41,76,240,145,76,141,145,32,176,145,169,
58,32
50 DATA240,145,76,48,145,32,176,145,169,44,32,240,145,32,48,145,32
,47,146
51 DATA160,0,177,98,32,240,145,200,192,3,208,246,169,32,76,240,145
,165,97
52 DATA32,58,145,165,96,76,58,145,32,70,145,72,152,32,240,145,104,
76,240
53 DATA145,72,74,74,74,74,32,83,145,168,104,76,83,145,41,15,201,10
,24,48
54 DATA2,105,7,105,48,96,169,36,32,240,145,32,141,145,160,0,177,96
,32,58
55 DATA145,76,141,145,169,36,32,240,145,160,2,177,96,32,58,145,160
,1,177
56 DATA96,32,58,145,32,141,145,32,141,145,76,141,145,230,96,208,10
,230,97
57 DATA208,6,72,169,255,133,102,104,96,169,44,32,240,145,169,88,76
,240,145
58 DATA169,44,32,240,145,169,89,76,240,145,32,42,146,169,46,76,240
,145,32
59 DATA200,145,133,2,152,32,200,145,10,10,10,10,5,2,96,201,65,56,4
8,2,233
60 DATA7,233,48,96,32,217,145,170,76,217,145,32,231,145,201,32,240
,249,168
61 DATA32,231,145,76,184,145,32,52,146,240,1,96,76,64,146,32,11,14
6,32,210
62 DATA255,76,23,146,32,11,146,32,207,255,76,23,146,32,11,146,32,6
2,241
63 DATA76,23,146,72,165,1,133,251,169,55,133,1,104,88,96,72,120,16
5,251
64 DATA133,1,104,96,32,2,146,201,3,240,1,96,76,64,146,169,13,76,24
0,145
65 DATA169,32,76,240,145,32,249,145,201,13,96,32,231,145,201,32,96
,32,176
66 DATA145,162,255,154,32,52,146,240,245,201,32,240,244,201,46,240
,240,160
67 DATA19,136,48,14,217,138,154,208,248,185,100,154,72,185,119,154
,72,96
68 DATA169,63,32,240,145,76,64,146,32,158,146,32,31,146,32,127,146
,144,199
```

```
69 DATA32,18,144,76,113,146,165,102,240,7,169,0,133,102,76,154,146
,165,97
70 DATA197,101,144,12,208,8,165,96,197,100,144,4,240,2,24,96,56,96
,32,210
71 DATA145,134,97,133,96,32,52,146,240,16,201,32,208,184,32,220,14
5,170
72 DATA32,217,145,134,101,133,100,96,165,96,133,100,165,97,133,101
,96,32
73 DATA210,145,134,97,133,96,32,52,146,208,151,88,108,96,0,120,108
,2,160
74 DATA32,158,146,32,31,146,32,127,146,144,6,32,235,146,76,218,146
,76,64
75 DATA146,32,6,145,160,0,132,3,177,96,72,32,47,146,104,32,58,145,
164,3
76 DATA200,192,8,208,237,162,8,160,0,32,47,146,169,1,133,199,177,9
6,41,127
77 DATA201,32,144,10,32,240,145,169,0,133,212,24,144,3,32,179,145,
32,141
78 DATA145,202,208,229,169,0,133,199,96,32,210,145,134,97,133,96,1
60,0,32
79 DATA231,145,132,3,32,231,145,32,135,147,168,32,231,145,32,135,1
47,32
80 DATA184,145,164,3,145,96,200,192,8,208,226,169,145,32,240,145,3
2,235
81 DATA146,32,176,145,169,58,141,119,2,165,97,32,70,145,140,120,2,
141,121
82 DATA2,165,96,32,70,145,140,122,2,141,123,2,169,32,141,124,2,169
,6,133
83 DATA198,76,67,146,201,32,240,1,96,76,102,146,32,210,145,134,97,
133,96
84 DATA169,155,133,99,32,58,146,240,251,32,62,148,224,31,144,11,22
4,39,176
85 DATA7,169,0,141,1,2,240,5,169,1,141,1,2,32,117,148,133,3,169,0,
224,0
86 DATA240,12,202,24,125,216,156,76,189,147,224,0,208,240,164,4,19
0,216
87 DATA156,168,165,3,217,65,156,208,10,185,171,155,160,0,145,96,24
,144,7
88 DATA200,202,208,237,76,102,146,200,162,6,165,3,202,48,25,221,15
7,154
```

```
89 DATA208,248,165,98,145,96,169,145,32,240,145,32,18,144,32,176,1
45,169
90 DATA44,76,98,147,201,11,240,236,201,9,208,30,56,165,99,233,2,13
3,99,165
91 DATA98,229,97,201,2,144,4,201,255,208,75,32,114,149,24,101,99,1
45,96
92 DATA76,252,147,165,99,145,96,200,165,98,145,96,76,252,147,162,0
,157,224
93 DATA158,232,224,3,240,6,32,231,145,24,144,242,162,0,134,98,160,
255,200
94 DATA185,224,158,209,98,240,15,232,230,98,230,98,230,98,164,98,1
92,168
95 DATA176,9,144,231,192,2,208,229,134,4,96,76,102,146,32,52,146,2
08,3,169
96 DATA11,96,201,32,240,244,201,35,208,19,32,231,145,32,73,149,32,
217,145
97 DATA133,98,32,52,146,208,221,169,0,96,201,40,208,54,32,231,145,
32,73
98 DATA149,32,217,145,133,98,32,231,145,201,41,240,25,201,44,240,3
,76,53
99 DATA149,32,87,149,32,231,145,201,41,208,179,32,52,146,208,174,1
69,7,96
100 DATA32,95,149,32,52,146,208,163,169,8,96,32,73,149,32,217,145,
133,98
101 DATA32,52,146,240,83,201,44,240,52,32,224,145,133,99,32,52,146
,240,31
102 DATA201,44,208,129,32,231,145,201,88,240,12,201,89,208,193,32,
52,146
103 DATA208,188,169,3,96,32,52,146,208,180,169,2,96,173,1,2,240,3,
169,1,96
104 DATA169,9,96,32,231,145,201,88,240,12,201,89,208,47,32,52,146,
208,42
105 DATA169,6,96,32,52,146,208,34,169,5,96,169,4,96,32,224,145,133
,99,32
106 DATA231,145,201,41,208,16,32,52,146,208,11,169,10,96,201,36,24
0,3,76
107 DATA102,146,96,76,102,146,32,106,149,32,231,145,201,88,208,243
,96,32
108 DATA106,149,32,231,145,201,89,208,232,96,32,231,145,201,44,208
,224,96
```

```
109 DATA165,96,73,255,170,232,138,96,32,231,149,32,52,146,208,7,16
9,8,133
110 DATA186,76,174,149,201,44,208,12,32,217,145,133,186,32,52,146,
240,23
111 DATA201,44,208,119,32,210,145,133,98,134,99,32,52,146,208,242,
169,0,133
112 DATA185,24,144,4,162,1,134,185,165,186,201,8,240,10,201,9,240,
3,76,215
113 DATA149,32,21,150,169,0,32,11,146,166,98,164,99,32,213,255,32,
105,150
114 DATA32,23,146,76,141,153,201,1,208,55,76,194,149,169,1,133,185
,133,186
115 DATA76,194,149,32,52,146,240,242,201,32,240,247,201,34,208,30,
162,0,32
116 DATA249,145,201,34,240,10,157,224,158,232,224,17,240,13,208,23
9,134,183
117 DATA169,158,133,188,169,224,133,187,96,76,102,146,169,165,162,
244,32
118 DATA69,150,162,3,189,163,154,157,84,159,202,16,247,162,15,189,
167,154
119 DATA157,105,159,202,16,247,162,2,189,183,154,157,142,159,202,1
6,247,169
120 DATA3,141,138,159,198,186,96,32,11,146,133,100,134,101,160,0,1
77,100
121 DATA153,0,159,200,208,248,169,0,141,48,3,141,50,3,169,159,141,
49,3,141
122 DATA51,3,76,23,146,162,3,189,76,253,157,48,3,202,16,247,96,32,
231,149
123 DATA32,106,149,32,217,145,133,186,32,106,149,32,210,145,133,96
,134,97
124 DATA32,106,149,32,210,145,133,98,134,99,32,52,146,208,53,165,1
86,201
125 DATA8,240,7,201,9,208,32,32,209,150,169,1,133,185,169,96,166,9
8,164,99
126 DATA232,208,1,200,32,11,146,32,216,255,32,23,146,32,105,150,76
,141,153
127 DATA72,32,42,146,104,201,1,208,2,240,216,76,102,146,169,237,16
2,245,32
128 DATA69,150,162,2,189,186,154,157,52,159,202,16,247,162,4,189,1
89,154
```

```
129 DATA157,60,159,202,16,247,198,186,96,32,23,146,177,172,32,11,1
46,76,221
130 DATA237,32,158,146,162,0,32,52,146,240,200,201,32,208,196,134,
3,32,217
131 DATA145,166,3,157,0,159,232,32,52,146,208,236,134,3,162,0,32,1
27,146
132 DATA144,17,160,0,189,0,159,145,96,32,141,145,232,228,3,208,236
,240,232
133 DATA76,64,146,32,158,146,32,210,145,133,98,134,99,228,97,144,6
,208,27
134 DATA197,96,176,23,32,127,146,144,228,160,0,177,96,145,98,32,14
1,145,230
135 DATA98,208,238,230,99,76,75,151,165,100,56,229,96,72,165,101,2
29,97,144
136 DATA61,24,101,99,133,99,104,24,101,98,144,2,230,99,133,98,32,1
27,146
137 DATA144,204,160,0,177,100,145,98,198,100,165,100,201,255,208,2
,198,101
138 DATA198,98,165,98,201,255,208,227,198,99,76,125,151,32,158,146
,32,52
139 DATA146,240,125,162,0,201,32,208,119,32,247,151,144,16,134,3,3
2,224,145
140 DATA166,3,157,0,159,232,32,52,146,208,231,32,42,146,134,3,160,
0,32,127
141 DATA146,144,38,32,31,146,185,0,2,240,13,177,96,217,0,159,240,6
,32,141
142 DATA145,76,199,151,200,196,3,208,225,32,48,145,32,141,145,32,4
7,146,76
143 DATA199,151,76,64,146,32,231,145,201,63,208,29,32,249,145,201,
63,208
144 DATA31,169,0,157,0,2,232,32,52,146,240,9,201,32,208,16,104,104
,76,173
145 DATA151,24,96,72,169,255,157,0,2,104,56,96,76,102,146,120,216,
104,141
146 DATA69,2,104,141,68,2,104,141,67,2,104,141,66,2,104,141,64,2,1
04,141
147 DATA65,2,206,64,2,208,3,206,65,2,186,142,70,2,162,0,189,194,15
4,32,240
148 DATA145,232,224,61,208,245,173,65,2,32,58,145,173,64,2,32,58,1
45,32,47
```

```
149 DATA146,173,21,3,32,58,145,173,20,3,32,58,145,162,0,32,47,146,
189,66
150 DATA2,32,58,145,232,224,5,208,242,32,42,146,76,64,146,141,67,2
,142,68
151 DATA2,140,69,2,76,53,152,32,158,146,32,210,145,133,98,134,99,3
2,42,146
152 DATA32,127,146,144,29,160,0,177,96,209,98,208,12,32,141,145,23
0,98,208
153 DATA236,230,99,76,164,152,32,48,145,32,47,146,76,177,152,76,64
,146,32
154 DATA210,145,168,32,42,146,138,72,152,170,104,32,11,146,32,205,
189,32
155 DATA23,146,76,64,146,160,0,132,96,132,97,32,249,145,41,15,24,1
01,96,133
156 DATA96,144,2,230,97,32,249,145,201,48,144,26,72,165,96,164,97,
10,38,97
157 DATA10,38,97,101,96,133,96,152,101,97,6,96,42,133,97,104,144,2
12,32,42
158 DATA146,32,48,145,76,64,146,32,158,146,165,96,24,101,100,133,9
6,165,97
159 DATA101,101,133,97,32,42,146,32,48,145,76,64,146,32,158,146,16
5,96,56
160 DATA229,100,133,96,165,97,229,101,133,97,76,47,153,32,11,146,3
2,231,255
161 DATA32,23,146,32,52,146,240,52,201,32,240,247,201,36,240,86,16
2,0,157
162 DATA224,158,232,224,17,240,73,32,52,146,208,243,138,162,224,16
0,158,32
163 DATA11,146,32,189,255,169,1,162,8,160,15,32,186,255,32,192,255
,32,231
164 DATA255,32,23,146,32,42,146,32,11,146,169,8,133,186,32,180,255
,169,111
165 DATA133,185,32,150,255,32,165,255,32,210,255,201,13,208,246,32
,171,255
166 DATA32,23,146,76,64,146,76,102,146,72,32,42,146,104,32,11,146,
141,224
167 DATA158,162,224,160,158,169,1,32,189,255,162,8,160,96,32,186,2
55,32,213
168 DATA243,165,186,32,180,255,165,185,32,150,255,169,0,133,144,16
0,3,140
```

```
169 DATA224,158,32,165,255,141,225,158,164,144,208,49,32,165,255,1
64,144
170 DATA208,42,172,224,158,136,208,230,174,225,158,32,205,189,169,
32,32,210
171 DATA255,32,165,255,166,144,208,18,170,240,6,32,210,255,76,10,1
54,169
172 DATA13,32,210,255,160,2,208,194,32,66,246,32,23,146,76,64,146,
32,52,146
173 DATA208,48,32,42,146,32,11,146,165,103,208,27,198,103,169,1,16
2,4,32
174 DATA186,255,169,0,133,183,32,192,255,162,1,32,201,255,32,23,14
6,76,64
175 DATA146,230,103,32,231,255,32,204,255,76,80,154,76,102,146,146
,146,146
176 DATA146,147,147,149,150,150,151,151,152,152,152,152,153,153,15
3,154,109
177 DATA214,194,210,45,142,121,116,251,54,158,76,150,200,224,30,55
,74,43
178 DATA68,77,71,88,58,44,76,83,70,84,72,82,67,36,35,43,45,64,80,0
,4,5,6
179 DATA7,8,234,234,24,144,120,72,169,48,133,1,104,145,174,169,55,
133,1,234
180 DATA234,234,76,169,245,32,221,237,32,241,150,234,234,13,13,73,
78,84,69
181 DATA82,78,32,77,79,78,73,84,79,82,32,66,89,32,82,46,32,71,69,7
6,70,65
182 DATA78,68,13,13,32,32,32,80,82,32,32,73,82,81,32,32,83,82,32,6
5,67,32
183 DATA88,82,32,89,82,32,83,80,13,46,42,0,76,68,65,76,68,88,76,68
,89,83
184 DATA84,65,83,84,88,83,84,89,84,65,88,84,65,89,84,88,65,84,89,6
5,84,88
185 DATA83,84,83,88,80,76,65,80,72,65,80,76,80,80,72,80,65,68,67,8
3,66,67
186 DATA73,78,67,68,69,67,73,78,88,68,69,88,73,78,89,68,69,89,65,7
8,68,79
187 DATA82,65,69,79,82,67,77,80,67,80,88,67,80,89,66,73,84,66,67,6
7,66,67
188 DATA83,66,69,81,66,78,69,66,77,73,66,80,76,66,86,67,66,86,83,7
4,77,80
```

```
189 DATA74,83,82,65,83,76,76,83,82,82,79,76,82,79,82,67,76,67,67,7
6,68,67
190 DATA76,73,67,76,86,83,69,67,83,69,68,83,69,73,78,79,80,82,84,8
3,82,84
191 DATA73,66,82,75,63,63,63,169,173,189,185,165,181,161,177,162,1
74,190
192 DATA166,182,160,172,188,164,180,141,157,153,133,149,129,145,14
2,134,150
193 DATA140,132,148,170,168,138,152,154,186,104,72,40,8,105,109,12
5,121,101
194 DATA117,97,113,233,237,253,249,229,245,225,241,238,254,230,246
,206,222
195 DATA198,214,232,202,200,136,41,45,61,57,37,53,33,49,9,13,29,25
,5,21,1
196 DATA17,73,77,93,89,69,85,65,81,201,205,221,217,197,213,193,209
,224,236
197 DATA228,192,204,196,44,36,144,176,240,208,48,16,80,112,76,108,
32,14,30
198 DATA6,22,10,78,94,70,86,74,46,62,38,54,42,110,126,102,118,106,
24,216
199 DATA88,184,56,248,120,234,96,64,0,1,2,3,4,5,7,8,0,1,3,4,6,0,1,
2,4,5,1
200 DATA2,3,4,5,7,8,1,4,6,1,4,5,11,11,11,11,11,11,11,11,11,11,0,1,
2,3,4,5
201 DATA7,8,0,1,2,3,4,5,7,8,1,2,4,5,1,2,4,5,11,11,11,11,0,1,2,3,4,
5,7,8,0
202 DATA1,2,3,4,5,7,8,0,1,2,3,4,5,7,8,0,1,2,3,4,5,7,8,0,1,4,0,1,4,
1,4,9,9
203 DATA9,9,9,9,9,9,1,10,1,1,2,4,5,11,1,2,4,5,11,1,2,4,5,11,1,2,4,
5,11,11
204 DATA11,11,11,11,11,11,11,11,11,11,8,5,5,7,3,3,1,1,1,1,1,1,1,1,
1,1,8,8
205 DATA4,4,1,1,1,1,8,8,8,8,3,3,2,1,1,1,1,1,1,1,1,2,1,5,5,5,5,1,1,
1,1,1,1
206 DATA1,1,1,1,1,-1
```

Anhang B

Glossar

Assembler:

Programmiersprache, die es ermöglicht, die CPU direkt anzusprechen. Sie ist sehr unkomfortabe, dafür aber die schnellste Sprache, da der Computer sie direkt verarbeiten kann (Maschinensprache).

Betriebssystem:

Das Betriebssystem ist im ROM fest abgelegt und ein Programm, das die wichtigsten Computerabläufe steuert, die für jede Programmiersprache gebraucht werden. Unter anderem sind dies die Aufgaben der Ein-/Ausgabesteuerung.

Bidirektional:

Bidirektional heißt, daß eine Operation in zwei (beide) Richtungen ausgeführt werden kann.

Expansionsport (Modul-Steckplatz):

Der Expansionsport ist der Erweiterungssteckplatz des Computers. An ihm können Zusatzmodule eingesteckt werden. Diese können BASIC-Erweiterungen, andere Programmiersprachen oder ein Steckmodul darstellen. Grundsätzlich ist dieser Port durch ein BASIC-Programm nicht zu bearbeiten.

Handshake:

Über eine Handshakeleitung verständigen sich Sender und Empfänger über die Bereitschaft, neue Daten zu senden beziehungsweise zu empfangen.

Latch:

Kommt bei einer Zweierteilung einer Speicherzelle vor. Der Latch (Vorspeicher) ist der Teil dieser Speicherzelle, der nur beschrieben werden kann. Bei einem Lesezugiff erhällt man jedoch immer den anderen Teil der Speicherzelle, der ebenfalls ein Acht-Bit-Wert ist.

Maus:

Eingabegerät, das am Joystickport angeschlossen wird. Bewegt man es auf einer Unterlage, werden ensprechend der Bewegung Objekte auf dem Bildschirm bewegt.

Mantisse:

Als Mantisse bezeichnet man bei der Exponentialschreibweise den Wert, mit dem die Potenz multipliziert wird. Die Mantisse liegt laut Definition zwischen 1 und 10. Bei der Zahl 1986, in Exponentialschreibweise 1.986*10^3, ist 1.986 die Mantisse.

NMI:

Nicht maskierbarer Interrupt. Eine Unterbrechung des Programmablaufs, die durch das Setzen (auf Masse legen) der NMI-Leitung ausgelöst wird. Dieser Interrupt ist vom Programmierer nicht beeinflußbar (maskierbar).

Paddle:

Eingabegerät, das am Joystickport angeschlossen wird. Es besteht aus einem Potentiometer, dessen Widerstand durch einen Drehknopf beeinflußt wird. Dieser Widerstand ist als Zahlwert in Port X und Y des SID anzulesen.

Parallel:

Form der Datenübertragung, bei der die Bits nebeneinander (parallel) und somit gleichzeitig übertragen werden. Dies ermöglicht eine hohe Datenübertragungsgeschwindigkeit.

Parität:

Anzahl der '1' einer Bitgruppe, die übertragen wird. Von einer geraden Parität wird gesprochen, wenn die Anzahl der '1' gerade ist. Dementsprechend bedeutet eine ungerade Parität die ungerade Anzahl von '1'.

Paritätsbit:

Bit, das bei der Datenübertragung zu der Bitgruppe mitgesandt wird. Ist die Parität gerade, so wird eine Eins als Bit mitgesendet. So kann der Empfänger überprüfen, ob bei der Übertragung ein Fehler aufgetreten ist.

Peripherie:

Darunter versteht man alle Geräte, die an einem Computer angeschlossen werden können. Solche Geräte sind Drucker, Floppy, Joystick, Lightpen, Monitor und so weiter.

Sektor:

Jede Spur auf Diskette ist in Sektoren aufgeteilt. Die Anzahl der Sektoren ist abhängig von der Lage der Spur. Innere Spuren haben weniger Sektoren als äußere. Die Länge eines Sektors beträgt 256 Bytes.

Seriell:

Form der Datenübertragung. Bei dieser Art der Datenübertragung werden die Bits nacheinander über eine Leitung geschickt. Der Vorteil ist, daß nur zwei Leitungen benötigt werden. Der Nachteil ist die geringe Geschwindigkeit.

Spur:

Die Diskette wird zu Verwaltungszwecken in 35 Spuren (Tracks) aufgeteilt. Diese Spuren sind ringförmig auf der Diskette angeordnet und von 1 bis 35 duchnumeriert. Außerdem sind die Spuren in verschiedene Sektoren unterteilt.

Stopp-Bit:

Das Stopp-Bit wird bei der seriellen Datenübertragung verwendet, um das Ende eines übertragenen Bytes zu kennzeichnen. Üblicherweise werden 1 oder 2 Stopp-Bits verwendet, was sowohl im Sender als auch im Empfänger festgelegt werden muß.

Systeminterrupt:

Ein Interrupt (Unterbrechung des Hauptprogramms), der von Timer A der CIA 1 je 1/60 Sekunde ausgelöst wird, um unter anderem die Tastatur abzufragen.

Taktzyklus:

Länge eines Systemtaktes. Wird als Zeiteinheit benutzt, um die Ausführungszeit eines Befehls darzustellen.

Trigger:

Ein Trigger ist ein Auslöser. Ist beispielsweise eine Operation timergetriggert, so bedeutet das, daß diese Operation nach dem Ablauf des Timers ausgelöst wird. Der Timer ist hierbei der Auslöser (Trigger).

Zeropage:

Die Zeropage ist eine Besonderheit der 65xx-Microprozessoren. Sie ist im Bereich von 0 bis 255 abgelegt und kann somit mit einem Zweibytebefehl angesprochen werden, was sowohl platz- als auch zeitsparend ist. In ihr sind unter anderem die wichtigsten Zeiger abgelegt.

Anhang C

Stichwortverzeichnis

A/D-Wandler 401
Absolute Adressierung 105
ADRBYT 289, 290
Adressierung 471
ADRFOR 289
Akkumulator 103
Alarmzeit 455
AM 454
Analog 401
AND 14, 243
Arrayberechnung 500
Arrays 22
Arrayvariablen 281
ASCII-Code 18
ASL 247
Atari 441
ATTACK 393
Ausgabepuffer 458
Auto-Start 501

BANDPASS 399
BASIC-Interpreter 20
BASIC-Kompaktor 61
BASIC-Loader 60
BASIC-RAM 92
BASIC-ROM 92
BASIC-String ausgeben 300
BASIC-Zeile 18
BASIN 312, 495
Baud 459
BCC 188
BCD-FORMAT 456
BCS 188
Befehlssatz 363
BEQ 154, 161

Bildschirmcode ... 494
Bildschirmreset ... 494
Bildschirmspeichers ... 92
Binärsystem ... 11
Bit ... 83, 322, 346
Blinkende Bildschirmzeile ... 206
Block ... 345
BMI ... 156, 161
BNE ... 124, 154, 161
BPL ... 156, 161
BRK ... 256
BRK-Befehl ... 110
BSOUT ... 139, 168, 417, 495
Buckelgehäuse ... 811
Byte ... 89

Carry-Flag ... 177
Carryflag ... 413
CHKIN ... 312, 417
CHKKOM ... 283
CHKOUT ... 312
CIAs ... 411
CKUOT ... 417
CLC ... 176
CLD ... 323
CLI ... 256
CLOSE ... 417
CLRCH ... 312
Commodore ... 501
CONTINUOUS ... 453
Controller ... 494
COS ... 494
Counter ... 452
CPU ... 88, 102, 331, 337
CTRL ... 501
Cursor ... 30

DATA ... 344
Datenformate ... 56
Datenregister ... 432

Datenrichtungsregister ... 432
Datentransfer ... 473
DDR ... 432
DEC ... 124
DECAY ... 393
Dekodiertabelle ... 502
Dekrementierbefehle ... 122, 153
DEX ... 124
DEY ... 124
Dezimalsystem ... 11, 82
Digitalwandler ... 401
Direktmodus ... 18
Division ... 50
Drop out ... 418
Dualzahl ... 83

E/A-Port ... 431
Echtzeituhr ... 454
Eingabewarteschleife ... 492
EOI ... 467
EOR ... 15, 243
EXKLUSIV-ODER-Verknüpfung ... 243
Exponent ... 51

FACs ... 53
Farbregister ... 347
Farb-RAM ... 338
Fehlermeldungen ... 235, 492
Feinscrolling ... 372
Feld ... 24
Filter ... 398, 405
Fließkommaarithmetik ... 50
Fließkommakonstante ... 56
Fließkommavariablen ... 280
FORCE LOAD ... 453
Frequenz ... 389
FRETOP ... 310
FRMEVL ... 499
FRMNUM ... 289, 500
FX80 ... 361

Geräteadresse 416, 426
GETBYT 283
GETIN 140
GETPOS 296
Grafik 345
Grafikhilfsprogramm 354, 356
Grafikmodus 333

Halbtonschritte 390
Handshake 459, 460
Header 421
Hertz 455
HEX-Zahl 414
Hexadezimalsystem 11, 86, 389
HIGH-Nibbel 393
Highbyte 366
HIGHPASS 399
Hochkomma 502
Hüllkurve 393, 400

IEC-BUS 465
INC 124
Indexregister 103
Indizierte Adressierung 119
Inkrementierbefehle 122
INT 97
INTEGER 503
Integervariablen 279
Interpreter 279
Interpretercode 19
Interruptvektor 365
Interrupt 337
Interruptprogrammierung 256, 365
Interruptroutine 369
INTOUT 320
Invertieren 223, 357
INX 124
INY 124
IRQ 257

Joystick 441

Kassette 418
Kassettenpuffer 60
Kernal 92
KEY-Bit 396
Kollision 348
Kompatibel 813
Komprimiert 62

Latch 368
LDA 79, 104, 114
LDX 114
LDY 114
LISTEN 469
Listing 344
Logische Filenummer 416
Logische Operationen 239
Logische Verknüpfungen 14
Low 366
LOW-Nibbel 394
LOWPASS 399
LSR 250

Mantisse 52
Maschinensprache 50
Matrix 432
MAUS 440
Monitor 94
Multicolormodus 349
Multicolorsprite 349
Multiplikationstabelle 359

N-Flag 157
Negativ-Flag 157
NMI 257, 462
NOT 15
NTSC-Version 503

ODER-Verknüpfung 242
ONE-SHOT 453
OPEN 493
ORA 242
Oszillator 400
Overlay 428

Paddle 438
PAL-Version 503
Parität 459
PEEK 97
Peripheriegeräte 506
PERSONAL COMPUTER 811
PHA 210
PHP 212
PLA 210
Platine 811
PLOT 142
PM 454
Polynom 501
Portleitung 474
POTX 402
POTY 402
PR 432
Prozessor-Status 103
Prozessor 365
Prüfsumme 420
Pullen 211
Pulsbreite 391
Push processor 212

Quarz 390

Rahmen zeichnen 146
Rahmenfarbe 369
RAM 93
Rasterzeileninterrupt 370
Rauschgenerator 405
REAL 503
Rechnen 174

Rechteckwelle ... 391
Register ... 346
Release ... 393
REM ... 19
RENEW ... 28
RESET ... 60
RESTORE ... 46
ROL ... 251
ROM ... 54, 93
ROM-Listing ... 489
ROR ... 251
Routine ... 489
Routinen des Betriebssystems ... 138
RS-232 ... 458
RTI ... 256
RTS ... 81

Schleifen ... 124
Screen-Scrolling ... 374
Scrollen ... 494
SED ... 323
SEI ... 256, 366
Serielle Schieberegister ... 478
Shift ... 501
SID ... 389
SIN ... 494
Soundcontroller ... 389
Source-Code ... 376
Speicheraufteilung ... 491
Speicherzelle ... 89
Spiegelschrift ... 143
Sprites ... 344
Sprungvektoren ... 43
SQR ... 493
ST ... 470
STA ... 104, 114
Stack ... 91
Stapeloperationen ... 209
Status-Register ... 103, 177
Stelle ... 86

Stellenwert ... 86
STOP+RESTORE ... 96
Stop-Taste ... 501
String ... 37
String-Ausgabe ... 168
Stringarray ausgeben ... 302
Stringstack ... 507
Stringvariablen ... 280
STRRES ... 310
STX ... 114
STY ... 114
SUSTAIN ... 393
SYS ... 30, 82
Systemtakt ... 478, 480, 487
Systeminterrupt ... 454
Taktzyklus ... 453

TALK ... 469
TAN ... 494
Tastaturabfrage ... 269
TAX ... 113
TAY ... 113
TI$ ... 454
Timer A ... 366
TOGGLE ... 453
Token ... 20
Tongenerator ... 389
Trigger ... 453
TSX ... 324
TXA ... 114
TXS ... 324
TYA ... 114

Unmittelbare Adressierung ... 105
Unterlauf ... 452, 453
USR ... 30

Variablen ... 22
Variablenbehandlung ... 279
Vektor ... 365

Vergleich der Rechner 811
VERIFY 493
VIA 431
VIC 370
Video-Reset 494
Videocontroller 494
Virus 59
Virus-Killer 59

Wahrheitstabelle 17
Windowing 225
Word 12

X-Register 103

Zeichen-ROM 92
Zeropage 91, 504
Zeropage-Adressierung 118

Ein Super-Buch zu einem Super-Rechner!

Das hat die C64-Welt noch nicht gesehen: Ein Buch mit allen Informationen rund um den C64 – über 1.000 Seiten Top-Know-how zu einem unschlagbar günstigen Preis. Ob zur Hardware, zu gekaufter Software oder zu selbstgeschriebenen Programmen – im großen Commodore-64-Buch finden Anfänger wie Fortgeschrittene das komplette Wissen, mit dem sie den beliebten Rechner voll und ganz ausnutzen können. Kapitel für Kapitel spannende Lektüre: alles über Standard-Software von der Textverarbeitung bis zu Spielen, BASIC- und Assembler-Programmierung leichtgemacht, das A und O der Datenverwaltung, die phantastische Wunderwelt der Grafik, Sound und Musik, die speziellen Schnittstellen, Floppy-Pflege und -Reparatur, Tips & Tricks zur Datasette, Übersichten, Belegungspläne, Codes und Adressenlisten. Das große Commodore-64-Buch – die ganze Welt Ihres Rechners in einem Band.

Hecht
Das große Commodore-64-Buch
1.142 Seiten, DM 29,80
ISBN 3-89011-370-2

Der perfekte Einsatz Ihrer Floppy-Station.

Mt der Floppy läßt sich weitaus mehr machen als nur laden und starten. Man muß sich lediglich ein wenig auskennen. Was Sie alles aus Ihrer Floppystation herausholen können, zeigt Ihnen das Commodore Floppybuch. Vom Einstieg bis zur Programmierung der Floppy in BASIC. Hier finden Sie alles über den Aufbau von Disketten, zu den einzelnen Dateitypen, zu den Systembefehlen und natürlich auch zu den verschiedenen Fehlermeldungen der unterschiedlichen Floppystationen. Was Sie über Ihre Floppy wissen sollten, finden Sie in diesem Buch – ob Einsteiger oder Profi und egal mit welcher Floppystation Sie arbeiten: der 1541, der II/41 oder der C/70/71/81.

Schönleber
Das Commodore-Floppybuch
240 Seiten, DM 29,-
ISBN 3-89011-269-2

Einfach & schnell: TEXTOMAT PLUS 64.

Von einfachen Blockoperationen bis zum Erstellen von Serienbriefe – z.B. mit DATAMAT, bietet Ihnen TEXTOMAT PLUS 64 alle Vorteile einer schnellen, leistungsfähigen Textverarbeitung. Selbst die Kombination von Text und Grafik beherrscht dieses Programm perfekt. TEXTOMAT PLUS 64 – die Textverarbeitung mit dem Leistungsplus: Komfortables Suchen und Ersetzen, komplette Bausteinverarbeitung, beliebig lange Texte durch Verknüpfung, frei programmierbare Steuerzeichen, Serienbrieferstellung, Floskeltasten, Wordwrap, frei einstellbare Tabulatoren, Rechenfunktionen für alle Grundrechenarten, Mischen von Grafiken und Texten u.v.a.m. TEXTOMAT PLUS 64 – die ideale Textverarbeitung für Ihren 64er. Zu einem unverschämt günstigen Preis.

TEXTOMAT PLUS 64
unverbindliche Preisempfehlung DM 59,-

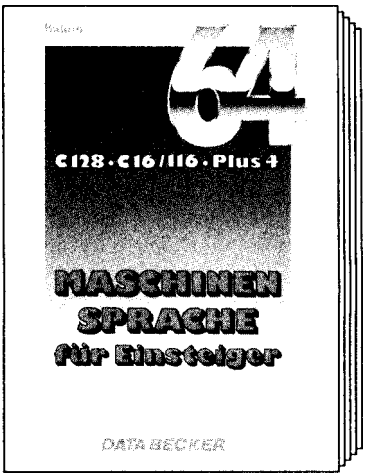
Mit Schwung und Laune Neues lernen.

Anfangen und gleich loslegen – das wünscht sich jeder, der in die Computerei einsteigt. Mit „64 für Einsteiger" geht das ganz problemlos. Vom Anschluß bis zum ersten Programm. Systematisch und leichtverständlich lernen Sie hier Ihren neuen Rechner kennen. Mit zahlreichen Anwendungsbeispielen, Erklärungen zur hochauflösenden Grafik und natürlich mit einer detaillierten Einführung in GEOS 2.0 deutsch. Ein umfangreiches Lexikon macht das Buch auch dann noch wertvoll, wenn Sie längst kein Einsteiger mehr sind. Aufstellen, anschließen, kennenlernen, das erste Programm entwickeln – Sie werden Spaß daran finden, etwas Neues zu lernen. 64 für Einsteiger – und Sie sind bestens für eine eigenständige Arbeit am Rechner vorbereitet.

Szczepanowski
64 für Einsteiger
251 Seiten, DM 29,-
ISBN 3-89011-010-X

Maschinensprache verständlich für jedermann.

Endlich einmal kein unverständliches Lehrbuch, sondern ein Buch, mit dem wirklich jeder, der sich dafür interessiert, schnell und einfach Maschinensprache lernen kann. Ohne das übliche Fachchinesisch wissen Sie schon bald, was ein professionelles Programm ausmacht: BASIC-Routinen heranziehen, Befehle und Strukturen vergleichen und schließlich selbst in Assembler umsetzen – durch dieses Konzept sind Sie sehr schnell in der Lage, die Vorteile dieser Sprache für eigene Anwendungen voll zu nutzen. Eine echte Chance für alle Assembler-Interessierten, denen diese Sprache bisher zu schwierig erschien. „Sprechen" Sie die Sprache der Profis – mit Maschinensprache für Einsteiger. Geeignet für den C64, den C128 und auch für die ganz Kleinen (C16/C116/Plus4).



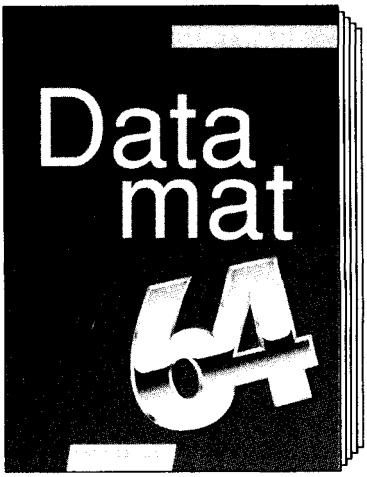
Baloui
Maschinensprache für Einsteiger
345 Seiten, DM 29,-
ISBN 3-89011-182-3

Die besten Tips und Tricks zum Commodore 64.

Jetzt noch mehr Tips & Tricks rund um Ihren C64. Die besten – als Einzeiler, Kurzprogramme oder Peeks und Pokes. Ob zur Datasette, zum Speicher oder zur Floppy. Ob zu BASIC oder zum Softwareschutz. Zu Grafik oder Sound. Oder auch zum Schönsten am C64: den Spielen. Zu allen Bereichen zeigt Ihnen dieses Buch, wie Sie Ihre Arbeit optimieren können. Mit den richtigen Kniffen zur rechten Zeit. Zusammengestellt von einem langjährigen 64er-Experten. Ersparen Sie sich langes Suchen in einschlägigen Zeitschriften, die besten Tips und Tricks für Ihre Arbeit stehen in diesem Buch. Holen Sie alles aus Ihrem C64. Mit den Tips & Tricks eines echten Profis.

Polk
Die besten Tips & Tricks
288 Seiten, DM 29,-
ISBN 3-89011-281-1

Die flexible Dateiverwaltung für Ihren C64.



Jeden Datensatz innerhalb kürzester Zeit suchen, nach beliebigen Feldern selektieren, nach allen Feldern gleichzeitig sortieren, Listen in völlig freiem Format drukken – wie, das alles kann Ihr C64 nicht? Dann probieren Sie es einmal mit DATAMAT 64 – Deutschlands meistverkaufte Dateiverwaltung. Überall, wo Daten und Informationen verwaltet werden, hält Sie diese flexible Dateiverwaltung auf dem neuesten Stand. Dabei besticht das Programm nicht nur durch seine enorme Geschwindigkeit, sondern vor allem auch durch die einfache Bedienerführung. Sogenannte Pulldown-Menüs und eine ausgefeilte Fenstertechnik machen es Ihnen so einfach wie möglich. Sollten sie trotzdem mal Probleme haben, steht Ihnen ein umfangreiches Handbuch mit einigen kleinen Übungsaufgaben zur Seite.

DATAMAT 64
unverbindliche Preisempfehlung DM 59,-

Das neue COMMODORE 64 INTERN Buch

Wer bislang mehrere Bücher brauchte, um effektiv auf dem C64 programmieren zu können, findet hier endlich alles über den C64 in einem Buch. In diesem Band erfahren Sie alles, was zum Thema Hardware, Betriebssystem und Systemprogrammierung auf dem C64 zu sagen ist. Und zwar so verständlich, daß auch die Nichtprofis unter Ihnen die Arbeitsweise des C64-Betriebssystems schnell und sicher verstehen werden. Natürlich ist dieses Werk eine riesige Fundgrube für jeden C64-Benutzer, denn hier zeigen die Autoren, wie man das Betriebssystem richtig nutzt. Zudem vertiefen die vielen prägnanten Beispielprogramme das Gelernte und zeigen, wie man schnelle Programme erstellt. Selbst der erfahrene C64-Anwender erhält eine Fülle von Kniffen, Facts und Details. Zusätzlich finden Sie einen kompletten Einsteigerkurs in die Programmiersprache Assembler, der Ihnen zeigt, wie's geht: ohne Fachchinesisch, dafür einfach, schnell und effektiv.

- **Einführung in Assembler**
  Von Anfang an und ohne Vorkenntnisse
- **Rechnen in Maschinensprache**
  Interne Zahlendarstellung und deren Nutzung
- **Fließkomma-Arithmetik**
- **BASIC-Erweiterungen selbst gemacht**
  Erstellung eigener nützlicher BASIC-Befehle
- **Kompletter Monitor zum Abtippen**
  Zum Einstieg in die Welt der Maschinensprache
- **Virus-Killer**
  Schützt Ihren Rechner vor dem Zugriff bösartiger, fremder Programme
- **BASIC-Kompaktor**
  Verkürtzt Ihre selbstgeschriebenen BASIC-Programme
- **Ein-/Ausgabesteuerung in Maschinensprache**
- **Joystick und Paddle**
- **Die Commodore-Maus 1351**
  Ein kompletter Maustreiber zum Abtippen
- **Der serielle IEC-Bus**
- **Zeilenweise kommentiertes ROM-Listing**
  Belegung der Zeropage, Nutzung der ROM-Listings, alphabetisches Listing aller ROM-Routinen
- **Grafikprogrammierung**
  Windowing, Multi-Color-Modus, Extended Background Color Mode, Fein- und Screenscrolling in alle Richtungen

ISBN 3-89011-307-9 DM +029.80

DM 29,80
ÖS 232,–
sFr 29,–

02980

9 783890 113074

DATA BECKER