KINGSOFT

Das große PLUS/4 Buch

Alles, was Sie schon immer
über den PLUS/4 wissen wollten.

COMMODORE PLUS/4

Fritz Schäfer

Das große Plus/4 Buch

COMMODORE PLUS/4 (C-116, C-16)

KINGSOFT

Wichtiger Hinweis!

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von dem Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Fritz Schäfer sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

INHALTSVERZEICHNIS

1. EINFÜHRUNG

Mit den Modellen C-116, C-16 und Plus/4 hat Commodore im Jahr 1984 drei Computer-Modelle herausgebracht, die trotz z.T. interessanter Leistungsmerkmale ein Mauerblümchendasein im Schatten des millionenfachen Bestsellers C-64 fristen. Mit diesem Buch wollen wir Ihnen zeigen, daß diese Modelle zu Unrecht vielfach unterschätzt werden. Denn sie bieten u.a. ein außergewöhnlich leistungsfähiges BASIC (wesentlich besser z.B. als das des C-64), hervorragende Grafik-Eigenschaften (320x200 Punkte in 121 Farben), vielfältige Erweiterungsmöglichkeiten und ein ständig wachsendes Software-Angebot aus allen Bereichen zu einem wirklichen Einsteiger-Preis.

Die Schwerpunkte in diesem Buch bilden Themen, die im Bedienungshandbuch nur unzureichend oder gar nicht behandelt werden. Es ist damit für alle diejenigen interessant, die ihre Kenntnisse in BASIC vertiefen wollen und mit kleinen Ausflügen in die Maschinensprache die vollen Möglichkeiten des PLUS/4 speziell bei Grafik und Sound ausnutzen wollen.

Ich werde in diesem Buch der Einfachheit halber immer vom PLUS/4 sprechen, da die Schwestermodelle C-116 und C-16 bis auf kleine Unterschiede (Tastatur, RAM) identisch sind. Sollten Abweichungen zu diesen Modellen auftreten, werde ich gezielt darauf hinweisen.

Sie werden nach der Lektüre dieses Buches Ihren Computer besser kennen und effektiver programmieren können. Ich habe mich bemüht, den Stoff mit zahlreichen Beispielprogrammen aufzulockern, da ja bekanntlich ein Bild mehr sagt als tausend Worte.

Abschließend möchte ich mich noch bei Birgit Menzenbach bedanken, die dieses Buch kritisch Korrektur gelesen hat.

Roetgen, im November 1986

2. GRAFIK

Eine der beeindruckendsten Eigenschaften aller Homecomputer ist ohne Zweifel die Grafik, die man mit diesen Geräten erzeugen kann. Der PLUS/4 hat hier eine seiner stärksten Seiten, denn welcher andere Homecomputer bietet eine Auflösung von 320x200 Punkten in 121 Farben, die von einem äußerst leistungsstarken BASIC-Befehlssatz unterstützt wird. Wir werden in diesem Kapitel mehrere Arten kennenlernen, auf dem PLUS/4 Grafiken zu erzeugen. Dabei werden wir uns auch mit der Möglichkeit beschäftigen, Grafik mit Hilfe von selbst definierten Zeichensätzen zu erzeugen. Der Vorteil dieser Methode liegt darin, daß weniger Speicherplatz verbraucht wird und größere Geschwindigkeiten erzielt werden können. Grafiken mit hoher Auflösung und in vielen Farben kosten bei jedem Computer viel Speicherplatz und so muß auch der PLUS/4 beim Arbeiten im hochauflösenden Grafik-Modus 10K für den Grafik-Speicher reservieren. Dagegen kostet ein eigener Zeichensatz nur 1K oder 2K RAM.

Ich will Ihnen in den folgenden Kapiteln zeigen, daß Sie beim PLUS/4 auch mit wenig Speicheraufwand hervorragende (und schnelle) Grafiken programmieren können. Schließlich laufen viele kommerziellen Spielprogramme schon auf der Grundversion des C-16 mit 16K-RAM, und darunter gibt es sicher einige, denen man den knappen Arbeitsspeicher nicht ansieht.

2.1 STANDARD-GRAFIK

Die einfachste Art, auf dem PLUS/4 Grafik zu erzeugen, besteht in dem vorhandenen Zeichenvorrat, der direkt von der Tastatur mit der COMMODORE- und der SHIFT-Taste abgerufen werden kann. Nebenbei bemerkt kostet diese Grafik auch keinerlei zusätzlichen Speicherplatz und ist daher hauptsächlich für längere Programme mit einfachen Grafik-Anforderungen geeignet. Sie können alle auf der Tastatur aufgedruckten Symbole in 121 verschiedenen Farben, wahlweise invertiert oder blinkend darstellen. Durch geeignete Wahl der Symbole können viele Sachen schnell und bequem dargestellt werden, z.B. Spielkarten, Rechtecke, usw.

Normalerweise werden diese Grafiken in BASIC mit PRINT-Befehlen programmiert, in denen die Grafik-Symbole, die Farbsteuerung usw. enthalten sind. In manchen Fällen kann es aber durchaus sinnvoll (oder unvermeidlich) sein, die gewünschten Zeichen direkt in den Grafik-Speicher des PLUS/4 zu schreiben. Wenn Sie z.B. versuchen, mit einem PRINT-Befehl in die untere rechte Ecke des Bildschirms (25. Zeile, 40. Spalte) ein Zeichen zu schreiben, so werden Sie feststellen, daß der Bildschirm nach dem ausgeführten PRINT-Befehl nach oben hochscrollt, wodurch die untere rechte Ecke wieder frei wird. Mit einem Direktzugriff in den Grafik-Speicher läßt sich dieses Problem aber einfach und schnell lösen.

Schauen wir uns daher zunächst einmal an, wie der PLUS/4 intern den Bildschirm verwaltet. Zu jedem der 1000 (40x25) darstellbaren Zeichen gehören im RAM des PLUS/4 zwei Speicherplätze. In dem einen ist der interne Zeichencode enthalten (das sogenannte Video-RAM), im anderen die Farbe (Farb-RAM). Das Video-RAM liegt im Speicher ab 3072, das Farb-RAM ab 2048; der Zeichencode des ersten Zeichens (oben links, sogenannte HOME-Position) wird also bei 3072 abgelegt, der dazugehörige Farbwert bei 2048. Das nächste Zeichen im RAM ist das Zeichen rechts von der ersten Position usw. bis zum Ende der ersten Zeile. Am einfachsten kann man sich das an den folgenden Übersichten veranschaulichen:

VIDEO-RAM (3072-4071)

Spalte:	1	2	3	...	38	39	40
Zeile:							
1	3072	3073	3074	...	3109	3110	3111
2	3112						3151
3	3152						3191
.	.						.
.	.						.
.	.						.
23	3952						3991
24	3992						4031
25	4032	4033	4034	...	4069	4070	4071

FARB-RAM (2048-3047)

Spalte:	1	2	3	...	38	39	40
Zeile:							
1	2048	2049	2050	...	2085	2086	2087
2	2088						2127
3	2128						2167
.	.						.
.	.						.
.	.						.
23	2928						2967
24	2968						3007
25	3008	3009	3010	...	3045	3046	3047

Die Codierung der einzelnen Zeichen finden Sie im Anhang Ihres Bedienungshandbuches (Seite 212 beim PLUS/4). So wird der Buchstabe A im Video-RAM als Wert 1 abgelegt. Der Wert, der im Farb-RAM abgespeichert wird, setzt sich zusammen aus der Farbe, der Helligkeit und der Information, ob das Zeichen blinkend dargestellt wird oder nicht. Das nachfolgende kurze Programm löscht den Bildschirm und schreibt ein rotes, blinkendes Herz in die rechte untere Bildschirmecke:

```
100 SCNCLR
110 POKE 4071,83
120 POKE 3047,184
```

Durch das direkte Schreiben in den Bildschirm- und Farbspeicher ändert sich übrigens nichts an der augenblicklichen Position des Cursors; auch das manchmal unerwünschte Scrollen (s.o.) unterbleibt.

2.2 DIE GRAFIK-MÖGLICHKEITEN DES TED

In diesem Grafik-Kapitel werden wir uns natürlich bevorzugt mit den Registern des TED beschäftigen, die für die Grafik verantwortlich sind. Bevor wir ins Detail gehen, gebe ich Lhnen erst einmal einen Überblick, welche grafischen Möglichkeiten uns der TED-Chip des PLUS/4 anbietet:

* Frei programmierbarer Zeichensatz von 256 Zeichen
* Erzeugung der reversen Zeichen soft- oder hardwaremäßig
* Standard- und Multicolour-Zeichen
* Extended Colour Modus
* Standard Hires Bitmap Modus
* Multicolour Bitmap Modus
* Programmierbarer Rasterinterrupt

Nachfolgend werden wir auf alle diese Möglichkeiten näher eingehen und diese mit Beispielprogrammen erläutern.

2.3 FREI PROGRAMMIERBARER ZEICHENSATZ

Wie Sie in dem Kapitel 2.1 bereits erfahren haben, besteht die einfachste und speicherplatzsparendste Grafik-Möglichkeit beim PLUS/4 in der Verwendung der vorgegebenen Grafik-Zeichen. Damit lassen sich maximal 128 Zeichen darstellen sowie weitere 128 reverse, macht also insgesamt 256. Leider befindet sich auch unter diesen 256 Zeichen oft nicht das gewünschte, so daß man sich fragt, ob man sich nicht ein Zeichen selbst definieren kann. Um keine unnötige Spannung aufzubauen - es ist möglich! Man kann dem TED-Chip nämlich mitteilen, woher er seine Informationen zur Erzeugung von eigenen Zeichen nehmen soll.

Normalerweise bezieht der TED seine Informationen über den Zeichensatz aus einem speziellen ROM, dem sogenannten Character-ROM. In diesem ist das Aussehen aller 128 Standard-Zeichen einprogrammiert. Wenn man dem TED-Chip nun mitteilt, daß er seine Informationen aus dem RAM beziehen soll, so können wir eigene Zeichen definieren. Pfiffigerweise bietet der TED-Chip sogar zwei Möglichkeiten eines eigenen Zeichensatzes: zum einen kann man den gesamten Zeichensatz, also 256 Zeichen, neu definieren, zum anderen ist es möglich, nur 128 Zeichen neu zu definieren, die entsprechenden reversen Zeichen erzeugt der TED-Chip dann selbst. Letzteres hat den großen Vorteil, daß Speicherplatz gespart wird. So müssen wir für einen kompletten Zeichensatz (256 Zeichen) 2K RAM reservieren, im anderen Fall (128 Zeichen) wird sogar nur 1K RAM benötigt!

Beschäftigen wir uns also überwiegend mit der zweiten Möglichkeit, denn mit 128 selbst definierbaren Zeichen läßt sich eigentlich fast jedes Problem lösen. Schauen wir uns doch einmal an, wie ein Zeichen sich zusammensetzt: es besteht immer aus einer Matrix von 8x8 Punkten. Betrachten wir einmal den Buchstaben **A**, so sieht seine Abbildung wie folgt aus:

```
...**...
..****..
.**..**.
.******.
.**..**.
.**..**.
.**..**.
........
```

Dabei steht ein Stern * immer für einen sichtbaren, und ein Punkt . immer für einen unsichtbaren Punkt. Innerhalb dieser 8x8-Matrix können wir uns jetzt auch beliebige Zeichen ausdenken. Bevor man eigene Zeichen entwirft und ausprobiert, empfiehlt es sich immer, erst den vorhandenen Zeichensatz aus dem ROM ins RAM zu kopieren, und dann einige Zeichen, die nicht gebraucht werden, durch eigene zu ersetzen. In dem nachfolgenden Programm wird zuerst der Zeichensatz ins RAM kopiert und der entsprechende Bereich danach vor dem Überschreiben durch ein BASIC-Programm geschützt. Das Zeichen für den Klammeraffen wird durch ein Grad-Symbol ersetzt und in einem kurzen Satz demonstriert. Geben Sie bitte einmal folgendes Beispiel ein:

```
100 POKE 55,0 : POKE 56,60 : CLR
110 POKE 1177,62
120 FOR I=0 TO 1023
130 : POKE 60*256+I,PEEK(53248+I)
140 NEXT I
150 POKE 1177,63
160 POKE 65299,60
170 POKE 65298,192
180 POKE 1351,128
190 FOR I=0 TO 7
200 : READ A
210 : POKE 60*256+I,A
220 NEXT I
230 SCNCLR
240 CHAR 1,6,12,"DIE TEMPERATUR BETRAEGT 24@C"
250 CHAR 1,2,23,"BITTE TASTE FUER NORMALEN ZEICHENSATZ"
260 GETKEY K$
270 POKE 65299,208
280 POKE 65298,196
290 POKE 1351,0
300 DATA 24,36,36,24,0,0,0,0
```

Hier nun eine zeilenweise Erklärung von diesem Programm:

100 Speicherplatz für Zeichensatz reservieren

110	Schalte PEEK-Routine auf ROM-Lesen
120-140	Kopiere Zeichensatz (1K) ins RAM
150	Schalte PEEK-Routine auf RAM-Lesen
160	Zeichensatz-Adresse auf 60*256 = 15360 ($3C00 hex)
170	Zeichensatz aus RAM statt ROM lesen
180	CBM/SHIFT-Taste abschalten
190-220	Zeichendefinition ins RAM poken
230	Bildschirm löschen
240-250	Text auf den Bildschirm ausgeben
260	Auf Taste warten
270-280	Zurück zum normalen Zeichensatz
290	CBM/SHIFT-Taste einschalten
300	Zeichendefinition für Grad-Symbol

2.4 HOCHAUFLÖSENDE GRAFIK

Beim Schreiben von Spielen, Zeichnen von Tabellen für Geschäftsanwendungen oder Schreiben von sonstigen Programmen werden Sie früher oder später Bildschirmdarstellungen mit hoher Auflösung benötigen.

Der Commodore PLUS/4 wurde genau hierfür konstruiert: Hohe Auflösung wird durch "Bit-Mapping" des Bildschirms möglich. "Bit-Mapping" ist die Methode, bei der jedem darstellbaren Punkt (Pixel) auf dem Bildschirm sein eigenes Bit (Platz) im Speicher zugeordnet wird. Ist dieses Speicherbit eine 1, so ist der entsprechende Punkt eingeschaltet. Ist das Bit eine 0, so ist der Punkt ausgeschaltet.

Das Arbeiten mit Grafiken hoher Auflösung hat jedoch einige Nachteile und wird daher nicht immer benutzt. Zunächst wird durch das Bit-Mapping des gesamten Bildschirms erhebliche Speicherkapazität in Anspruch genommen. Jeder Pixel benötigt nämlich ein Speicherbit, d.h. Sie brauchen 1 Byte für 8 Pixel. Da jedes Zeichen aus einer 8x8-Matrix besteht und 40 Zeilen mal 25 Spalten vorhanden sind, beträgt die Auflösung 320 mal 200 Punkte oder Pixel. Hieraus ergeben sich 64000 Pixel, von denen jedes ein Speicherbit benötigt. Für ein Bit-Mapping des gesamten Bildschirms brauchen Sie also 8000 Bytes.

Um Ihnen die Nachteile des Hires-Modus vor Augen zu führen, geben Sie mal das nachfolgende Beispiel ein. Sie werden sehen, daß es zu ungewollten Farbeffekten kommt. Dies liegt daran, daß der PLUS/4 im Hires-Modus in jedem 8x8-Feld jeweils nur eine Vordergrund- und Hintergrund-Farbe darstellen kann. Wenn Sie versuchen, ein bereits benutztes 8x8-Feld neu zu beschreiben, so wird dieses umgefärbt. Genau dies macht unser Demo:

```
100 GRAPHIC 1,1
110 FOR X=0 TO 190 STEP 10
120 : COLOR 1,((X/10)AND15)+1,5
130 : BOX 1,X,X,X+16,X+10,,1
140 NEXT X
```

```
150 FOR X=0 TO 192 STEP 8
160 : COLOR 1,((X/8)AND15)+1,5
170 : BOX 1,X+64,X,X+80,X+8,,1
180 NEXT X
190 COLOR 1,1,0 : CHAR 1,1,20,"BITTE TASTE"
200 GETKEY K$ : GRAPHIC 0
```

Eine mögliche Abhilfe für dieses Problem liefert der Multicolour-Modus, der im nächsten Kapitel besprochen wird. Als letztes Beispiel für den Hires-Modus wollen wir Ihnen ein Mini-Malprogramm in Maschinensprache präsentieren. Alle, die noch keine Ahnung von Maschinensprache haben, sollten jetzt erstmal Kapitel 4 studieren, bevor Sie sich um das nachfolgende Programm kümmern. Wenn Sie Maschinensprache und den eingebauten Monitor bereits soweit beherrschen, das Sie Programm eingeben und abspeichern können, dann dürfte Ihnen das nachfolgende Programm keine Probleme bereiten. Zur Eingabe verwenden Sie dann am besten den Hexdump, den Sie mit der nachfolgenden Zeile auf Kassette bzw. Diskette abspeichern können. Abschließend folgt noch ein disassembliertes Listing zum leichteren Verständnis des Programms. Sie können das Programm entweder in BASIC mit: **RUN** oder im Maschinensprache-Monitor mit: **G 1010** starten. Bedenken Sie stets, das Sie ein Maschinenprogramm **immer zuerst abspeichern**, und **dann ausprobieren** sollten! Die Bedienung des Malprogramms ist übrigens sehr einfach: mit dem Joystick (in Port 2) können Sie Linien in allen Richtungen zeichnen. Sobald Sie den Knopf am Joystick drücken, können Sie die Linien wieder löschen, sobald Sie über sie hinweg fahren. Mit der Leertaste löschen Sie Ihr gesamtes Kunstwerk. Für fortgeschrittene Maschinensprache-Programmierer kann dieses Programm durchaus als Gerüst dienen für eigene Erweiterungen; nur keine Hemmungen, mehr als "abstürzen" kann der PLUS/4 auch nicht!

```
>1000 00 0B 10 00 00 9E 34 31 :.......41
>1008 31 32 00 00 00 00 00 00 :12......
>1010 A9 36 8D 06 FF A9 C8 8D :)6.)H.
>1018 12 FF 20 EA 10 A2 00 A9 : j.".)
>1020 33 9D 00 08 9D 00 09 9D :3.......
>1028 00 0A 9D 00 0B A9 05 9D :.....)..
>1030 00 0C 9D 00 0D 9D 00 0E :........
>1038 9D 00 0F CA D0 E1 86 D6 :...JPa.V
```

```
>1040 A9 A0 85 D5 A9 64 85 D4 :) .U)d.T
>1048 E8 86 D2 20 E4 FF C9 20 :h.R I
>1050 D0 03 20 EA 10 20 C5 10 :P. j. E.
>1058 D0 02 E6 D2 A5 D0 F0 33 :P.fR%Pp3
>1060 30 1C A5 D5 18 69 01 85 :0.%U.i..
>1068 D5 AA A5 D6 69 00 85 D6 :U*%Vi..V
>1070 F0 21 E0 40 90 1D A9 00 :p!`@..).
>1078 85 D5 85 D6 F0 15 38 A5 :.U.Vp.8%
>1080 D5 E9 01 85 D5 A5 D6 E9 :Ui..U%Vi
>1088 00 B0 06 A9 3F 85 D5 A9 :.0.)?.U)
>1090 01 85 D6 A5 D1 F0 1B 30 :..V%Qp.0
>1098 0E A4 D4 C8 C0 C8 D0 02 :.$TH@HP.
>10A0 A0 00 84 D4 4C B2 10 A5 : ..TL2.%
>10A8 D4 38 E9 01 B0 02 A9 C7 :T8i.0.)G
>10B0 85 D4 A5 D2 4A 20 02 11 :.T%RJ ..
>10B8 A2 0F A0 FF 88 D0 FD CA :"..P)J
>10C0 D0 F8 4C 4B 10 78 A9 FD :PxLK.x))
>10C8 8D 08 FF AD 08 FF A0 00 :.- .
>10D0 A2 00 4A B0 01 88 4A B0 :".J0..J0
>10D8 01 C8 4A B0 01 CA 4A B0 :.HJ0.JJ0
>10E0 01 E8 86 D0 84 D1 29 08 :.h.P.Q).
>10E8 58 60 A9 00 85 D7 A9 20 :X`)..W)
>10F0 85 D8 A2 20 A0 00 98 91 :.X"  ...
>10F8 D7 88 D0 FB E6 D8 CA D0 :W.P{fXJP
>1100 F6 60 B0 0B 20 1A 11 BD :v`0. ..=
>1108 64 11 31 D7 91 D7 60 20 :d.1W.W`
>1110 1A 11 BD 5C 11 11 D7 91 :..=\..W.
>1118 D7 60 A5 D4 29 07 A8 A5 :W`%T).(%
>1120 D4 29 F8 85 D7 A9 00 06 :T)x.W)..
>1128 D7 2A 06 D7 2A 06 D7 2A :W*.W*.W*
>1130 85 D8 85 D3 A5 D7 0A 26 :.X.S%W.&
>1138 D3 0A 26 D3 65 D7 85 D7 :S.&SeW.W
>1140 A5 D8 65 D3 85 D8 A5 D5 :%XeS.X%U
>1148 29 07 AA A5 D5 29 F8 65 :).*%U)xe
>1150 D7 85 D7 A5 D6 65 D8 69 :W.W%VeXi
>1158 20 85 D8 60 80 40 20 10 : .X`.@ .
>1160 08 04 02 01 7F BF DF EF :...?_o
>1168 F7 FB FD FE DB 20 37 20 :w{}~[ 7
```

```
S"HIRESDRAW",01,1001,116C
```

```
. 1010  A9 36     LDA #$36
. 1012  8D 06 FF  STA $FF06
. 1015  A9 C8     LDA #$C8
. 1017  8D 12 FF  STA $FF12
. 101A  20 EA 10  JSR $10EA
. 101D  A2 00     LDX #$00
. 101F  A9 33     LDA #$33
. 1021  9D 00 08  STA $0800,X
. 1024  9D 00 09  STA $0900,X
. 1027  9D 00 0A  STA $0A00,X
. 102A  9D 00 0B  STA $0B00,X
. 102D  A9 05     LDA #$05
. 102F  9D 00 0C  STA $0C00,X
. 1032  9D 00 0D  STA $0D00,X
. 1035  9D 00 0E  STA $0E00,X
. 1038  9D 00 0F  STA $0F00,X
. 103B  CA        DEX
. 103C  D0 E1     BNE $101F
. 103E  86 D6     STX $D6
. 1040  A9 A0     LDA #$A0
. 1042  85 D5     STA $D5
. 1044  A9 64     LDA #$64
. 1046  85 D4     STA $D4
. 1048  E8        INX
. 1049  86 D2     STX $D2
. 104B  20 E4 FF  JSR $FFE4
. 104E  C9 20     CMP #$20
. 1050  D0 03     BNE $1055
. 1052  20 EA 10  JSR $10EA
. 1055  20 C5 10  JSR $10C5
. 1058  D0 02     BNE $105C
. 105A  E6 D2     INC $D2
. 105C  A5 D0     LDA $D0
. 105E  F0 33     BEQ $1093
. 1060  30 1C     BMI $107E
. 1062  A5 D5     LDA $D5
. 1064  18        CLC
. 1065  69 01     ADC #$01
. 1067  85 D5     STA $D5
. 1069  AA        TAX
. 106A  A5 D6     LDA $D6
```

```
. 106C  69 00     ADC #$00
. 106E  85 D6     STA $D6
. 1070  F0 21     BEQ $1093
. 1072  E0 40     CPX #$40
. 1074  90 1D     BCC $1093
. 1076  A9 00     LDA #$00
. 1078  85 D5     STA $D5
. 107A  85 D6     STA $D6
. 107C  F0 15     BEQ $1093
. 107E  38        SEC
. 107F  A5 D5     LDA $D5
. 1081  E9 01     SBC #$01
. 1083  85 D5     STA $D5
. 1085  A5 D6     LDA $D6
. 1087  E9 00     SBC #$00
. 1089  B0 06     BCS $1091
. 108B  A9 3F     LDA #$3F
. 108D  85 D5     STA $D5
. 108F  A9 01     LDA #$01
. 1091  85 D6     STA $D6
. 1093  A5 D1     LDA $D1
. 1095  F0 1B     BEQ $10B2
. 1097  30 0E     BMI $10A7
. 1099  A4 D4     LDY $D4
. 109B  C8        INY
. 109C  C0 C8     CPY #$C8
. 109E  D0 02     BNE $10A2
. 10A0  A0 00     LDY #$00
. 10A2  84 D4     STY $D4
. 10A4  4C B2 10  JMP $10B2
. 10A7  A5 D4     LDA $D4
. 10A9  38        SEC
. 10AA  E9 01     SBC #$01
. 10AC  B0 02     BCS $10B0
. 10AE  A9 C7     LDA #$C7
. 10B0  85 D4     STA $D4
. 10B2  A5 D2     LDA $D2
. 10B4  4A        LSR
. 10B5  20 02 11  JSR $1102
. 10B8  A2 0F     LDX #$0F
. 10BA  A0 FF     LDY #$FF
```

```
. 10BC  88       DEY
. 10BD  D0 FD    BNE $10BC
. 10BF  CA       DEX
. 10C0  D0 F8    BNE $10BA
. 10C2  4C 4B 10 JMP $104B
. 10C5  78       SEI
. 10C6  A9 FD    LDA #$FD
. 10C8  8D 08 FF STA $FF08
. 10CB  AD 08 FF LDA $FF08
. 10CE  A0 00    LDY #$00
. 10D0  A2 00    LDX #$00
. 10D2  4A       LSR
. 10D3  B0 01    BCS $10D6
. 10D5  88       DEY
. 10D6  4A       LSR
. 10D7  B0 01    BCS $10DA
. 10D9  C8       INY
. 10DA  4A       LSR
. 10DB  B0 01    BCS $10DE
. 10DD  CA       DEX
. 10DE  4A       LSR
. 10DF  B0 01    BCS $10E2
. 10E1  E8       INX
. 10E2  86 D0    STX $D0
. 10E4  84 D1    STY $D1
. 10E6  29 08    AND #$08
. 10E8  58       CLI
. 10E9  60       RTS
. 10EA  A9 00    LDA #$00
. 10EC  85 D7    STA $D7
. 10EE  A9 20    LDA #$20
. 10F0  85 D8    STA $D8
. 10F2  A2 20    LDX #$20
. 10F4  A0 00    LDY #$00
. 10F6  98       TYA
. 10F7  91 D7    STA ($D7),Y
. 10F9  88       DEY
. 10FA  D0 FB    BNE $10F7
. 10FC  E6 D8    INC $D8
. 10FE  CA       DEX
. 10FF  D0 F6    BNE $10F7
```

```
. 1101  60       RTS
. 1102  B0 0B    BCS $110F
. 1104  20 1A 11 JSR $111A
. 1107  BD 64 11 LDA $1164,X
. 110A  31 D7    AND ($D7),Y
. 110C  91 D7    STA ($D7),Y
. 110E  60       RTS
. 110F  20 1A 11 JSR $111A
. 1112  BD 5C 11 LDA $115C,X
. 1115  11 D7    ORA ($D7),Y
. 1117  91 D7    STA ($D7),Y
. 1119  60       RTS
. 111A  A5 D4    LDA $D4
. 111C  29 07    AND #$07
. 111E  A8       TAY
. 111F  A5 D4    LDA $D4
. 1121  29 F8    AND #$F8
. 1123  85 D7    STA $D7
. 1125  A9 00    LDA #$00
. 1127  06 D7    ASL $D7
. 1129  2A       ROL
. 112A  06 D7    ASL $D7
. 112C  2A       ROL
. 112D  06 D7    ASL $D7
. 112F  2A       ROL
. 1130  85 D8    STA $D8
. 1132  85 D3    STA $D3
. 1134  A5 D7    LDA $D7
. 1136  0A       ASL
. 1137  26 D3    ROL $D3
. 1139  0A       ASL
. 113A  26 D3    ROL $D3
. 113C  65 D7    ADC $D7
. 113E  85 D7    STA $D7
. 1140  A5 D8    LDA $D8
. 1142  65 D3    ADC $D3
. 1144  85 D8    STA $D8
. 1146  A5 D5    LDA $D5
. 1148  29 07    AND #$07
. 114A  AA       TAX
. 114B  A5 D5    LDA $D5
```

```
. 114D  29 F8     AND #$F8
. 114F  65 D7     ADC $D7
. 1151  85 D7     STA $D7
. 1153  A5 D6     LDA $D6
. 1155  65 D8     ADC $D8
. 1157  69 20     ADC #$20
. 1159  85 D8     STA $D8
. 115B  60        RTS

>115C 80 40 20 10 08 04 02 01 :.@ .....
>1164 7F BF DF EF F7 FB FD FE :?_ow{}~
```

2.5 MULTICOLOUR-GRAFIK

Durch die standardmäßige hochauflösende Grafik können Sie selbst Einzelpunkte auf dem Bildschirm ansteuern. Für jeden Punkt im Zeichenspeicher stehen zwei Werte zur Verfügung: eine 1 für "an" und 0 für "aus". Hat ein Punkt den Wert 1, so wird er in der von Ihnen für die jeweilige Bildschirmposition gewählten Farbe angezeigt. Bei der hochauflösenden Grafik können alle Punkte innerhalb einer 8x8-Matrix entweder in der Vorder- oder Hintergrundfarbe angezeigt werden. Hierdurch wird die Farbauflösung innerhalb dieses Bereiches eingeschränkt. Es können z.B. Schwierigkeiten entstehen, wenn sich zwei Linien mit verschiedenen Farben kreuzen (siehe dazu auch das letzte Beispiel im vorigen Kapitel).

Dieses Problem wird durch den Multicolour-Modus gelöst. Hierbei kann jeder Punkt eine von vier Farben haben: Bildschirmfarbe (Hintergrundregister 0, 65301), Hintergrundfarbe 1 (65302), Hintergrundfarbe 2 (65303) oder die Zeichenfarbe. Die einzige Einschränkung liegt in der horizontalen Auflösung, da im Multicolour-Modus jeder Punkt doppelt so breit ist wie bei der Hires-Grafik. Es überwiegen jedoch bei weitem die Vorteile des Multicolour-Modus. Dazu ein erstes Beispiel:

```
100 GRAPHIC 3,1
110 COLOR 0,1,0 : COLOR 1,3,0 : COLOR 2,8,7 : COLOR 3,13,4
120 FOR C=3 TO 1 STEP -1
130 : FOR A=0 TO 180 STEP 10
140 :   CIRCLE C,20+C*30,100,10,50,,,A,20
150 : NEXT A
160 NEXT C
170 COLOR 1,2,7 : CHAR 1,1,20,"BITTE TASTE"
180 GETKEY K$ : DRAW 0,0,100 TO 159,100
190 GETKEY K$ : GRAPHIC 0
```

Hier die zeilenweise Erklärung:

100 Multicolour-Grafik-Modus einschalten

110 Definition von Hintergrundfarbe 0, Vordergrundfarbe, Hintergrundfarben 1 & 2

```
120       Zeichne in jeder der drei Farben
130       Zeichne Ovale in verschiedenen Winkeln
140       Oval zeichnen
150       Nächster Winkel
160       Nächste Farbe
170       Zeichenfarbe auf weiß, Text ausgeben
180       Auf Taste warten. Linie ziehen
190       Auf Taste warten. Zurück zum normalen Text-Modus
```

Wie Sie sehen, werden die drei Kreise in verschiedenen Farben dargestellt, ohne daß es Nachfärbungen gibt. Erst wenn Sie auf Tastendruck eine weitere Farbe ins Spiel bringen, tritt dieser Effekt auf. In der Praxis (und in den meisten kommerziellen Spielen) wird hauptsächlich in diesem Grafik-Modus gearbeitet, der u.a. auch die Programmierung von Software-Sprites erlaubt (dazu später mehr).

Wegen des begrenzten Speicherplatzes ist es auch in diesem Modus sinnvoll, mit selbst definierten Zeichen zu arbeiten. Dabei müssen Sie allerdings beachten, daß Texteinblendungen nicht so ohne weiteres möglich sind, da der eingebaute Zeichensatz im PLUS/4 für den normalen Modus ausgelegt ist. Dadurch sehen die Buchstaben meist leicht verfremdet aus. Abhilfe schafft hier jedoch ein eigener Zeichensatz. Im Gegensatz zum normalen Modus wird bei den Multicolour-Zeichen nicht mehr jeweils ein Bit für ein Pixel verwendet, sondern jeweils 2 Bit bestimmen die Farbe. Dadurch stehen in der Breite nur noch 4 (breite) Pixel zur Verfügung (daher die Auflösung von 160x200). Die Zuordnung der Farben zu den Bitkombinationen ist wie folgt:

BITS	FARBQUELLE
00	Hintergrund 0 (65301)
01	" 1 (65302)
10	" 2 (65303)
11	Vordergrund

Betrachten wir einmal folgendes selbstdefiniertes Zeichen, wobei wir als Hintergrundfarbe 0 (H0) schwarz, als Hintergrundfarbe 1 (H1) rot, als Hintergrundfarbe 2 (H2) gelb und als Vordergrundfarbe (V0) grün gewählt haben:

BITMUSTER	FARBQUELLE	FARBEN
00 01 10 00	H0 H1 H2 H0	schwarz rot gelb schwarz
00 11 11 00	H0 V0 V0 H0	schwarz grün grün schwarz
01 10 01 10	H1 H2 H1 H2	rot gelb rot gelb
01 11 11 10	H1 V0 V0 H2	rot grün grün gelb
01 10 01 10	H1 H2 H1 H2	rot gelb rot gelb
01 10 01 10	H1 H2 H1 H2	rot gelb rot gelb
01 10 01 10	H1 H2 H1 H2	rot gelb rot gelb
00 00 00 00	H0 H0 H0 H0	schwarz schwarz schwarz schwarz

Das nachfolgende Beispiel kopiert wieder den normalen Zeichensatz ins RAM und erstellt dann eigene Multicolour-Zeichen für einen Schmetterling:

```
100 POKE 55,0 : POKE 56,60 : CLR
110 POKE 1177,62
120 FOR I=0 TO 1023
130 : POKE 60*256+I,PEEK(53248+I)
140 NEXT I
150 POKE 1177,63
160 POKE 65299,60
170 POKE 65298,192
180 POKE 65287,24
190 POKE 1351,128
200 POKE 65302,0 : POKE 65303,93
210 FOR I=0 TO 15
220 : READ A
230 : POKE 60*256+8*40+I,A
240 NEXT I
```

```
250 SCNCLR
260 PRINT TAB(5)CHR$(154)"HALLO IHR SCHMETTERLINGE"
270 PRINT : PRINT TAB(10)"()"SPC(10)CHR$(153)"()"
280 CHAR 1,2,23,"BITTE TASTE FUER NORMALEN ZEICHENSATZ"
290 GETKEY K$
300 POKE 65299,208
310 POKE 65298,196
320 POKE 65287,8
330 POKE 1351,0
340 DATA 196,241,237,237,253,253,241,193
350 DATA 76,60,236,236,252,252,60,12
```

Hier wieder die zeilenweise Erklärung:

100	Speicherplatz für Zeichensatz reservieren
110	Schalte PEEK-Routine auf ROM-Lesen
120-140	Kopiere Zeichensatz (1K) ins RAM
150	Schalte PEEK-Routine auf RAM-Lesen
160	Zeichensatz-Adresse auf 60*256 = 15360 ($3C00)
170	Zeichensatz aus RAM statt ROM lesen
180	Multicolour-Modus einschalten
190	CBM/SHIFT-Taste abschalten
200	Hintergrundfarbe 1 auf schwarz, 2 auf hellblau Nr. 5
210-240	Zeichendefinition von Schmetterling ins RAM poken
250	Bildschirm löschen
260-280	Text und Schmetterlinge ausgeben
290	Auf Taste warten
300-320	Standard-Werte wiederherstellen
330	CBM/SHIFT-Taste einschalten
340-350	Zeichendefintion für Multicolour-Schmetterling

Abschließend noch ein Beispiel für eine animierte Grafik, sowie eine Multicolour-Version des Mini-Malprogramms. Zuerst wandeln wir das vorherige Schmetterling-Demo ab, indem wir 3 verschiedene Schmetterlinge definieren und diese abwechselnd darstellen, und

zwar in der Folge: 1,2,3,2,1,2,3,2,1,... Dadurch entsteht der Eindruck, der Schmetterling würde sich bewegen.

Das nächste Beispiel ist eine Multicolour-Variante des Mini-Malprogramms, das Sie im vorigen Kapitel gesehen haben. Diesmal können Sie per Knopfdruck zwischen den 3 Multicolour-Farben wählen und mit der Leertaste wird der Bildschirm wieder gelöscht.

```
100 POKE 55,0 : POKE 56,60 : CLR
110 POKE 1177,62
120 FOR I=0 TO 1023
130 : POKE 60*256+I,PEEK(53248+I)
140 NEXT I
150 POKE 1177,63
160 POKE 65299,60
170 POKE 65298,192
180 POKE 65287,24
190 POKE 1351,128
200 POKE 65302,0 : POKE 65303,93
210 FOR I=0 TO 47
220 : READ A
230 : POKE 60*256+8*40+I,A
240 NEXT I
250 SCNCLR
260 PRINT TAB(6)CHR$(154)"HALLO   IHR   SCHMETTERLINGE"
270 CHAR 1,1,23,"BITTE TASTE  FUER  NORMALEN ZEICHENSATZ"
280 Y=24 : Y1=24
290 GET K$ : IF K$<>"" THEN PRINT CHR$(144) : GOTO 350
300 A$="()" : GOSUB 400
310 A$="*+" : GOSUB 400
320 A$=",-" : GOSUB 400
330 A$="*+" : GOSUB 400
340 GOTO 290
350 POKE 65299,208
360 POKE 65298,196
370 POKE 65287,8
380 POKE 1351,0
390 END
400 CHAR 1,12,Y1,"  "
```

```
410 CHAR 1,12,Y,CHR$(154)+A$
420 CHAR 1,18,Y1,"  "
430 CHAR 1,18,Y,CHR$(153)+A$
440 Y1=Y : Y=Y-1 : IF Y<0 THEN Y=24
450 RETURN
460 DATA 196,241,237,237,253,253,241,193
470 DATA 76,60,236,236,252,252,60,12
480 DATA 52,49,57,61,61,49,49,49
490 DATA 112,48,176,240,240,48,48,48
500 DATA 4,3,3,3,3,3,3,3
510 DATA 64,0,0,0,0,0,0,0
```

Hier die zeilenweise Erklärung:

100	Speicherplatz für Zeichensatz reservieren
110	Schalte PEEK-Routine auf ROM-Lesen
120-140	Kopiere Zeichensatz (1K) ins RAM
150	Schalte PEEK-Routine auf RAM-Lesen
160	Zeichensatz-Adresse auf 60*256 = 15360 ($3C00)
170	Zeichensatz aus RAM statt ROM lesen
180	Multicolour-Modus einschalten
190	CBM/SHIFT-Taste abschalten
200	Hintergrundfarbe 1 auf schwarz, 2 auf hellblau Nr. 5
210-240	Zeichendefinition von Schmetterling ins RAM poken
250	Bildschirm löschen
260-270	Text ausgeben
280	Start der Schmetterlinge am unteren Bildschirmrand
290	Auf Taste warten. Wenn ja, Programm beenden
300	1. Variante des Schmetterlings darstellen
310	2. " " " "
320	3. " " " "
330	4. " " " "
340	Zurück zum Schleifenbeginn
350-390	Normale Farben, Programm-Ende

400 Zwei Leerstellen in die vorherige Position des rechten Schmetterlings schreiben

410 Farbe auf blau, rechten Schmetterling darstellen

420 Zwei Leerstellen in die vorherige Position des linken Schmetterlings schreiben

430 Farbe auf grün, linken Schmetterling darstellen

440 Derzeitige Position in Y1 zwischenspeichern. Y erniedrigen (eine Zeile höher), am oberen Rand Y wieder auf 24.

450 Unterprogramm Ende

460-510 Definition der 3 Varianten des Schmetterlings

Hier nun das letzte Beispiel, das Mini-Malprogramm, das Sie schon aus dem Hires-Kapitel kennen, in Multicolour:

```
>1000 00 0B 10 00 00 9E 34 31 :.......41
>1008 31 32 00 00 00 00 00 00 :12......
>1010 A9 36 8D 06 FF A9 18 8D :)6.)..
>1018 07 FF A9 C8 8D 12 FF 20 :)H.
>1020 CE 10 A2 00 A9 33 9D 00 :N.".)3..
>1028 08 9D 00 09 9D 00 0A 9D :........
>1030 00 0B A9 52 9D 00 0C 9D :..)R....
>1038 00 0D 9D 00 0E 9D 00 0F :........
>1040 CA D0 E1 A9 30 8D 15 FF :JPa)0.
>1048 A9 77 8D 16 FF E8 86 D2 :)w.h.R
>1050 A9 50 85 D6 A9 64 85 D5 :)P.V)d.U
>1058 20 E4 FF C9 20 D0 03 20 : I P.
>1060 CE 10 20 A9 10 D0 08 E6 :N. ).P.f
>1068 D2 A5 D2 29 03 85 D2 A5 :R%R)..R%
>1070 D0 18 65 D6 C9 FF D0 04 :P.eVP.
>1078 A9 9F D0 06 C9 A0 90 02 :).P.I ..
>1080 A9 00 85 D6 A5 D1 18 65 :)..V%Q.e
>1088 D5 C9 FF D0 04 A9 C7 D0 :UP.)GP
>1090 06 C9 C8 90 02 A9 00 85 :.IH..)..
>1098 D5 20 E6 10 A2 0F A0 FF :U f.".
>10A0 88 D0 FD CA D0 F8 4C 58 :.P)JPxLX
```

```
>10A8 10 78 A9 FD 8D 08 FF AD :.x)}.-
>10B0 08 FF A0 00 A2 00 4A B0 : .".J0
>10B8 01 88 4A B0 01 C8 4A B0 :..J0.HJ0
>10C0 01 CA 4A B0 01 E8 86 D0 :.JJ0.h.P
>10C8 84 D1 29 08 58 60 A9 00 :.Q).X`).
>10D0 85 D7 A9 20 85 D8 A2 20 :.W) .X"
>10D8 A0 00 98 91 D7 88 D0 FB : ...W.P{
>10E0 E6 D8 CA D0 F6 60 A5 D5 :fXJPv`%U
>10E8 29 07 A8 A5 D5 29 F8 85 :).(%U)x.
>10F0 D7 A9 00 85 D4 06 D7 2A :W)..T.W*
>10F8 06 D7 2A 06 D7 2A 85 D8 :.W*.W*.X
>1100 85 D3 A5 D7 0A 26 D3 0A :.S%W.&S.
>1108 26 D3 65 D7 85 D7 A5 D8 :&SeW.W%X
>1110 65 D3 85 D8 A5 D6 29 FC :eS.X%V)¦
>1118 0A 26 D4 65 D7 85 D7 A9 :.&TeW.W)
>1120 20 65 D8 65 D4 85 D8 A5 : eXeT.X%
>1128 D6 29 03 AA B1 D7 3D 42 :V).*1W=B
>1130 11 85 D3 BD 42 11 49 FF :..S=B.
>1138 A6 D2 3D 46 11 05 D3 91 :&R=F..S.
>1140 D7 60 3F CF F3 FC 00 55 :W`?Os¦.U
>1148 AA FF AA AA AA AA AA AA :******
```

S"MULTIDRAW",01,1001,114A

```
. 1010  A9 36    LDA #$36
. 1012  8D 06 FF STA $FF06
. 1015  A9 18    LDA #$18
. 1017  8D 07 FF STA $FF07
. 101A  A9 C8    LDA #$C8
. 101C  8D 12 FF STA $FF12
. 101F  20 CE 10 JSR $10CE
. 1022  A2 00    LDX #$00
. 1024  A9 33    LDA #$33
. 1026  9D 00 08 STA $0800,X
. 1029  9D 00 09 STA $0900,X
. 102C  9D 00 0A STA $0A00,X
. 102F  9D 00 0B STA $0B00,X
. 1032  A9 52    LDA #$52
. 1034  9D 00 0C STA $0C00,X
. 1037  9D 00 0D STA $0D00,X
. 103A  9D 00 0E STA $0E00,X
. 103D  9D 00 0F STA $0F00,X
```

```
. 1040  CA       DEX
. 1041  D0 E1    BNE $1024
. 1043  A9 30    LDA #$30
. 1045  8D 15 FF STA $FF15
. 1048  A9 77    LDA #$77
. 104A  8D 16 FF STA $FF16
. 104D  E8       INX
. 104E  86 D2    STX $D2
. 1050  A9 50    LDA #$50
. 1052  85 D6    STA $D6
. 1054  A9 64    LDA #$64
. 1056  85 D5    STA $D5
. 1058  20 E4 FF JSR $FFE4
. 105B  C9 20    CMP #$20
. 105D  D0 03    BNE $1062
. 105F  20 CE 10 JSR $10CE
. 1062  20 A9 10 JSR $10A9
. 1065  D0 08    BNE $106F
. 1067  E6 D2    INC $D2
. 1069  A5 D2    LDA $D2
. 106B  29 03    AND #$03
. 106D  85 D2    STA $D2
. 106F  A5 D0    LDA $D0
. 1071  18       CLC
. 1072  65 D6    ADC $D6
. 1074  C9 FF    CMP #$FF
. 1076  D0 04    BNE $107C
. 1078  A9 9F    LDA #$9F
. 107A  D0 06    BNE $1082
. 107C  C9 A0    CMP #$A0
. 107E  90 02    BCC $1082
. 1080  A9 00    LDA #$00
. 1082  85 D6    STA $D6
. 1084  A5 D1    LDA $D1
. 1086  18       CLC
. 1087  65 D5    ADC $D5
. 1089  C9 FF    CMP #$FF
. 108B  D0 04    BNE $1091
. 108D  A9 C7    LDA #$C7
. 108F  D0 06    BNE $1097
. 1091  C9 C8    CMP #$C8
. 1093  90 02    BCC $1097
. 1095  A9 00    LDA #$00
```

```
. 1097  85 D5     STA $D5
. 1099  20 E6 10  JSR $10E6
. 109C  A2 0F     LDX #$0F
. 109E  A0 FF     LDY #$FF
. 10A0  88        DEY
. 10A1  D0 FD     BNE $10A0
. 10A3  CA        DEX
. 10A4  D0 F8     BNE $109E
. 10A6  4C 58 10  JMP $1058
. 10A9  78        SEI
. 10AA  A9 FD     LDA #$FD
. 10AC  8D 08 FF  STA $FF08
. 10AF  AD 08 FF  LDA $FF08
. 10B2  A0 00     LDY #$00
. 10B4  A2 00     LDX #$00
. 10B6  4A        LSR
. 10B7  B0 01     BCS $10BA
. 10B9  88        DEY
. 10BA  4A        LSR
. 10BB  B0 01     BCS $10BE
. 10BD  C8        INY
. 10BE  4A        LSR
. 10BF  B0 01     BCS $10C2
. 10C1  CA        DEX
. 10C2  4A        LSR
. 10C3  B0 01     BCS $10C6
. 10C5  E8        INX
. 10C6  86 D0     STX $D0
. 10C8  84 D1     STY $D1
. 10CA  29 08     AND #$08
. 10CC  58        CLI
. 10CD  60        RTS
. 10CE  A9 00     LDA #$00
. 10D0  85 D7     STA $D7
. 10D2  A9 20     LDA #$20
. 10D4  85 D8     STA $D8
. 10D6  A2 20     LDX #$20
. 10D8  A0 00     LDY #$00
. 10DA  98        TYA
. 10DB  91 D7     STA ($D7),Y
```

```
. 10DD  88       DEY
. 10DE  D0 FB    BNE $10DB
. 10E0  E6 D8    INC $D8
. 10E2  CA       DEX
. 10E3  D0 F6    BNE $10DB
. 10E5  60       RTS
. 10E6  A5 D5    LDA $D5
. 10E8  29 07    AND #$07
. 10EA  A8       TAY
. 10EB  A5 D5    LDA $D5
. 10ED  29 F8    AND #$F8
. 10EF  85 D7    STA $D7
. 10F1  A9 00    LDA #$00
. 10F3  85 D4    STA $D4
. 10F5  06 D7    ASL $D7
. 10F7  2A       ROL
. 10F8  06 D7    ASL $D7
. 10FA  2A       ROL
. 10FB  06 D7    ASL $D7
. 10FD  2A       ROL
. 10FE  85 D8    STA $D8
. 1100  85 D3    STA $D3
. 1102  A5 D7    LDA $D7
. 1104  0A       ASL
. 1105  26 D3    ROL $D3
. 1107  0A       ASL
. 1108  26 D3    ROL $D3
. 110A  65 D7    ADC $D7
. 110C  85 D7    STA $D7
. 110E  A5 D8    LDA $D8
. 1110  65 D3    ADC $D3
. 1112  85 D8    STA $D8
. 1114  A5 D6    LDA $D6
. 1116  29 FC    AND #$FC
. 1118  0A       ASL
. 1119  26 D4    ROL $D4
. 111B  65 D7    ADC $D7
. 111D  85 D7    STA $D7
. 111F  A9 20    LDA #$20
. 1121  65 D8    ADC $D8
```

```
. 1123  65 D4     ADC $D4
. 1125  85 D8     STA $D8
. 1127  A5 D6     LDA $D6
. 1129  29 03     AND #$03
. 112B  AA        TAX
. 112C  B1 D7     LDA ($D7),Y
. 112E  3D 42 11  AND $1142,X
. 1131  85 D3     STA $D3
. 1133  BD 42 11  LDA $1142,X
. 1136  49 FF     EOR #$FF
. 1138  A6 D2     LDX $D2
. 113A  3D 46 11  AND $1146,X
. 113D  05 D3     ORA $D3
. 113F  91 D7     STA ($D7),Y
. 1141  60        RTS

>1142 3F CF F3 FC 00 55 AA FF :?Os:.U
```

2.6 EXTENDED-COLOUR-GRAFIK

In diesem, allgemein leider nur wenig bekannten, Modus können Sie jedem Zeichen auf dem Bildschirm eine der 4 Hintergrundfarben zuordnen. So ist es z.B. möglich, auf einem weißen Bildschirm ein blaues Zeichen mit gelbem Hintergrund anzuzeigen. Diese Farbenvielfalt hat natürlich auch ihren Preis: wir können nur noch 64 verschiedene Zeichen verwenden, da die obersten beiden Bits des Zeichens zur Definition der Hintergrundfarbe herangezogen werden müssen. Die nachfolgende Tabelle soll dies verdeutlichen:

BILDSCHIRMCODE	ZEICHEN	HINTERGRUND-FARBREGISTER
0 - 63	0 - 63	65301
64 - 127	0 - 63	65302
128 - 191	0 - 63	65303
192 - 255	0 - 63	65304

Wie immer soll Ihnen auch hier ein Demo-Programm die praktische Anwendung dieses Modus vor Augen führen:

```
100 SCNCLR
110 COLOR 0,1,2
120 COLOR 1,2,3
130 COLOR 2,4,5
140 COLOR 3,6,7
150 POKE 65286,PEEK(65286)OR64
160 FOR I=0 TO 63
170 : POKE 3072+I,I
180 : POKE 3136+I,I+64
190 : POKE 3200+I,I+128
200 : POKE 3264+I,I+192
210 NEXT I
220 I=0
230 FOR C=0 TO 15
240 : FOR L=0 TO 7
250 :   POKE 2048+I,L*16+C
```

```
260 :    POKE 2176+I,L*16+C
270 :    I=I+1
280 : NEXT L
290 NEXT C
300 CHAR 1,14,12,"BITTE TASTE" : GETKEY K$ : POKE 65286,PEEK(65286)
310 COLOR 0,2,7
```

Hier die zeilenweise Erklärung:

100	Bildschirm löschen
110-140	Hintergrundfarben setzen
150	Extended Colour Modus einschalten
160	Für jedes mögliche Zeichen:
170	Einen Satz mit Hintergrundfarbe 1 erzeugen
180	" " " " 2 "
190	" " " " 3 "
200	" " " " 4 "
210	Nächstes Zeichen
220	Variable I auf 0 setzen
230-240	Für jede Farbe und Helligkeit:
250-260	Farbe und Helligkeit in den Farbspeicher schreiben
270	Zeiger erhöhen
280-290	Nächste Helligkeit und Farbe
300	Text ausgeben, auf Taste warten, Normal-Modus
310	Normale Hintergrundfarbe (weiß) wieder einschalten

2.7 WEICHES SCROLLING

Wenn die letzte Zeile des PLUS/4-Bildschirms mit Text gefüllt wird, wird das gesamte Bild automatisch um eine Zeile nach oben gescrollt. Das heißt, daß die oberste Zeile entfernt wird, und die anderen Zeilen jeweils um eine Position nach oben nachrücken. Dieses Nachrücken geschieht normalerweise zeilenweise, aber es gibt durchaus andere Gelegenheiten, wo es wünschenswert wäre, wenn das Nachrücken weich, d.h. um jeweils 1 Pixel nach oben gescrollt würde.

Auch diese Möglichkeit bietet uns der TED-Grafikchip und zumindest das vertikale Scrolling ist sogar in BASIC programmierbar. Als erstes muß die Anzahl der Bildschirmzeilen auf 24 reduziert werden (Bit 3 vom Register 65286). Die unteren 3 Bits des gleichen Registers definieren die Scroll-Position von 0 bis 7. Sobald dieser Wert bei 7 angekommen ist, ist eine komplette neue Zeile sichtbar geworden; wir müssen dann diesen Wert wieder auf 0 setzen und die nächste Zeile holen. Folgendes BASIC-Programm demonstriert diesen Effekt:

```
100 SCNCLR
110 FOR I=0 TO 24
120 : GOSUB 240
130 NEXT I
140 SR=65286
150 FOR T=1 TO 20 : NEXT T
160 POKE SR,(PEEK(SR)AND240)OR 7
170 GOSUB 240
180 FOR I=6 TO 0 STEP -1
190 : FOR T=1 TO 60 : NEXT T
200 : POKE SR,(PEEK(SR)AND240)OR I
210 NEXT I
220 GET K$ : IF K$="" THEN 150
230 POKE SR,(PEEK(SR)AND240)OR 11 : END
240 PRINT CHR$(13)"BITTE TASTE DRUECKEN"; : RETURN
```

Hier die zeilenweise Erläuterung:

100	Bildschirm löschen
110-130	Text auf Bildschirm ausgeben
140	Setze SR auf die Adresse des Scroll-Registers (vertikal)
150	Warteschleife
160	Auf 24 Zeilen setzen und Scroll-Wert auf 7
170	Nachricht auf die momentan unsichtbare 25. Zeile drucken
180	Scroll-Wert herunterzählen
190	Warteschleife
200	Scroll-Wert setzen
210	Nächster Wert
220	Auf Tastendruck prüfen
230	Wieder auf 25 Zeilen setzen und normalen Scroll-Wert (3)
240	Unterprogramm, um Nachricht auszugeben

Im nächsten Kapitel werden Sie noch ein Beispiel für horizontales Scrolling (in Maschinensprache) finden, welches in Zusammenarbeit mit dem Raster Interrupt für schöne Effekte verantwortlich ist.

2.8 DER RASTER-INTERRUPT

Der PLUS/4 verfügt über die Möglichkeit, den Prozessor zu unterbrechen in Abhängigkeit vom Bildschirmstrahl, der das Fernsehbild erzeugt. Eine Anwendung dafür besteht darin, Daten auf dem Bildschirm zu schreiben, während der Rasterstrahl sich im nicht sichtbaren Bereich befindet. Dadurch wird ein Flickern vermieden. Ferner ist es möglich, den Bildschirm in verschiedene Arbeitsbereiche zu unterteilen. Dies wird auch vom BASIC des PLUS/4 genutzt, um den geteilten Bildschirm in den Grafik-Modi 2 und 4 zu realisieren.

Um einen Raster-Interrupt zu erzeugen, muß man Bit 1 des Interrupt Enable Registers ($FF0A) auf 1 setzen. Die unteren 8 Bits der Zeile, in der ein Interrupt ausgelöst werden soll, muß ins Register $FF0B geschrieben werden und das 9. Bit nach Bit 0 des Registers $FF0A. Sobald der Raster-Strahl jetzt in der gewünschten Zeile ankommt, wird ein Raster-Interrupt ausgelöst und Bit 1 des Raster Interrupt Status Registers ($FF09) wird auf 1 gesetzt. Am einfachsten schauen Sie sich das nächste Beispiel an, welches ein schwarz gefärbtes Band im Bildschirm-Hintergrund erzeugt.

```
>1000 00 0C 10 00 00 9E 34 31 :.......41
>1008 31 32 00 00 00 00 00 00 :12......
>1010 78 A9 29 8D 14 03 A9 10 :x))...).
>1018 8D 15 03 A9 02 8D 0A FF :...)..
>1020 A9 30 8D 0B FF 58 4C 26 :)0.XL&
>1028 10 AD 09 FF 8D 09 FF AD :.-.-
>1030 0B FF C9 50 90 0B A9 30 :IP..)0
>1038 8D 0B FF EE 15 FF 4C 49 :.nLI
>1040 10 A9 50 8D 0B FF CE 15 :.)P.N.
>1048 FF 68 A8 68 AA 68 40 AA :h(h*h@*

S"RASTER-DEMO",01,1001,104F

. 1010  78       SEI
. 1011  A9 29    LDA #$29
. 1013  8D 14 03 STA $0314
. 1016  A9 10    LDA #$10
. 1018  8D 15 03 STA $0315
```

```
. 101B  A9 02    LDA #$02
. 101D  8D 0A FF STA $FF0A
. 1020  A9 30    LDA #$30
. 1022  8D 0B FF STA $FF0B
. 1025  58       CLI
. 1026  4C 26 10 JMP $1026
. 1029  AD 09 FF LDA $FF09
. 102C  8D 09 FF STA $FF09
. 102F  AD 0B FF LDA $FF0B
. 1032  C9 50    CMP #$50
. 1034  90 0B    BCC $1041
. 1036  A9 30    LDA #$30
. 1038  8D 0B FF STA $FF0B
. 103B  EE 15 FF INC $FF15
. 103E  4C 49 10 JMP $1049
. 1041  A9 50    LDA #$50
. 1043  8D 0B FF STA $FF0B
. 1046  CE 15 FF DEC $FF15
. 1049  68       PLA
. 104A  A8       TAY
. 104B  68       PLA
. 104C  AA       TAX
. 104D  68       PLA
. 104E  40       RTI
```

Abschließend wollen wir Ihnen noch ein Programm zeigen, welches die Grafik-Möglichkeiten des TED mit Softscrolling und Raster Interrupts eindrucksvoll demonstriert. Die Kombination dieser beiden Eigenschaften wird in vielen kommerziellen Spielen genutzt, um z.B. eine Landschaft an Ihnen vorbei ziehen zu lassen. In unserem Beispiel werden wir 3 Texte auf den Bildschirm bringen, von denen der mittlere mit dem Joystick weich über den Bildschirm verschoben werden kann - in beide Richtungen mit variabler Geschwindigkeit. Wenn Sie dieses Beispiel analysiert und verstanden haben, so dürfte es für Sie keine größere Schwierigkeit mehr sein, tolle eigene Grafik-Effekte und -Programme zu schreiben.

```
>1000 00 0B 10 00 00 9E 34 31 :......41
>1008 31 32 00 00 00 00 00 00 :12......
>1010 A9 CF 8D 15 FF A2 00 A9 :)O.".)
>1018 20 9D 00 0C 9D 00 0D 9D : .......
>1020 00 0E 9D 00 0F A9 00 9D :......)..
>1028 00 08 9D 00 09 9D 00 0A :........
>1030 9D 00 0B CA D0 E1 A0 12 :...JPa .
>1038 B9 9F 11 99 82 0C 99 2A :9......*
>1040 0F B9 B2 11 99 D2 0C 99 :.92..R..
>1048 7A 0F B9 C5 11 99 C2 0D :z.9E..B.
>1050 B9 D8 11 99 62 0E A9 26 :9X..b.)&
>1058 99 C2 09 99 62 0A 88 10 :.B..b...
>1060 D7 86 D0 86 DB 20 8F 10 :W.P.[ ..
>1068 20 B1 10 A9 FD 8D 08 FF : 1.)}.
>1070 AD 08 FF A2 00 4A 4A 4A :-".JJJ
>1078 B0 01 CA 4A B0 01 E8 8A :0.JJ0.h.
>1080 18 65 D0 C9 F7 F0 E1 C9 :.ePIwpaI
>1088 09 F0 DD 85 D0 D0 D9 78 :.pJ.PPYx
>1090 A9 73 8D 14 03 A9 11 8D :)s...)..
>1098 15 03 A9 02 8D 0A FF A9 :..)..)
>10A0 00 85 DA 85 D9 A9 5A 8D :..Z.Y)Z.
>10A8 0B FF A9 00 8D 07 FF 58 :)..X
>10B0 60 A5 DB F0 FC A9 00 85 :`%[p!)..
>10B8 DB A5 D9 18 65 D0 A8 29 :[%Y.eP()
>10C0 07 85 D9 98 29 08 D0 01 :..Y.).P.
>10C8 60 A2 05 A9 0D 85 D2 85 :`".)..R.
>10D0 D4 A9 09 85 D6 85 D8 A9 :T)..V.X)
>10D8 90 85 D1 98 10 31 A0 FF :..Q..1
>10E0 A5 D1 18 69 28 85 D1 85 :%Q.i(.Q.
>10E8 D3 85 D5 85 D7 90 08 E6 :S.U.W..f
>10F0 D2 E6 D4 E6 D6 E6 D8 E6 :RfTfVfXf
>10F8 D3 E6 D7 C8 B1 D3 91 D1 :SfWH1S.Q
>1100 B1 D7 91 D5 C0 27 D0 F3 :1W.U@'Ps
>1108 CA D0 D3 84 DA F0 2D A0 :JPS.Zp-
>1110 26 A5 D1 18 69 28 85 D1 :&%Q.i(.Q
>1118 85 D3 85 D5 85 D7 90 08 :.S.U.W..
>1120 E6 D2 E6 D4 E6 D6 E6 D8 :fRfTfVfX
>1128 E6 D3 E6 D7 B1 D1 91 D3 :fSfW1Q.S
>1130 B1 D5 91 D7 88 10 F5 CA :1U.W..uJ
>1138 D0 D5 86 DA A2 05 A4 DA :PU.Z".$Z
>1140 A9 0D 85 D2 A9 09 85 D6 :)..R)..V
```

```
>1148 A9 90 85 D1 85 D5 A5 D1 :)..Q.U%Q
>1150 18 69 28 85 D1 85 D5 90 :.i(.Q.U.
>1158 04 E6 D2 E6 D6 98 49 27 :.fRfV.I'
>1160 A8 B1 D1 85 DC B1 D5 A4 :(1Q.\1U$
>1168 DA 91 D5 A5 DC 91 D1 CA :Z.U%\.QJ
>1170 D0 DC 60 AD 09 FF 8D 09 :P\`-..
>1178 FF AE 0B FF A9 40 E0 83 :.)@`.
>1180 F0 08 05 D9 A2 83 86 DB :p..Y"..[
>1188 D0 02 A2 5A 8E 0B FF 29 :P."Z.)
>1190 07 EC 1D FF F0 FB 8D 07 :.lp{..
>1198 FF 68 A8 68 AA 68 40 20 :h(h*h@
>11A0 04 09 05 13 05 12 20 14 :..... .
>11A8 05 09 0C 20 08 09 05 12 :... ....
>11B0 20 20 20 20 20 02 0C 05 :      ...
>11B8 09 02 14 20 13 14 05 08 :... ....
>11C0 05 0E 20 20 20 16 05 12 :..   ...
>11C8 13 03 08 09 05 02 05 20 :.......
>11D0 04 01 13 20 08 09 05 12 :... ....
>11D8 0D 09 14 20 05 09 0E 05 :... ....
>11E0 0D 20 0A 0F 19 13 14 09 :. ......
>11E8 03 0B 21 AA AA AA AA AA :..!*****
```

```
S"HSCROLL",01,1001,11EB

. 1010  A9 CF    LDA #$CF
. 1012  8D 15 FF STA $FF15
. 1015  A2 00    LDX #$00
. 1017  A9 20    LDA #$20
. 1019  9D 00 0C STA $0C00,X
. 101C  9D 00 0D STA $0D00,X
. 101F  9D 00 0E STA $0E00,X
. 1022  9D 00 0F STA $0F00,X
. 1025  A9 00    LDA #$00
. 1027  9D 00 08 STA $0800,X
. 102A  9D 00 09 STA $0900,X
. 102D  9D 00 0A STA $0A00,X
. 1030  9D 00 0B STA $0B00,X
. 1033  CA       DEX
. 1034  D0 E1    BNE $1017
. 1036  A0 12    LDY #$12
. 1038  B9 9F 11 LDA $119F,Y
. 103B  99 82 0C STA $0C82,Y
```

```
. 103E  99 2A 0F STA $0F2A,Y
. 1041  B9 B2 11 LDA $11B2,Y
. 1044  99 D2 0C STA $0CD2,Y
. 1047  99 7A 0F STA $0F7A,Y
. 104A  B9 C5 11 LDA $11C5,Y
. 104D  99 C2 0D STA $0DC2,Y
. 1050  B9 D8 11 LDA $11D8,Y
. 1053  99 62 0E STA $0E62,Y
. 1056  A9 26    LDA #$26
. 1058  99 C2 09 STA $09C2,Y
. 105B  99 62 0A STA $0A62,Y
. 105E  88       DEY
. 105F  10 D7    BPL $1038
. 1061  86 D0    STX $D0
. 1063  86 DB    STX $DB
. 1065  20 8F 10 JSR $108F
. 1068  20 B1 10 JSR $10B1
. 106B  A9 FD    LDA #$FD
. 106D  8D 08 FF STA $FF08
. 1070  AD 08 FF LDA $FF08
. 1073  A2 00    LDX #$00
. 1075  4A       LSR
. 1076  4A       LSR
. 1077  4A       LSR
. 1078  B0 01    BCS $107B
. 107A  CA       DEX
. 107B  4A       LSR
. 107C  B0 01    BCS $107F
. 107E  E8       INX
. 107F  8A       TXA
. 1080  18       CLC
. 1081  65 D0    ADC $D0
. 1083  C9 F7    CMP #$F7
. 1085  F0 E1    BEQ $1068
. 1087  C9 09    CMP #$09
. 1089  F0 DD    BEQ $1068
. 108B  85 D0    STA $D0
. 108D  D0 D9    BNE $1068
. 108F  78       SEI
. 1090  A9 73    LDA #$73
. 1092  8D 14 03 STA $0314
```

```
. 1095  A9 11     LDA #$11
. 1097  8D 15 03 STA $0315
. 109A  A9 02     LDA #$02
. 109C  8D 0A FF STA $FF0A
. 109F  A9 00     LDA #$00
. 10A1  85 DA     STA $DA
. 10A3  85 D9     STA $D9
. 10A5  A9 5A     LDA #$5A
. 10A7  8D 0B FF STA $FF0B
. 10AA  A9 00     LDA #$00
. 10AC  8D 07 FF STA $FF07
. 10AF  58        CLI
. 10B0  60        RTS
. 10B1  A5 DB     LDA $DB
. 10B3  F0 FC     BEQ $10B1
. 10B5  A9 00     LDA #$00
. 10B7  85 DB     STA $DB
. 10B9  A5 D9     LDA $D9
. 10BB  18        CLC
. 10BC  65 D0     ADC $D0
. 10BE  A8        TAY
. 10BF  29 07     AND #$07
. 10C1  85 D9     STA $D9
. 10C3  98        TYA
. 10C4  29 08     AND #$08
. 10C6  D0 01     BNE $10C9
. 10C8  60        RTS
. 10C9  A2 05     LDX #$05
. 10CB  A9 0D     LDA #$0D
. 10CD  85 D2     STA $D2
. 10CF  85 D4     STA $D4
. 10D1  A9 09     LDA #$09
. 10D3  85 D6     STA $D6
. 10D5  85 D8     STA $D8
. 10D7  A9 90     LDA #$90
. 10D9  85 D1     STA $D1
. 10DB  98        TYA
. 10DC  10 31     BPL $110F
. 10DE  A0 FF     LDY #$FF
. 10E0  A5 D1     LDA $D1
. 10E2  18        CLC
```

```
. 10E3  69 28    ADC #$28
. 10E5  85 D1    STA $D1
. 10E7  85 D3    STA $D3
. 10E9  85 D5    STA $D5
. 10EB  85 D7    STA $D7
. 10ED  90 08    BCC $10F7
. 10EF  E6 D2    INC $D2
. 10F1  E6 D4    INC $D4
. 10F3  E6 D6    INC $D6
. 10F5  E6 D8    INC $D8
. 10F7  E6 D3    INC $D3
. 10F9  E6 D7    INC $D7
. 10FB  C8       INY
. 10FC  B1 D3    LDA ($D3),Y
. 10FE  91 D1    STA ($D1),Y
. 1100  B1 D7    LDA ($D7),Y
. 1102  91 D5    STA ($D5),Y
. 1104  C0 27    CPY #$27
. 1106  D0 F3    BNE $10FB
. 1108  CA       DEX
. 1109  D0 D3    BNE $10DE
. 110B  84 DA    STY $DA
. 110D  F0 2D    BEQ $113C
. 110F  A0 26    LDY #$26
. 1111  A5 D1    LDA $D1
. 1113  18       CLC
. 1114  69 28    ADC #$28
. 1116  85 D1    STA $D1
. 1118  85 D3    STA $D3
. 111A  85 D5    STA $D5
. 111C  85 D7    STA $D7
. 111E  90 08    BCC $1128
. 1120  E6 D2    INC $D2
. 1122  E6 D4    INC $D4
. 1124  E6 D6    INC $D6
. 1126  E6 D8    INC $D8
. 1128  E6 D3    INC $D3
. 112A  E6 D7    INC $D7
. 112C  B1 D1    LDA ($D1),Y
. 112E  91 D3    STA ($D3),Y
. 1130  B1 D5    LDA ($D5),Y
```

```
. 1132  91 D7     STA ($D7),Y
. 1134  88        DEY
. 1135  10 F5     BPL $112C
. 1137  CA        DEX
. 1138  D0 D5     BNE $110F
. 113A  86 DA     STX $DA
. 113C  A2 05     LDX #$05
. 113E  A4 DA     LDY $DA
. 1140  A9 0D     LDA #$0D
. 1142  85 D2     STA $D2
. 1144  A9 09     LDA #$09
. 1146  85 D6     STA $D6
. 1148  A9 90     LDA #$90
. 114A  85 D1     STA $D1
. 114C  85 D5     STA $D5
. 114E  A5 D1     LDA $D1
. 1150  18        CLC
. 1151  69 28     ADC #$28
. 1153  85 D1     STA $D1
. 1155  85 D5     STA $D5
. 1157  90 04     BCC $115D
. 1159  E6 D2     INC $D2
. 1158  E6 D6     INC $D6
. 115D  98        TYA
. 115E  49 27     EOR #$27
. 1160  A8        TAY
. 1161  B1 D1     LDA ($D1),Y
. 1163  85 DC     STA $DC
. 1165  B1 D5     LDA ($D5),Y
. 1167  A4 DA     LDY $DA
. 1169  91 D5     STA ($D5),Y
. 116B  A5 DC     LDA $DC
. 116D  91 D1     STA ($D1),Y
. 116F  CA        DEX
. 1170  D0 DC     BNE $114E
. 1172  60        RTS
. 1173  AD 09 FF  LDA $FF09
. 1176  8D 09 FF  STA $FF09
. 1179  AE 0B FF  LDX $FF0B
. 117C  A9 40     LDA #$40
. 117E  E0 83     CPX #$83
```

```
. 1180  F0 08     BEQ $118A
. 1182  05 D9     ORA $D9
. 1184  A2 83     LDX #$83
. 1186  86 DB     STX $DB
. 1188  D0 02     BNE $118C
. 118A  A2 5A     LDX #$5A
. 118C  8E 0B FF STX $FF0B
. 118F  29 07     AND #$07
. 1191  EC 1D FF CPX $FF1D
. 1194  F0 FB     BEQ $1191
. 1196  8D 07 FF STA $FF07
. 1199  68        PLA
. 119A  A8        TAY
. 119B  68        PLA
. 119C  AA        TAX
. 119D  68        PLA
. 119E  40        RTI

>119F 20 04 09 05 13 05 12 20 : ......
>11A7 14 05 09 0C 20 08 09 05 :.... ...
>11AF 12 20 20 20 20 20 02 0C :.      ..
>11B7 05 09 02 14 20 13 14 05 :.... ...
>11BF 08 05 0E 20 20 20 16 05 :...    ..
>11C7 12 13 03 08 09 05 02 05 :........
>11CF 20 04 01 13 20 08 09 05 : ... ...
>11D7 12 0D 09 14 20 05 09 0E :.... ...
>11DF 05 0D 20 0A 0F 19 13 14 :.. .....
>11E7 09 03 0B 21 AA AA AA AA :...!****
```

2.9 TECHNISCHE DETAILS ZUM TED-CHIP

In allen verschiedenen Modi stellt der TED-Chip 25 Zeilen a 40 Zeichen dar. Jedes Zeichen auf dem Bildschirm kann 16 verschiedene Farben in 8 verschiedenen Helligkeitsstufen annehmen. Außerdem können die Zeichen noch blinkend dargestellt werden.

Die Werte im Video-RAM legen jeweils fest, welches Zeichen an einer bestimmten Stelle dargestellt werden soll. Dazu gehört immer ein entsprechendes **Attribut**, welches für Farbe, Helligkeit und Blinken sorgt.

Der TED-Chip holt sich diese Informationen aus dem Video-RAM bzw. Farb-RAM. Das Video-RAM besteht aus 1000 Speicherstellen, die jeweils aus einem 8 Bit Zeiger bestehen (siehe auch Kapitel 2.1). Wo sich das Video-RAM im Speicherplan des PLUS/4 befindet, wird durch das **Video Matrix Register** im TED-Chip festgelegt. Die Register des TED-Chip liegen im Speicherplan des PLUS/4 ab $FF00 (dezimal: 65280). Das Video Matrix Register ist das TED-Register 20, die gesamte Adresse lautet also: 65280+20=65300 (hex: $FF00+$14=$FF14). Wenn wir die Lage des Video-RAMs im Speicher des PLUS/4 verändern wollen, so müssen wir also dieses Register manipulieren. Das TED-Register 20 hat folgenden Aufbau:

Bit:	7	6	5	4	3	2	1	0
Funktion:	VM4	VM3	VM2	VM1	VM0	NB	NB	NB

NB bedeutet "Nicht Belegt" und kann ignoriert werden, VM4-VM1 bestimmen nun die Lage des Video-RAMs im Speicher, siehe nachfolgende Tabelle:

VM4-VM1	Video-RAM	VM4-VM1	Video-RAM
00000	$0400	10000	$8400
00001	$0C00 (Standard)	10001	$8C00
00010	$1400	10010	$9400
00011	$1C00	10011	$9C00
00100	$2400	10100	$A400
00101	$2C00	10101	$AC00
00110	$3400	10110	$B400

00111	$3C00	10111	$BC00
01000	$4400	11000	$C400
01001	$4C00	11001	$CC00
01010	$5400	11010	$D400
01011	$5C00	11011	$DC00
01100	$6400	11100	$E400
01101	$6C00	11101	$EC00
01110	$7400	11110	$F400
01111	$7C00	11111	$FC00

Wenn Sie einen PLUS/4 besitzen, so können Sie Ihr Video-RAM prinzipiell an 8 verschiedene Stellen legen ($0400 ... $3C00). Sie müssen dabei allerdings beachten, daß das Farb-RAM stets genau 1K unterhalb des Video-RAMs installiert wird. Wenn Sie also (wie nach dem Einschalten) das Video-RAM bei $0C00 liegen haben, so wird das Farb-RAM bei $0800 landen.

Jede Speicherstelle in der Videomatrix wird als Zeiger auf die Zeichendefinition des entsprechenden Zeichens gebraucht. Das 8. Bit (MSB) dieses Zeigers kann auf zwei verschiedene Arten interpretiert werden. Wenn das RVS-Bit (Bit 7) des TED-Registers 7 ($FF07) 0 ist, wird das MSB der Videomatrix (VM7) benutzt, um zu entscheiden, ob das Zeichen revers oder normal dargestellt wird. Wenn VM7 gleich 0 ist, wird es normal dargestellt, wenn es 1 ist, wird es revers dargestellt, wobei der TED-Chip hardwaremäßig die Revers-Darstellung erzeugt. Dadurch sind dann 128 Zeichen frei definierbar. Wenn das RVS-Bit vom TED-Chip gleich 1 ist, so können 256 eigene Zeichen definiert werden.

VIDEO MATRIX ADRESSE

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
VM4	VM3	VM2	VM1	VM0	1	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Der Attribut-Speicher besteht ebenso aus 1000 aufeinanderfolgenden Speicherstellen und enthält das Blink-Bit (Bit 7), die Helligkeit (Bit 4-6) und die Farbe (Bit 0-3). Die Lage des Attribut- oder Farb-RAMs wird auch durch das Video Matrix Register bestimmt.

Da aber hier A10 stets gleich 0 ist, liegt das Attribut-RAM immer genau 1K unterhalb des Video-RAMs.

ATTRIBUT ADRESSE

```
A15 A14 A13 A12 A11 A10 A9  A8  A7  A6  A5  A4  A3  A2  A1  A0
-----------------------------------------------------------------
VM4 VM3 VM2 VM1 VM0  0  VC9 VC8 VC7 VC6 VC5 VC4 VC3 VC2 VC1 VC0
```

Jedes Zeichen besteht aus einer Matrix von 8x8 Punkten, die im ROM als 8 aufeinanderfolgende Bytes abgespeichert sind. Die Adresse dieses Zeichenspeichers wird durch CB4 bis CB0 des TED-Registers 19 ($FF13) bestimmt. Diese Bits sind die höherwertigen Bits der Zeichensatz-Adresse.

```
A15 A14 A13 A12 A11 A10 A9  A8  A7  A6  A5  A4  A3  A2  A1  A0
-----------------------------------------------------------------
CB5 CB4 CB3 CB2 CB1 VM7 VM6 VM5 VM4 VM3 VM2 VM1 VM0
                    CB0 (wenn RVS-Bit auf 1 ist)
```

3.1 MUSIK MIT DEN BASIC-BEFEHLEN

Der Commodore PLUS/4 hat einfache, aber gute Möglichkeiten der Sound-Erzeugung. Er verfügt über zwei unabhängige Tonkanäle, die sehr einfach mit den entsprechenden BASIC-Befehlen VOL (Lautstärke) und SOUND (Stimme, Frequenz, Dauer) programmiert werden können. Wir wollen nun nicht näher auf die genauen Details dieser BASIC-Befehle eingehen (dazu ist schließlich das Handbuch da), sondern wir wollen Ihnen an einem kleinen Beispiel zeigen, wie man kleine zweistimmige Musikstücke in BASIC programmieren kann.

```
100 VOL 8
110 DIM N1%(66),N2%(66),D1%(66),D2%(66)
120 I=0
130 READ N1%(I),D1%(I) : IF N1%(I)<0 THEN 150
140 I=I+1 : GOTO 130
150 T1=I : I=0
160 READ N2%(I),D2%(I) : IF N2%(I)<0 THEN 180
170 I=I+1 : GOTO 160
180 I1=-1 : I2=-1
190 IF D1>0 THEN 220 : ELSE SOUND 1,N1,0
200 I1=I1+1 : IF I1<T1 THEN D1=D1%(I1) : N1=N1%(I1) : ELSE 270
210 IF N1>0 THEN SOUND 1,N1,300
220 IF D2>0 THEN 250 : ELSE SOUND 2,N2,0
230 I2=I2+1 : D2=D2%(I2) : N2=N2%(I2)
240 IF N2>0 THEN SOUND 2,N2,300
250 D1=D1-1 : D2=D2-1
260 FOR I=1 TO 80 : NEXT I : GOTO 190
270 VOL 0
280 DATA   0,1,685,1,770,1,810,1
290 DATA 798,1,685,1,798,1,834,1
300 DATA 810,2,854,2,755,2,854,2
310 DATA 770,1,685,1,770,1,810,1
320 DATA 798,1,685,1,798,1,834,1
330 DATA 810,2,770,2,  0,4
340 DATA   0,1,854,1,810,1,854,1
350 DATA 770,1,810,1,685,1,739,1
360 DATA 704,2,770,2,834,2,864,2
370 DATA 864,1,834,1,798,1,834,1
```

```
380 DATA 739,1,798,1,643,1,704,1
390 DATA 685,2,739,2,810,2,254,2
400 DATA 854,1,810,1,770,1,810,1
410 DATA 704,2,834,2,834,1,798,1
420 DATA 739,1,798,1,685,2,810,2
430 DATA 810,1,770,1,704,1,770,1
440 DATA 643,2,798,2,810,6
450 DATA -1,-1
460 DATA 7,2,516,4,485,2
470 DATA 516,1,345,1,516,1,596,1
480 DATA 571,1,345,1,571,1,643,1
490 DATA 596,2,516,2,485,2,345,2
500 DATA 516,1,345,1,516,1,596,1
510 DATA 571,1,345,1,571,1,643,1
520 DATA 596,2,516,2,596,2,516,2
530 DATA 643,1,516,1,383,1,516,1
540 DATA 262,1,383,1,  7,1,169,1
550 DATA 118,2,262,2,453,2,571,2
560 DATA 571,1,453,1,345,1,453,1
570 DATA 169,1,345,1,118,1,118,1
580 DATA   7,2,169,2,262,2,383,1
590 DATA 118,1,262,1,118,2,118,2
600 DATA 169,1,345,1,  7,1,169,1
610 DATA   7,2,  7,2
620 DATA 118,1,453,1,383,1,453,1
630 DATA 596,6
640 DATA -1,-1
```

Hier wieder die zeilenweise Erklärung des Programms:

100 Setze Lautstärke auf Maximum.

110 Datenfelder dimensionieren.

120 I zählt die Anzahl der Noten.

130 Lese Note und Dauer für Stimme 1 ein; eine negative Zahl bedeutet: Fertig!

140 Erhöhe Zähler I und mach weiter.

150 I1 ist die Gesamtzahl der Noten für Stimme 1. Setze I

wieder auf 0.

160 Lese Note und Dauer für Stimme 2 ein; eine negative Zahl bedeutet: Fertig!

170 Erhöhe Zähler I und mach weiter.

180 I1 und I2 sind Zeiger für die Datenfelder.

190 Wenn Stimme 1 fertig ist, gehe nach 220; sonst stoppe Stimme 1.

200 Erhöhe den Zeiger von Stimme 1. Wenn fertig, dann Schleife verlassen; sonst Note und Dauer setzen.

210 Wenn es keine Pause ist, Note starten.

220 Wenn Stimme 2 fertig ist, gehe nach 250; sonst stoppe Stimme 2.

230 Erhöhe den Zeiger von Stimme 2 und setze Note und Dauer.

240 Wenn es keine Pause ist, Note starten.

250 Erniedrige die Dauer.

260 Kurz warten. In dieser Zeile können Sie alle Noten kürzer oder länger spielen lassen.

270 Lautstärke abschalten.

280-440 Daten für Stimme 1.

450 Ende der Daten für Stimme 1.

460-630 Daten für Stimme 2.

640 Ende der Daten für Stimme 2.

3.2 DIE SOUND-REGISTER DES TED

Die Sound-Register sind in dem gleichen Chip untergebracht wie die Grafik-Register, im TED. Wegen der einfachen Sound-Möglichkeiten werden aber nur fünf Register benötigt, die folgende Funktion haben:

ADRESSE	BITS	FUNKTION
$FF0E	0-7	Low-Byte der Frequenz von Stimme 1
$FF0F	0-7	" " " " " " 2
$FF10	0-1	Obere 2 Bits der Frequenz von Stimme 1
$FF11	0-3	Lautstärke
	4	Wähle Stimme 1 (0=Aus, 1=Ein)
	5	" " 2 " " " "
	6	" Geräusch für Stimme 2 (0=Aus, 1=Ein)
	7	Sound-Schalter (0=Ein, 1=Aus)
$FF12	0-1	Obere 2 Bits der Frequenz von Stimme 2

Um einen Sound zu erzeugen, wählen Sie zuerst die Stimme oder die Stimmen aus und die Lautstärke mit dem Register $FF11. Normalerweise setzt man Bit 7 dieses Registers erstmal zu 1, um absolute Ruhe zu haben. Als nächstes setzt man die Frequenzen, wobei Sie beachten sollten, daß die Bits 2-7 des Registers $FF12 für andere Zwecke benutzt werden und nicht verändert werden dürfen. Als letztes können Sie jetzt den Sound mit Bit 7 von $FF11 anschalten und später wieder abschalten.

3.3 INTERRUPT-GESTEUERTE MUSIK

In diesem Kapitel wollen wir einmal die Möglichkeit untersuchen, Musik auf dem PLUS/4 zu erzeugen, die unabhängig vom übrigen Geschehen abläuft. Diese Art von Hintergrundmusik findet man oft in Spielen und es ist dabei sehr praktisch, sich (programmtechnisch) auf das eigentliche Spiel zu konzentrieren und sich nicht um die Musik zu kümmern. Diese automatische Hintergrundmusik läßt sich auf dem PLUS/4 erstaunlich einfach verwirklichen. Wie Sie vielleicht wissen, arbeitet der PLUS/4 alle 1/60 Sekunde einen sogenannten Interrupt ab. Dies ist eine kurze Unterbrechung des normalen Programmablaufs, in der der PLUS/4 nachsieht, ob eine Taste gedrückt wurde, in der die interne Uhr weitergeschaltet wird und einiges anderes. Wenn wir uns in diesen Interrupt "einhängen", so kann der PLUS/4 neben diesen obigen Tätigkeiten auch die jeweiligen Sound-Parameter aktualisieren. Am einfachsten, Sie geben das nachfolgende Programm ein und überzeugen sich selber. Eine etwas detailliertere Erklärung dieses Programms folgt im Anschluß.

```
100 FOR I=1536 TO 1536+173
110 : READ A
120 : POKE I,A
130 : C=C+A
140 NEXT I
150 IF C<>23952 THEN PRINT "PRUEFSUMMENFEHLER" : STOP
160 FOR I=1712 TO 1712+119 STEP 3
170 : READ A,B
180 : POKE I,A : POKE I+1,B-(INT(B/256)*256)
190 : POKE I+2,INT(B/256)
200 : D=D+A+B
210 NEXT I
220 IF D<>34808 THEN PRINT "PRUEFSUMMENFEHLER" : STOP
230 POKE 208,176 : POKE 209,6
240 VOL 7 : SYS 1536
250 :
260 DATA 120,169,36,141,20,3,169,6,141,21,3,169,255,141,252,4
270 DATA 141,254,4,88,96,120,169,14,141,20,3,169,206,141,21,3
280 DATA 88,96,255,0,173,252,4,201,255,240,3,76,14,206,160,0
290 DATA 162,0,177,208,149,210,200,232,138,201,3,208,245,165,208,105
```

```
300 DATA 2,133,208,144,2,230,209,165,210,201,255,208,6,32,21,6
310 DATA 24,144,216,201,254,208,19,165,208,133,213,165,211,133,208,165
320 DATA 209,133,214,165,212,133,209,24,144,196,201,253,208,11,165,213
330 DATA 133,208,165,214,133,209,24,144,181,201,252,208,13,173,17,255
340 DATA 41,240,5,211,141,17,255,24,144,164,73,255,141,252,4,169
350 DATA 255,141,254,4,173,18,255,41,252,5,212,141,18,255,165,211
360 DATA 141,14,255,173,17,255,9,16,141,17,255,24,144,162
370 :
380 DATA 254,1727,254,1796,254,1796,254,1727,254,1712
390 DATA 20,685,20,739,40,739,20,685,20,643,40,596,20,643,20,685
400 DATA 20,739,20,685,80,643,20,685,20,739,40,739
410 DATA 20,685,20,643,40,596,20,643,20,685,20,643
420 DATA 20,596,80,596,253,0
430 DATA 20,770,20,810,40,810,20,796,20,739,40,770
440 DATA 20,770,20,810,20,798,20,739,80,770,253,0
```

Hier wieder die disassemblierte Ausgabe des eigentlichen Maschinenprogramms:

```
. 0600  78       SEI
. 0601  A9 24    LDA #$24
. 0603  8D 14 03 STA $0314
. 0606  A9 06    LDA #$06
. 0608  8D 15 03 STA $0315
. 060B  A9 FF    LDA #$FF
. 060D  8D FC 04 STA $04FC
. 0610  8D FE 04 STA $04FE
. 0613  58       CLI
. 0614  60       RTS
. 0615  78       SEI
. 0616  A9 0E    LDA #$0E
. 0618  8D 14 03 STA $0314
. 061B  A9 CE    LDA #$CE
. 061D  8D 15 03 STA $0315
. 0620  58       CLI
. 0621  60       RTS
. 0622  EA       NOP
. 0623  EA       NOP
. 0624  AD FC 04 LDA $04FC
```

```
. 0627  C9 FF     CMP #$FF
. 0629  F0 03     BEQ $062E
. 062B  4C 0E CE  JMP $CE0E
. 062E  A0 00     LDY #$00
. 0630  A2 00     LDX #$00
. 0632  B1 D0     LDA ($D0),Y
. 0634  95 D2     STA $D2,X
. 0636  C8        INY
. 0637  E8        INX
. 0638  8A        TXA
. 0639  C9 03     CMP #$03
. 063B  D0 F5     BNE $0632
. 063D  A5 D0     LDA $D0
. 063F  69 02     ADC #$02
. 0641  85 D0     STA $D0
. 0643  90 02     BCC $0647
. 0645  E6 D1     INC $D1
. 0647  A5 D2     LDA $D2
. 0649  C9 FF     CMP #$FF
. 064B  D0 06     BNE $0653
. 064D  20 15 06  JSR $0615
. 0650  18        CLC
. 0651  90 D8     BCC $062B
. 0653  C9 FE     CMP #$FE
. 0655  D0 13     BNE $066A
. 0657  A5 D0     LDA $D0
. 0659  85 D5     STA $D5
. 065B  A5 D3     LDA $D3
. 065D  85 D0     STA $D0
. 065F  A5 D1     LDA $D1
. 0661  85 D6     STA $D6
. 0663  A5 D4     LDA $D4
. 0665  85 D1     STA $D1
. 0667  18        CLC
. 0668  90 C4     BCC $062E
. 066A  C9 FD     CMP #$FD
. 066C  D0 0B     BNE $0679
. 066E  A5 D5     LDA $D5
. 0670  85 D0     STA $D0
. 0672  A5 D6     LDA $D6
. 0674  85 D1     STA $D1
```

```
. 0676  18        CLC
. 0677  90 B5     BCC $062E
. 0679  C9 FC     CMP #$FC
. 067B  D0 0D     BNE $068A
. 067D  AD 11 FF  LDA $FF11
. 0680  29 F0     AND #$F0
. 0682  05 D3     ORA $D3
. 0684  8D 11 FF  STA $FF11
. 0687  18        CLC
. 0688  90 A4     BCC $062E
. 068A  49 FF     EOR #$FF
. 068C  8D FC 04  STA $04FC
. 068F  A9 FF     LDA #$FF
. 0691  8D FE 04  STA $04FE
. 0694  AD 12 FF  LDA $FF12
. 0697  29 FC     AND #$FC
. 0699  05 D4     ORA $D4
. 069B  8D 12 FF  STA $FF12
. 069E  A5 D3     LDA $D3
. 06A0  8D 0E FF  STA $FF0E
. 06A3  AD 11 FF  LDA $FF11
. 06A6  09 10     ORA #$10
. 06A8  8D 11 FF  STA $FF11
. 06AB  18        CLC
. 06AC  90 A2     BCC $0650
```

Die Syntax unseres Interrupt-Spielers ist relativ einfach: es werden die Tonhöhe und die Länge abgespeichert. Zusätzlich haben einige Bytes noch Sonderfunktionen folgender Art:

$FC (252) Lautstärke. Hiermit können Sie die Lautstärke innerhalb eines Stückes ändern. Das Low-Byte der Frequenz enthält die neue Lautstärke von 0 bis 8. Das High-Byte muß vorhanden sein, wird aber nicht benutzt.

$FD (253) Dies ist eine Art GOSUB-Befehl. Er verzweigt zu der Adresse, die anstelle der Frequenz angegeben ist. Fer-

ner wird die alte Adresse abgespeichert, um eine Rückkehr vom Unterprogramm zu erlauben.

$FE (254) Das ist der entsprechende RETURN-Befehl, der ein Unterprogramm abschließt. Die beiden Frequenzbytes müssen vorhanden sein, werden aber nicht ausgewertet.

$FF (255) Ende der Sequenz

4.1 EINFÜHRUNG IN DIE MASCHINENSPRACHE

Wer seinen Rechner wirklich beherrschen will (wer will schon, daß es anders herum ist), der kommt um das Programmieren in Maschinensprache nicht herum. Das Schöne daran ist eigentlich die atemberaubende Geschwindigkeit und die uneingeschränkte Kontrolle über die Hardware-Möglichkeiten des Rechners.

Was den BASIC-Programmierer immer abschreckt, ist die angebliche Komplexität der Maschinensprache und die Tatsache, daß bei einem Fehler die Maschine auch mal abstürzen kann; dies richtet aber niemals bleibende Schäden an. Mit dem eingebautem Maschinensprache-Monitor kann auch ein absoluter Neuling innerhalb kurzer Zeit kleine Routinen in Maschinensprache schreiben. Diese Einführung soll mit einigen Beispielen zum Selbst-Experimentieren anregen.

WAS IST EIGENTLICH MASCHINENSPRACHE ?

Der PLUS/4 hat als Herz einen Mikro-Prozessor (7501), der eine Reihe von Befehlen ausführen kann; diese Befehle haben eine ähnliche Form wie diejenigen bei einem programmierbaren Taschenrechner (TI, HP, usw.). Der Programmierer hat u.a. folgendes zur Verfügung: ein Rechenregister (Akkumulator), 2 Index-Register (X-Register und Y-Register) und die 65536 Speicherstellen des PLUS/4 (entsprechend 64K-RAM). Allerdings haben viele Speicherstellen des PLUS/4 eine bestimmte Funktion, die man als Maschinensprache-Programmierer kennen muß; daher ist die Kenntnis eines Speicherbelegungsplans (Memory Map) unumgänglich (siehe Kapitel 4.4). Die Speicherstellen im PLUS/4 können nur Werte zwischen 0 und 255 annehmen (versuchen Sie mal von BASIC aus: POKE 82,256 o.ä.). Eine Zahl in einer Speicherstelle kann einen Maschinensprachebefehl darstellen, welcher vom Prozessor verstanden und ausgeführt wird, oder irgendwelche Daten; dies muß der Benutzer entscheiden. Welche Zahl welchen Maschinensprache-Befehl darstellt, muß man sich nicht merken, das erledigt der eingebaute Mini-Assembler im Maschinensprache-Monitor des PLUS/4. Man muß sich lediglich die sogenannten Mnemonics merken, dies sind Eselsbrücken mit drei Buchstaben für die einzelnen Maschinen-

Befehle. Der Assembler generiert daraus dann die Werte zwischen 0 und 255 und legt sie im Speicher ab.

4.2 BEISPIELE FÜR DIE WICHTIGSTEN BEFEHLE

1. DIE BEFEHLE LDA UND STA, DAS ERSTE PROGRAMM

Nehmen wir an, die Rahmenfarbe des PLUS/4 soll auf schwarz geändert werden. IN BASIC könnte man entweder den COLOR-Befehl benutzen oder mit einem POKE-Befehl den Wert direkt in das Hintergrund-Register des Videochip übertragen. Die BASIC-Zeile hieße also:

```
1 COLOR 4,1,0
```

oder:

```
1 POKE 65305,0
```

Wir wollen nun die letzte Zeile statt in BASIC einmal in Maschinensprache programmieren. Als erstes muß man dabei wissen, wo das Maschinen-Programm laufen soll (z.B. Aufruf mit: SYS 818). Der Kassetten-Buffer (er liegt bei $0332-03F2 bzw. 818-1010) ist für unsere kleinen Demos gut geeignet, denn er wird nur während einer Kassetten-Operation benutzt.

Will man in Maschinensprache die 0 in die Speicherstelle 65305 poken, so muß man sich hierfür zunächst einmal die 0 beschaffen. Dafür kann man den Befehl LDA benutzen, was soviel heißen soll wie **LoaD Akkumulator**, also lade das Rechenregister. Der Wert, womit das Rechenregister geladen werden soll, steht unmittelbar dahinter im Speicher, in diesem Fall die 0. In Maschinensprache wird dies durch das #-Zeichen dargestellt, gefolgt von der gewünschten Zahl. Um dem Mini-Assembler mitzuteilen, daß es sich um eine hexadezimale Zahl handelt (er kann nämlich keine anderen verarbeiten), muß man jeder Zahl immer ein **$** voranstellen und die Zahl muß zweistellig sein. Der komplette Befehl lautet nun: **LDA #$00**. Nun hat man den Wert im Rechenregister und muß ihn in die Speicherstelle 65305 (hex. $FF19) bringen. Dafür kann man den Befehl STA benutzen (**STore Akkumulator**). Dahinter muß die gewünschte Adresse folgen, also $FF19: **STA $FF19**. Als letztes muß der Rücksprung ins BASIC erfolgen. Dies geschieht mit dem Befehl RTS (**ReTurn Subroutine**): **RTS**.

Um dieses kleine Programm (und die folgenden Demos) in den PLUS/4 einzugeben, gehen Sie mit dem BASIC-Befehl: **MONITOR** in den Maschinensprache-Monitor und tippen Sie dann nur das Fettgedruckte ein (jede Zeile natürlich mit der RETURN-Taste abschließen!); die normal gedruckten Zeichen werden vom Computer ausgegeben:

```
MONITOR   (RETURN)

MONITOR

   PC  SR AC XR YR SP
; 3AB3 00 00 00 00 F9
A 0332 LDA #$00     (RETURN)
A 0334 STA $FF19    (RETURN)
A 0337 RTS          (RETURN)
A 0338              (RETURN)
X                   (RETURN)
READY.
```

Da unser Programm mit dem RTS-Befehl endet, können wir den Mini-Assembler verlassen. Wenn dieser sich mit: A 0338 meldet, drücken Sie einfach RETURN und verlassen Sie den Maschinensprache-Monitor mit: **X**.

Sie können Ihr erstes Maschinensprache-Programm jetzt mit dem Befehl: SYS 818 starten. Wer den Mut jetzt noch nicht verloren hat, der kann sich im nächsten Abschnitt die Programmierung einer Schleife ansehen.

2. EINE EWIGE SCHLEIFE (INC, JMP)

Will man die Rahmenfarbe kontinuierlich ändern, so würde man in BASIC folgendes Programm schreiben:

```
1 FOR I=0 TO 127
2 : POKE 65305,I
3 NEXT I
4 GOTO 1
```

Um die Rahmenfarbe zu ändern, kann man den Inhalt von Speicherstelle 65305 immer um eins erhöhen. Dafür kann man den Befehl: INC adresse (**INC**rement) benutzen. Dieser bewirkt weiter nichts, als den Inhalt von Adresse um eins zu erhöhen. Wenn die Speicherstelle dabei überläuft (>255), wird automatisch wieder bei Null angefangen. Unser erster Befehl lautet damit: **INC $FF19**. Da wir den Inhalt von $FF19 immer wieder um eins erhöhen wollen, müssen wir wieder zurück zu diesem Befehl springen. Dafür kann man den Befehl JMP adresse (**JuMP**) benutzen. Da unser Programm wieder bei $0332 beginnen soll, lautet die zweite (und letzte) Zeile: **JMP $0332**. Gehen Sie also mit: MONITOR in den Monitor und geben Sie folgendes ein:

```
A 0332 INC $FF19
A 0335 JMP $0332
```

Mit: SYS 818 erlebt man jetzt den ersten Effekt, der in BASIC nicht zu realisieren ist.

Da kein Programm-Ende existiert und nicht mit: RTS ins BASIC zurückgesprungen wird, muß man die RESET-Taste bemühen.

3. SCHLEIFE MIT ENTSCHEIDUNG (LDX, DEX, BNE, NOP)

Das Beispiel aus Abschnitt 2 soll nun um eine Warteschleife erweitert werden. Um eine Zeitschleife zu programmieren, läßt man den Rechner am einfachsten etwas sinnloses tun, z.B. einen Zähler auf Null bringen und bei Erreichen derselben etwas neues anzufangen. Hierbei muß das Programm also erstens einen Zähler haben und zweitens eine Entscheidung darüber treffen, ob der Zähler schon auf Null ist.

Zu diesem Zweck bietet sich eines der beiden Index-Register (X oder Y an. Der Inhalt kann zwischen 0 und 255 (hex $FF) betragen.

Unser Programm soll wieder mit dem Befehl: **INC $FF19** beginnen, es soll also die Rahmenfarbe um eins erhöhen. Als nächstes wird das Index-Register, das wir herunterzählen wollen, mit dem Startwert geladen; das geht analog zum LDA-Befehl mit: LDX adresse; in unserem Fall: **LDX #$B1**, wobei LDX **LoaD X**register bedeutet. Das Herunterzählen des X-Registers um eins ist einfach und geschieht mit dem Befehl DEX (**DE**crement **X**register). Nach dem Herunterzählen müssen wir überprüfen, ob das X-Register bei Null angelangt ist. Dazu benutzen wir den Befehl: BNE (**B**ranch **N**ot **E**qual). BNE testet, ob der vorher ausgeführte Maschinenbefehl (hier also DEX) ein bestimmtes Flag, das ZERO-Flag, gesetzt hat. Dies ist bei DEX genau dann der Fall, wenn nach der Ausführung eine 0 im X-Register steht. Ist dies nicht der Fall, so wird zu der Adresse gesprungen, die hinter dem BNE steht: **BNE $0337**; hiermit springen wir erneut zum DEX-Befehl und bilden eine Schleife.

Um eine kurze Zeit zu warten, kann man den Befehl NOP (**N**o **OP**eration) benutzen. Am Ende unseres Programms wollen wir wieder von vorne anfangen, also: **JMP $0332**. Das gesamte Programm sieht jetzt so aus:

```
A 0332 INC $FF19
A 0335 LDX #$B1
A 0337 DEX
A 0338 BNE $0337
A 033A NOP
A 033B JMP $0332
```

Probieren Sie das Programm einmal mit: SYS 818 aus. Um die Breite der Streifen zu ändern, kann man den Startwert ändern.

Anstelle des X-Registers kann auch das Y-Register verwendet werden, anstatt LDX nehme man dann LDY (LoaD Yregister) und analog dazu, anstatt DEX DEY (DEcrement Yregister).

4. BENUTZEN DES BETRIEBSSYSTEMS (JSR)

Wenn das Maschinenprogramm der Außenwelt etwas mitteilen soll, z.B. Drucker bedienen oder Nachrichten auf den Bildschirm bringen, dann läßt sich das am bequemsten mit den vorhandenen Routinen im Betriebssystem des PLUS/4 realisieren. Eine Zusammenstellung und Bedienungsanleitung dieser KERNAL-Routinen finden Sie im Kapitel 4.4 dieses Buches. Als Beispiel soll der Bildschirm von Maschinensprache aus gelöscht werden. In BASIC würde dies so aussehen:

```
1 PRINT CHR$(147)
```

Die Routine, welche ein Zeichen ab der aktuellen Cursor-Position auf den Bildschirm bringt, beginnt im Betriebssystem ab der Adresse $FFD2 (hex). Sie erwartet den ASCII-Wert des auszugebenden Zeichens im Akkumulator. Mit dem bekannten LDA-Befehl holen wir das Zeichen in den Akku und mit dem Befehl JSR (Jump SubRoutine) ruft man die Routine auf. Danach wird der nächste Maschinenbefehl abgearbeitet. Das Programm zum Löschen des Bildschirms in Maschinensprache würde dann im Monitor so eingegeben:

```
A 0332 LDA #$93
A 0334 JSR $FFD2
A 0337 RTS
```

Mit SYS 818 können Sie jetzt den Bildschirm löschen.

5. NACHRICHT DRUCKEN (TABELLE, ADRESSIERUNG)

Will man mehr als nur ein Zeichen auf dem Bildschirm ausgeben, so programmiert man am besten eine Schleife, die auf eine Tabelle zugreift. Es wird automatisch immer das nächste Zeichen aus der Tabelle geholt. Das Ende der Tabelle soll durch eine Null gekennzeichnet sein.

Der Zugriff auf die Tabelle kann mit dem Befehl LDA adresse,X erfolgen. Der Inhalt des X-Registers plus dem Wert für adresse ergibt die Speicherstelle, aus der die Zahl in den Akkumulator geladen wird. Wird eine Null (das selbst definierte Ende der Tabelle) in den Akku geladen, wird wieder ein Flag (das ZERO-Flag) gesetzt, das mit dem Befehl: BEQ adresse (**B**ranch if **EQ**ual) abgefragt werden kann. Mit DEX oder INX (**IN**crement **X**register) kann jetzt der Wert im X-Register verändert werden, um dann mit einem erneuten LDA adresse,X die nächste Adresse ansprechen zu können.

```
A 0332 LDX #$00
A 0334 LDA $0341,X
A 0337 BEQ $0340
A 0339 JSR $FFD2
A 033C INX
A 033D JMP $0334
A 0340 RTS
```

Um die Tabelle mit der gewünschten Nachricht einzugeben, tippen wir im Monitor den Befehl: **M 0341** ein, der eine hexadezimale Ausgabe der Speicherstellen ab $0341 bewirkt. Wir gehen mit dem Cursor auf die erste Speicherstelle und geben die hexadezimalen ASCII-Werte unserer Nachricht ein. Unsere Nachricht soll HALLO lauten, die entsprechenden ASCII-Werte lauten: 48 41 4C 4C 4F. Nachdem Sie diese Zahlen eingegeben haben (RETURN nicht vergessen), sollte der Bildschirm wie folgt aussehen (fett gedruckt Ihre Eingaben):

```
M0341
>0341 48 41 4C 4C 4F 00 00 00 :HALLO...
>0349 00 00 00 00 00 00 00 00 :........
>0351 00 00 00 00 00 00 00 00 :........
>0359 00 00 00 00 00 00 00 00 :........
```

```
>0361 00 00 00 00 00 00 00 00 :........
>0369 00 00 00 00 00 00 00 00 :........
>0371 00 00 00 00 00 00 00 00 :........
>0379 00 00 00 00 00 00 00 00 :........
>0381 00 00 00 00 00 00 00 00 :........
>0389 00 00 00 00 00 00 00 00 :........
>0391 00 00 00 00 00 00 00 00 :........
>0399 00 00 00 00 00 00 00 00 :........
```

Verlassen Sie den Maschinensprache-Monitor mit: **X** (RETURN), löschen Sie den Bildschirm mit der CLEAR-Taste und testen Sie Ihr Programm mit: SYS 818.

6. SPEICHERBEREICH BESCHREIBEN (INDIREKTE ADRESSIERUNG)

In unserem folgenden Beispiel wollen wir den ganzen Bildschirm mit Herzen füllen. In BASIC würde das Programm so aussehen:

```
1 FOR I=3072 TO 4071
2 : POKE I,83
3 NEXT I
```

Beachten Sie die Ausführungsgeschwindigkeit und vergleichen Sie später unser entsprechendes Maschinenprogramm. Das Beschreiben einer größeren Fläche erledigt man in Maschinensprache am elegantesten mit der indirekten Adressierung; der Zugriff auf eine bestimmte Speicherstelle erfolgt dabei mit den Befehlen:

```
LDA (zeiger),Y
STA (zeiger),Y
```

Der Inhalt von der Adresse zeiger plus dem Inhalt des Y-Registers ergibt die Adresse, aus der eine Zahl in den Akkumlator geladen (LDA) bzw. wohin sie geschrieben wird (STA).

Dabei besteht zeiger aus zwei hintereinanderliegenden Speicherstellen mit jeweils einer Adresse, die kleiner als 256 ist. Der Inhalt einer einzelnen Speicherstelle kann ja bekanntlich nur Werte zwischen 0 und 255 annehmen. Der Rechner hat aber 65535 mögliche Speicherstellen. Die gewünschte Adresse muß deshalb zuerst in zwei Anteile zerlegt werden. Der niederwertige Anteil wird dann in der ersten der beiden Speicherstellen abegelegt, der höherwertige in der darauffolgenden. Danach kann der Befehl benutzt werden, siehe nachfolgendes Programm:

```
A 0332 LDY #$00
A 0334 LDA #$00
A 0336 STA $D0
A 0338 LDA #$0C
A 033A STA $D1
A 033C LDA #$53
A 033E STA ($D0),Y
A 0340 INY
A 0341 BNE $033E
```

```
A 0343 RTS
```

Nach Aufruf mit SYS 818 werden 256 Herzen auf dem Bildschirm ausgegeben. Auf obige Weise kann nämlich nur ein 256er-Block beschrieben werden. Um den nächsten Block zu beschreiben, muß man den höherwertigen Anteil von zeiger um eins erhöhen und den Vorgang wiederholen. Da der Bildschirmspeicher 1K lang ist und 1K aus vier Blöcken besteht, muß die obige Schleife also viermal wiederholt werden. Als Zähler dafür kann man das X-Register nehmen.

Ändern Sie obiges Programm wie folgt ab:

```
A 033E LDX #$04
A 0340 STA ($D0),Y
A 0342 INY
A 0343 BNE $0340
A 0345 INC $D1
A 0347 DEX
A 0348 BNE $0340
A 034A RTS
```

Vergewissern Sie sich noch einmal, ob das Programm korrekt erstellt wurde (mit dem Monitor-Befehl: **D 0332 034A**), kehren Sie zurück ins BASIC und probieren Sie es mit SYS 818 aus. Wenn alles geklappt hat, wird Sie die Ausführungsgeschwindigkeit sicher überzeugt haben, oder? Zum Vergleich: unser BASIC-Programm benötigt für diese Aufgabe ca. 5,9 Sekunden, das Maschinenprogramm ca. 16 Millisekunden (=16/1000 Sekunden); es ist damit in diesem Fall 370mal schneller als das entsprechende BASIC-Programm!

7. UNTERPROGRAMME (JSR, RTS)

Will man einen Teil seines Maschinen-Programms von verschiedenen Stellen aus anspringen und nach Abarbeitung wieder zum Ausgangspunkt zurückspringen, so kann man hierfür den Befehl JSR adresse verwenden. Dieser wurde schon beim Ausdrucken einer Meldung auf dem Bildschirm benutzt. Der Rücksprung aus dieser angesprungenen Routine erfolgt durch den Befehl RTS (ReTurn from Subroutine). Dieser springt dann nicht ins BASIC, sondern hinter den Ausgangspunkt JSR adresse zurück.

Das Löschen des Bildschirms kann dann auch so aussehen:

```
A 0332 JSR $0336
A 0335 RTS
A 0336 LDA #$93
A 0338 JSR $FFD2
A 033B RTS
```

Mit dem gewohnten SYS 818 können Sie Ihr erstes Unterprogramm einweihen.

8. VERGLEICHS-BEFEHLE (CMP, CPX, BCC, BCS)

Oft muß der Inhalt bestimmter Speicherstellen mit Konstanten oder mit dem Inhalt anderer Speicherstellen verglichen werden. Beim Vergleich auf Null geschieht dies automatisch mit dem ZERO-Flag (siehe Abschnitt 3). Beim Vergleich mit etwas anderem muß man dagegen einen der Vergleichsbefehle CMP (CoMPare akku), CPY (ComPare Yregister) oder CPX (ComPare Xregister) benutzen.

Womit verglichen werden soll, das steht hinter dem Befehl. Mit dem #-Zeichen wird der Inhalt des entsprechenden Prozessor-Registers mit der angegebenen Zahl verglichen, ohne das #-Zeichen mit dem Inhalt der spezifizierten Adresse.

Ist dabei der Inhalt des Prozessor-Registers größer oder gleich der angegebenen Zahl oder dem Inhalt der spezifizierten Adresse, so wird ein bestimmtes Flag, das CARRY-Flag, auf 1 gesetzt. Bei Gleichheit wird zusätzlich das ZERO-Flag gesetzt, bei Ungleichheit ist es gelöscht.

Ist der Inhalt des Prozessor-Registers kleiner, so wird das CARRY-Flag auf 0 gesetzt.

Die Informationen in den Flags kann jetzt von den Verzweigungs-Befehlen genutzt werden:

BNE Branch if Not Equal
BEQ Branch if EQual
BCC Branch if Carry Clear
BCS Branch if Carry Set

Hier ein Beispielprogramm für einen Test mit anschließender Verzweigung:

```
A 0332 LDA #$F1
A 0334 CMP $FF19
A 0337 BCC $033C
A 0339 INC $FF19
A 033C RTS
```

War die Rahmenfarbe schwarz oder weiß (Inhalt von $FF19 war kleiner/gleich $F1, der Inhalt vom Akku, die $F1 war größer als der Inhalt von $FF19), so spricht die Verzweigung BCC nicht an und der Befehl INC $FF19 wird ausgeführt. Dieser bewirkt, daß die Farbe geändert wird.

War der Inhalt von $FF19 gleich $F2 oder darüber, so war der Inhalt vom Akku, die $F1, kleiner, und BCC springt nach RTS, und es passiert gar nichts.

EIN WEITERES BEISPIEL FÜR EINEN VERGLEICHSTEST:

```
A 0332 LDX $0C00
A 0335 CPX #$01
A 0337 BEQ $0340
A 0339 BCS $0341
A 033B LDA #$00
A 033D STA $FF19
A 0340 RTS
A 0341 LDA #$F1
A 0343 STA $FF19
A 0346 RTS
```

Steht in der linken oberen Bildschirmecke ein A, dann macht das Programm nichts, es springt gleich ins BASIC zurück. Zeichen mit größerem Code schalten den Rahmen auf weiß, die kleineren (der Klammeraffe) auf schwarz.

Wer bis hier angekommen ist und alle Beispiele durchprobiert oder gar ein wenig modifiziert hat, der kann sicher sein, daß ihm die Assembler-Programmierung Spaß machen wird. Eine gründlichere Einführung mit theoretischen Grundlagen (ab jetzt sollte man ernsthaft weiterlernen, viele Befehle wurden weggelassen!) findet man z.B. in der Zeitschrift 64´er ab Ausgabe 9/84 als Kurs. Weiterhin ist zu empfehlen das Buch "PROGRAMMIERUNG DES 6502" von Rodnay Zaks, erschienen im SYBEX-Verlag (der 6502 hat exakt den selben Befehlssatz wie der 7501 im PLUS/4), sowie viele andere Bücher, die sich mit dem 6502 beschäftigen. Einen kleinen Vorgeschmack auf alle Befehle des 6502/7501 finden Sie im nächsten Kapitel. Und nun: nur Mut!!!

NOCH EIN PAAR TIPS FÜR EINSTEIGER

Maschinensprache erlernt man am besten, indem man sich Maschinensprache-Programme von anderen anschaut. Solche Programme werden ständig in Zeitungen und Zeitschriften veröffentlicht. Beschäftigen Sie sich mit dem Programm, auch wenn dieses sich auf einen anderen Computer bezieht, der mit dem Mikroprozessor 6502 (oder 6510) arbeitet. Vergewissern Sie sich, ob Sie den Code verstehen. Dies erfordert Ausdauer, besonders wenn es sich um eine Ihnen noch nicht bekannte Technik handelt. Dies kann sich als äußerst mühsam erweisen, bei ausreichender Geduld gehen Sie jedoch als Sieger hervor.

Nachdem Sie andere Maschinensprache-Programme angesehen haben, müssen Sie unbedingt eigene schreiben. Hierbei kann es sich um Dienstprogramme für Ihre BASIC-Programme oder um ein reines Maschinensprache-Programm handeln. Auch hier gilt wie so oft: Übung macht den Meister.

Sie sollten auch die entweder in dem Computer oder in einem Programm verfügbaren Hilfsmittel benutzen, die Ihnen beim Schreiben, Aufbereiten sowie Überprüfen von Maschinensprache-Programmen helfen. Als Beispiel dient hier das KERNAL (siehe Kapitel 4.4), der Ihnen die Tastaturabfrage, Textanzeige, Steuerung von Peripherie-Geräten wie z.B. Kassettenrekorder, Floppy Disk, Drucker, usw., Speicherverwaltung und Bildschirmsteuerung ermöglicht. Das KERNAL ist äußerst leistungsstark, und seine Benutzung kann daher mit Nachdruck empfohlen werden.

4.3 DIE BEFEHLE DES 7501-PROZESSORS

Der im PLUS/4 eingebaute Mikroprozessor 7501 ist eine Weiterentwicklung der Serien 6502 (Atari, Apple, PET, VC-20) und 6510 (C-64). Er ist daher voll softwarekompatibel zu diesen und besitzt exakt den gleichen Befehlssatz. Dies erleichtert das Umschreiben von Programmen dieser Rechner ungemein. Zu beachten ist aber die unterschiedliche Programmierung von Ein-/Ausgabe (Floppy, Kassette), Grafik oder Sound. Wenn Sie ein Programm vom Commodore C-64 vorliegen haben, so können Sie unsere Vergleichstabelle in Kapitel 4.6 zu Rate ziehen.

Für alle Einsteiger zum Kennenlernen und für alle Fortgeschrittenen zum Nachschlagen folgt nun die alphabetische Auflistung aller Befehle des 7501 (bzw. 6502/6510). Eine ausführliche Beschreibung jedes einzelnen Befehls würde den Rahmen dieses Buches sprengen; für diesen Zweck gibt es bereits genügend Bücher in jeder Buchhandlung.

Zur Tabelle: aufgeführt werden der Befehl und eine Kurz-Beschreibung der Bedeutung, die Adressierart, das Assembler-Sprachenformat, der OP-Code, die Anzahl der Bytes und die Anzahl der Taktzyklen (ein + bedeutet, daß je nach Page-Überschreitung 1 oder 2 mehr Zyklen verbraucht werden).

BEFEHL	Beschreibung	Adressierart	Assembler-Format	OP-Code	B.	Z.
ADC	Mit Übertrag addieren	unmittelbar	ADC #Operand	69	2	2
		Zero-Page	ADC Operand	65	2	3
		Zero-Page,X	ADC Operand,X	75	2	4
		Absolut	ADC Operand	6D	3	4
		Absolut,X	ADC Operand,X	7D	3	4+
		Absolut,Y	ADC Operand,Y	79	3	4+
		(Indirekt,X)	ADC (Operand,X)	61	2	6
		(Indirekt),Y	ADC (Operand),Y	71	2	5+

AND	Logisches UND					
		unmittelbar	AND #Operand	29	2	2
		Zero-Page	AND Operand	25	2	3
		Zero-Page,X	AND Operand,X	35	2	4
		Absolut	AND Operand	2D	3	4
		Absolut,X	AND Operand,X	3D	3	4+
		Absolut,Y	AND Operand,Y	39	3	4+
		(Indirekt,X)	AND (Operand,X)	21	2	6
		(Indirekt),Y	AND (Operand),Y	31	2	5
ASL	Verschiebung					
	nach links	Akkumulator	ASL A	0A	1	2
	um 1 Bit	Zero-Page	ASL Operand	06	2	5
		Zero-Page,X	ASL Operand,X	16	2	6
		Absolut	ASL Operand	0E	3	6
		Absolut,X	ASL Operand,X	1E	3	7
BCC	Verzweigung bei					
	gelöschtem Üb.	Relativ	BCC Operand	90	2	2+
BCS	Verzweigung bei					
	gesetztem Üb.	Relativ	BCS Operand	B0	2	2+
BEQ	Verzweigung bei					
	Ergebnis = 0	Relativ	BEQ Operand	F0	2	2+
BIT	Speicherbits					
	testen	Zero-Page	BIT Operand	24	2	3
		Absolut	BIT Operand	2C	3	4
BMI	Verzweigung bei					
	Minus-Resultat	Relativ	BMI Operand	30	2	2+
BNE	Verzweigung bei					
	Ergebnis <> 0	Relativ	BNE Operand	D0	2	2+
BPL	Verzweigung bei					
	Plus-Resultat	Relativ	BPL Operand	10	2	2+
BRK	Unterbrechung					
		Impliziert	BRK	00	1	7

BVC	Verzweigung bei kein Überlauf	Relativ	BVC Operand	50	2	2+
BVS	Verzweigung bei Überlauf	Relativ	BVS Operand	70	2	2+
CLC	Löschen des Übertrag-Flags	Impliziert	CLC	18	1	2
CLD	Löschen des Dezimal-Modus	Impliziert	CLD	D8	1	2
CLI	Löschen des IRQ-Disable-Bits	Impliziert	CLI	58	1	2
CLV	Löschen des Übertrag-Flags	Impliziert	CLV	B8	1	2
CMP	Vergleich von Speicher/Akku	Umittelbar	CMP #Operand	C9	2	2
		Zero-Page	CMP Operand	C5	2	3
		Zero-Page,X	CMP Operand,X	D5	2	4
		Absolut	CMP Operand	CD	3	4
		Absolut,X	CMP Operand,X	DD	3	4+
		(Indirekt,X)	CMP (Operand,X)	C1	2	6
		(Indirekt),Y	CMP (Operand),Y	D1	2	5+
CPX	Vergleich von Speicher/X-Reg.	Unmittelbar	CPX #Operand	E0	2	2
		Zero-Page	CPX Operand	E4	2	3
		Absolut	CPX Operand	EC	3	4
CPY	Vergleich von Speicher/Y-Reg.	Umittelbar	CPY #Operand	C0	2	2
		Zero-Page	CPY Operand	C4	2	3
		Absolut	CPY Operand	CC	3	4
DEC	Speicherdekrement um 1	Zero-Page	DEC Operand	C6	2	5
		Zero-Page,X	DEC Operand,X	D6	2	6

	Absolut	DEC Operand	CE	3	3
	Absolut,X	DEC Operand,X	DE	3	7
DEX Dekrement von X-Reg. um 1	Impliziert	DEX	CA	1	2
DEY Dekrement von Y-Reg. um 1	Impliziert	DEY	88	1	2
EOR Exklusiv-Oder-Vergleich von Speicher/Akku	Unmittelbar	EOR #Operand	49	2	2
	Zero-Page	EOR Operand	45	2	3
	Zero-Page,X	EOR Operand,X	55	2	4
	Absolut	EOR Operand	4D	3	4
	Absolut,X	EOR Operand,X	5D	3	4+
	(Indirekt,X)	EOR (Operand,X)	41	2	6
	(Indirekt),Y	EOR (Operand),Y	51	2	5+
INC Speicherinkrement um 1	Zero-Page	INC Operand	E6	2	5
	Zero-Page,X	INC Operand,X	F6	2	6
	Absolut	INC Operand	EE	3	6
	Absolut,X	INC Operand,X	FE	3	7
INX Inkrement von X-Reg. um 1	Impliziert	INX	E8	1	2
INY Inkrement von Y-Reg. um 1	Impliziert	INY	C8	1	2
JMP Sprung zu neuem Speicherplatz	Absolut	JMP Operand	4C	3	3
	Indirekt	JMP (Operand)	6C	3	5
JSR Sprung zu einem Unterprogramm	Absolut	JSR Operand	20	3	6
LDA Speicherübertragung zum Akku	Unmittelbar	LDA #Operand	A9	2	2
	Zero-Page	LDA Operand	A5	2	3
	Zero-Page,X	LDA Operand,X	B5	2	4

		Absolut	LDA Operand	AD	3	4
		Absolut,X	LDA Operand,X	BD	3	4+
		(Indirekt,X)	LDA (Operand,X)	A1	2	6
		(Indirekt),Y	LDA (Operand),Y	B1	2	5+
LDX	Speicherübertra-					
	gung zum X-Reg.	Unmittelbar	LDX #Operand	A2	2	2
		Zero-Page	LDX Operand	A6	2	3
		Zero-Page,Y	LDX Operand,Y	B6	2	4
		Absolut	LDX Operand	AE	3	4
		Absolut,Y	LDX Operand,Y	BE	3	4+
LDY	Speicherübertra-					
	gung zum Y-Reg.	Unmittelbar	LDY #Operand	A0	2	2
		Zero-Page	LDY Operand	A4	2	3
		Zero-Page,X	LDY Operand,X	B4	2	4
		Absolut	LDY Operand	AC	3	4
		Absolut,X	LDY Operand,X	BC	3	4+
LSR	Verschiebung um					
	1 Bit . rechts	Akkumulator	LSR A	4A	1	2
		Zero-Page	LSR Operand	46	2	5
		Zero-Page,X	LSR Operand,X	56	2	6
		Absolut	LSR Operand	4E	3	6
		Absolut,X	LSR Operand,X	5E	3	7
NOP	Keine					
	Operation	Impliziert	NOP	EA	1	2
ORA	ODER-Verknüpf.					
	von Speicher	Umittelbar	ORA #Operand	09	2	2
	und Akku	Zero-Page	ORA Operand	05	2	3
		Zero-Page,X	ORA Operand,X	15	2	4
		Absolut	ORA Operand	0D	3	4
		Absolut,X	ORA Operand,X	1D	3	4+
		Absolut,Y	ORA Operand,Y	19	3	4+
		(Indirekt,X)	ORA (Operand,X)	01	2	6
		(Indirekt),Y	ORA (Operand),Y	11	2	5
PHA	Speicherung des					
	Akku im Stack	Impliziert	PHA	48	1	3

PHP	Speicherung des Status im Stack	Impliziert	PHP	08	1	3
PLA	Akkumulator vom Stack holen	Impliziert	PLA	68	1	4
PLP	Status vom Stack holen	Impliziert	PLP	28	1	4
ROL	Rotiere um 1 Bit nach links	Akkumulator	ROL A	2A	1	2
		Zero-Page	ROL Operand	26	2	5
		Zero-Page,X	ROL Operand,X	36	2	6
		Absolut	ROL Operand	2E	3	6
		Absolut,X	ROL Operand,X	3E	3	7
ROR	Rotiere um 1 Bit nach rechts	Akkumulator	ROR A	6A	1	2
		Zero-Page	ROR Operand	66	2	5
		Zero-Page,X	ROR Operand,X	76	2	6
		Absolut	ROR Operand	5E	3	6
		Absolut,X	ROR Operand,X	7E	3	7
RTI	Rückkehr vom Interrupt	Impliziert	RTI	40	1	6
RTS	Rückkehr von Unterprogramm	Impliziert	RTS	60	1	6
SBC	Speicher und Übertrag vom Akku subtrah.	Unmittelbar	SBC #Operand	E9	2	2
		Zero-Page	SBC Operand	E5	2	3
		Zero-Page,X	SBC Operand,X	F5	2	4
		Absolut	SBC Operand	ED	3	4
		Absolut,X	SBC Operand,X	FD	3	4+
		(Indirekt,X)	SBC (Operand,X)	E1	2	6
		(Indirekt),Y	SBC (Operand),Y	F1	2	5+
SEC	Übertrag-Flag setzen	Impliziert	SEC	38	1	2

SED	Dezimal-Modus einschalten	Impliziert	SED	F8	1	2
SEI	Setzen des IRQ-Disable-Bits	Impliziert	SEI	78	1	2
STA	Akku in Speicher ablegen	Zero-Page	STA Operand	85	2	3
		Zero-Page,X	STA Operand,X	95	2	4
		Absolut	STA Operand	8D	3	4
		Absolut,X	STA Operand,X	9D	3	5
		Absolut,Y	STA Operand,Y	99	3	5
		(Indirekt,X)	STA (Operand,X)	81	2	6
		(Indirekt),Y	STA (Operand),Y	91	2	6
STX	X-Register in Speicher able.	Zero-Page	STX Operand	86	2	3
		Zero-Page,Y	STX Operand,Y	96	2	4
		Absolut	STX Operand	8E	3	4
STY	Y-Register in Speicher able.	Zero-Page	STY Operand	84	2	3
		Zero-Page,X	STY Operand,X	94	2	4
		Absolut	STY Operand	8C	3	4
TAX	Akku ins X-Reg. übertragen	Impliziert	TAX	AA	1	2
TAY	Akku ins Y-Reg. übertragen	Impliziert	TAY	A8	1	2
TSX	Stackpointer in X-Reg. übertr.	Impliziert	TSX	BA	1	2
TXA	X-Reg. in Akku übertragen	Impliziert	TXA	8A	1	2
TXS	X-Reg. in Stackpointer übertr.	Impliziert	TXS	9A	1	2

TYA Y-Reg. in Akku
übertragen Impliziert TYA 98 1 2

4.4 DER GEBRAUCH DER KERNAL-ROUTINEN

Auf dem Microcomputer-Sektor gibt es eine Frage, die Programmierer immer wieder beschäftigt: Was tun, wenn das Computer-Betriebssystem von der Hersteller-Firma geändert wird? Langwierig erstellte Maschinensprache-Programme funktionieren möglicherweise nicht mehr und müssen grundlegend geändert werden. Um dieses Problem zu mindern, hat Commodore ein Prinzip entwickelt, um Programmierern die Arbeit zu erleichtern. Es handelt sich hierbei um das sogenannte **KERNAL**. Im wesentlichen ist KERNAL eine Standard-Sprungtabelle für Eingabe, Ausgabe und Speicherverwaltungsprogramme im Betriebssystem. Bei einer Verbesserung des Systems können die Plätze der einzelnen Programme im ROM sich ändern. Die KERNAL-Sprungtabelle wird jedoch auch stets entsprechend geändert.

Wenn Ihre Maschinensprache-Programme die Betriebssystem-Routinen nur über den KERNAL benutzen, so können Sie gegebenenfalls wesentlich einfacher gestaltet werden.

Der KERNAL ist das Betriebssystem des Commodore PLUS/4. Über ihn werden sämtliche Eingaben, Ausgaben sowie die Speicherverwaltung gesteuert.

Um Ihre Maschinensprache-Programme zu vereinfachen und sicherzustellen, daß die Programme aufgrund einer künftigen Verbesserung des Betriebssystems vom Commodore PLUS/4 nicht veralten, enthält der KERNAL eine Sprungtabelle. Durch die 39 Ein-/Ausgabe-Routinen und weitere Hilfsprogramme, die über diese Tabelle erreichbar sind, können Sie nicht nur Zeit sparen, sondern Ihre Programme von einem Commodore-Computer an den anderen anpassen. Die Sprungtabelle befindet sich beim PLUS/4 auf der letzten Page des gesamten Adreßraums ($FF49-FFF5).

Um die KERNAL-Sprungtabelle zu benutzen, geben Sie zunächst die erforderlichen Parameter für die KERNAL-Routine ein. Dann springen Sie über die **JSR**-Anweisung an die geeignete Stelle in die KERNAL-Sprungtabelle. Nach Beendigung der Routine überträgt der KERNAL die Steuerung wieder Ihrem Maschinensprache-Programm. Je nach verwendeter KERNAL-Routine werden Parameter durch bestimmte Register in Ihr Programm zurückgegeben. Die jeweiligen Adressen der einzelnen KERNAL-Routinen finden Sie in den Beschreibungen der KERNAL-Unterprogramme.

Warum benutzt man die Sprungtabelle überhaupt? Warum springt man nicht einfach direkt in das entsprechende KERNAL-Unterprogramm? Das ist eine gute Frage. Die Sprungtabelle wird benutzt, damit Ihre Maschinensprache-Programme auch dann funktionieren, wenn der KERNAL oder der BASIC-Interpreter geändert werden. In künftigen Betriebssystemen können die Speicherplätze der einzelnen Routinen an unterschiedlichen Positionen im Adreßbereich liegen... Die Sprungtabelle arbeitet jedoch immer noch richtig!

KERNAL-FUNKTIONEN NACH DEM EINSCHALTEN BZW. RESET

1) Nach dem Einschalten der Stromversorgung wird durch den KERNAL zunächst der Stack-Pointer zurückgesetzt und der Dezimalmodus gelöscht.

2) Dann prüft das KERNAL, ob in Adresse $8000 (dez. 32768) ein ROM mit Autostart (z.B. ein Spielmodul) vorhanden ist. Ist dieses vorhanden, dann wird die normale Initialisierung abgebrochen und die Steuerung dem im ROM abgelegten Programm übertragen. Ist ein solches ROM nicht vorhanden, wird die normale Systeminitialisierung fortgesetzt.

3) Als nächstes initialisiert der KERNAL alle Ein-/Ausgabe-Vorrichtungen. Der serielle Bus wird initialisiert, der Kassettenmotor abgestellt und der TED-Chip mit den Standard-Werten geladen.

4) Jetzt prüft der KERNAL, ob die STOP-Taste gedrückt ist. Wenn ja, verzweigt es zum Maschinensprache-Monitor.

5) Als nächstes führt der KERNAL einen RAM-Test durch und setzt oben und unten die RAM-Pointer. Auch die Zero-Page wird initialisiert und der Kassettenpuffer vorbereitet.

6) Abschließend führt der KERNAL folgende Routinen aus: Die I/O-Vektoren werden auf die Standard-Werte gesetzt. Der Bildschirm wird dann gelöscht und alle Bildschirm-Editor-Variablen zurückgesetzt. Dann wird der BASIC-Interpreter bei $8000 gestartet.

ARBEITEN MIT DEM KERNAL

Beim Schreiben von Programmen in Maschinensprache ist es oft empfehlenswert, Betriebssystem-Routinen zu benutzen. Diese umfassen Ein-/Ausgabe, Zugriff auf den System-Taktgeber, Speicherverwaltung und ähnliche Funktionen. Es ist überflüssig, diese Routinen ständig neu zu schreiben. Durch den einfachen Zugriff auf das Betriebssystem wird daher das Programmieren in Maschinensprache beschleunigt.

Wie bereits erwähnt, stellt der KERNAL eine Sprungtabelle dar. Diese ist lediglich eine Ansammlung von JMP-Anweisungen zu zahlreichen Betriebssystem-Routinen.

Um mit einer KERNAL-Routine zu arbeiten, müssen Sie zunächst alle für diese Routine erforderlichen Vorbereitungen treffen. Wenn eine Routine z.B. zunächst den Aufruf einer anderen KERNAL-Routine erfordert, dann müssen Sie diese aufrufen. Setzt die Routine die Eingabe einer Zahl in den Akkumulator voraus, dann muß diese Zahl auch eingegeben sein. Werden die Bedingungen nicht erfüllt, dann können die Routinen natürlich auch nicht richtig arbeiten.

Nach der Durchführung sämtlicher Vorbereitungen rufen Sie die Routine über den JSR-Befehl auf. Alle zugänglichen KERNAL-Routinen sind wie Unterprogramme aufgebaut und müssen mit einem RTS-Befehl enden. Wenn die KERNAL-Routine die entsprechende Aufgabe beendet hat, wird die Steuerung bei dem Befehl nach JSR wieder Ihrem Programm übertragen.

Viele KERNAL-Routinen zeigen bei Störungen Fehler-Codes im Statuswort oder Akkumulator an. Will man gut programmieren und lauffähige Maschinenprogramme erstellen, so dürfen diese Fehler-Rückgaben nicht außer acht gelassen werden, da sonst das übrige Programm zerstört werden kann.

Dies ist alles, was Sie beim Arbeiten mit dem KERNAL zu tun haben. Gehen Sie einfach wie folgt vor:

1) Einrichten
2) Routinenaufruf
3) Fehlerbehandlung

Folgende Konventionen werden bei der Beschreibung von KERNAL-Routinen benutzt:

NAME: Bezeichnung der KERNAL-Routine

ZWECK: Kurzbeschreibung des Zwecks dieser KERNAL-Routine

ADRESSE: Dies ist die Aufrufadresse der KERNAL-Routine in Hexadezimaldarstellung.

ÜBERGABE: Unter dieser Überschrift aufgeführte Register werden zur Übertragung von Parametern zu bzw. von KERNAL-Routinen benutzt.

VORBEREITUNG: Bei bestimmten KERNAL-Routinen ist zuvor eine Dateneingabe erforderlich. Die erforderliche Routinen werden hier aufgeführt.

FEHLERMELDUNG: Ist nach dem Abarbeiten einer KERNAL-Routine das CARRY-Flag gesetzt, so zeigt dies an, daß bei einer Verarbeitung ein Fehler festgestellt wurde. Die Fehlerzahl ist im Akkumulator enthalten.

STACKBEDARF: Dies ist die tatsächliche Anzahl von Stack-Bytes, die von der KERNAL-Routine benutzt werden.

REGISTER: Alle von KERNAL-Routinen benutzte Register werden hier aufgeführt.

BESCHREIBUNG: Hier finden Sie eine kurze Funktionsbeschreibung der KERNAL-Routine.

VORGEHEN: Beschreibung der Vorgehensweise in geeigneter Reihenfolge.

BEISPIEL: Verdeutlicht die Funktionsweise an einem kleinen Beispiel.

NAME: **ACPTR**

ZWECK: **Daten vom seriellen Bus lesen**
ADRESSE: **$FFA5**
ÜBERGABE: **A**
VORBEREITUNG: **TALK, TKSA**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **13**
REGISTER: **A, X**

BESCHREIBUNG:

Diese Routine benutzen Sie, wenn Sie Informationen von einem Gerät am seriellen Bus, wie z.B. einer Diskette lesen wollen. Diese Routine liest direkt ein Datenbyte vom seriellen Bus. Dieses Datum wird in den Akkumulator übertragen. Als Vorbereitung muß zunächst die TALK-Routine aufgerufen werden. Über diese erhält der serielle Bus den Befehl für die Bus-Datenübertragung.

Wenn die Eingabevorrichtung einen Sekundärbefehl erfordert, muß dieser vor dem Aufruf über die KERNAL-Routine TKSA übertragen werden. Fehler werden im Statuswort rückgemeldet. Zum Lesen des Statusworts wird die READST-Routine benutzt.

VORGEHEN:

1) Gerät am seriellen Bus für die Datenübertragung zum PLUS/4 vorbereiten. (KERNAL-Routinen TALK und TKSA benutzen.)

2) Diese Routine aufrufen (über JSR).

3) Daten speichern oder benutzen.

BEISPIEL:

```
$1010                           ; HOLE EIN BYTE VOM BUS
$1010  20 A5 FF  JSR ACPTR
$1013  85 D0     STA DATA
```

NAME: **CHKIN**

ZWECK: **Kanal für Eingabe öffnen**
ADRESSE: **$FFC6**
ÜBERGABE: **X**
VORBEREITUNG: **(OPEN)**
FEHLERMELDUNG:
STACKBEDARF: **Keiner**
REGISTER: **A, X**

BESCHREIBUNG:

Jede über die KERNAL-Routine OPEN geöffnete logische Datei kann über diese Routine als Eingabekanal definiert werden. Natürlich muß es sich dabei um ein Eingabegerät handeln, da sonst ein Fehler auftritt und die Routine unterbrochen wird.

Werden die Daten nicht über die Tastatur eingegeben, dann muß diese Routine vor dem Arbeiten mit den KERNAL-Routinen CHRIN oder GETIN für die Dateneingabe zuvor aufgerufen werden. Soll die Eingabe über die Tastatur erfolgen und sind keine weiteren Eingabekanäle geöffnet, dann wird diese Routine und die OPEN-Routine nicht benötigt.

Wird diese Routine mit einem Gerät am seriellen Bus benutzt, dann wird über den Bus automatisch die Talk-Adresse (und die Sekundäradresse, wenn eine solche durch die OPEN-Routine festgelegt wurde) übertragen.

VORGEHEN:

1) Logische Datei öffnen (gegebenenfalls dazugehörige Beschreibung durchlesen).

2) Register X mit der Nummer der zu verwendenden logischen Datei laden.

3) Diese Routine aufrufen (über JSR).

Mögliche Fehler:

#3 : Datei nicht offen
#5 : Gerät nicht vorhanden
#6 : Datei ist keine Eingabedatei

BEISPIEL:

```
$1010                              ; VORBEREITUNG AUF EINE EINGABE
$1010                              ; VOM LOGISCHEN FILE 2
$1010  A2 02      LDX #2
$1012  20 C6 FF   JSR CHKIN
```

NAME: **CHKOUT**

ZWECK: **Kanal für Ausgabe öffnen**
ADRESSE: **$FFC9**
ÜBERGABE: **X**
VORBEREITUNG: **(OPEN)**
FEHLERMELDUNG: **0, 3, 5, 7 (siehe READST)**
STACKBEDARF: **4+**
REGISTER: **A, X**

BESCHREIBUNG:

Eine über die KERNAL-Routien OPEN erstellte logische Dateinummer kann als Ausgabekanal definiert werden. Natürlich muß es sich hierbei um ein Ausgabegerät handeln, da es sonst zu einem Fehler kommt und die Routine unterbrochen wird.

Ehe Daten zu einem Ausgabegerät übertragen werden, ist ein Aufruf dieser Routine erforderlich. Es sei denn, Sie wollen den Bildschirm des PLUS/4 als Ausgabegerät benutzen. Wird eine Bildschirmausgabe gewünscht und ist noch kein anderer Ausgabekanal definiert, dann werden diese Routinen und die OPEN-Routine nicht benötigt.

Beim Öffnen des Kanals zum seriellen Bus überträgt diese Routine automatisch die durch die OPEN-Routine festgelegte LISTEN-Adresse (und gegebenenfalls eine Sekundäradresse).

VORGEHEN:

1) Eine logische Dateinummer, eine LISTEN-Adresse und eine Sekundäradresse (falls erforderlich) über die KERNAL-Routine OPEN festlegen.

2) Register X mit der in der OPEN-Anweisung benutzten logischen Dateinummer laden.

3) Diese Routine aufrufen (über JSR).

Mögliche Fehler:

#3 : Datei nicht offen
#5 : Gerät nicht vorhanden
#7 : Keine Ausgabedatei

BEISPIEL:

```
$1010  A2 03     LDX #3          ; DEFINIERE DEN LOGISCHEN FILE 3
$1012  20 C9 FF  JSR CHKOUT      ; ALS AUSGABE-KANAL
```

NAME: **CHRIN**

ZWECK: **Zeicheneingabe**
ADRESSE: **$FFCF**
ÜBERGABE: **A**
VORBEREITUNG: **(OPEN, CHKIN)**
FEHLERMELDUNG: **0 (siehe READST)**
STACKBEDARF: **7+**
REGISTER: **A, X**

BESCHREIBUNG:

Über diese Routine wird ein Datenbyte von dem über die KERNAL-Routien CHKIN bereits als Eingabekanal festgelegten Kanal gelesen. Wurde CHKIN nicht zur Definition eines weiteren Eingabekanals benutzt, dann wird davon ausgegangen, daß sämtliche Daten über die Tastatur eingegeben werden. Das Datenbyte wird in den Akkumulator übertragen. Nach dem Aufruf bleibt der Kanal offen.

Eingaben über die Tastatur werden auf besondere Weise gehandhabt. Zunächst wird der Cursor eingeschaltet und blinkt so lange, bis ein CR (Carriage Return) eingegeben wird. Alle Zeichen in einer Zeile (max. 88 Zeichen) werden im BASIC-Eingabepuffer gespeichert. Diese Zeichen können einzeln aufgerufen werden, indem man diese Routine für jedes einzelne Zeichen aufruft. Nach dem "Carriage Return" wird die gesamte Zeile verarbeitet. Beim nachfolgenden Aufruf dieser Routine beginnt der gleiche Vorgang wieder von vorn, d.h. mit einem Blinken des Cursors.

VORGEHEN (von der Tastatur):

1) Ein Datenbyte durch diese Routine aufrufen.

2) Datenbyte speichern

3) Prüfen, ob es sich um das letzte Datenbyte handelt (ist es ein CR?).

4) Wenn nicht, bei Schritt 1) fortsetzen.

BEISPIEL:

```
$1010  A2 00     LDY #$00          ; Y-REGISTER VORBEREITEN
$1012  20 CF FF  JSR CHRIN
$1015  99 D0 00  STA DATA,Y        ; SPEICHERE DAS Y. DATEN-BYTE AN
$1018                              ; DER Y. STELLE IM DATENBEREICH
$1018  C8        INY
$1019  C9 13     CMP #CR           ; IST ES EIN "CARRIAGE RETURN"?
$101B  D0 F5     BNE $1012         ; NEIN, HOLE NÄCHSTES BYTE
```

VORGEHEN (von anderer Peripherie):

1) KERNAL-Routinen OPEN und CHKIN benutzen.

2) Diese Routine aufrufen (über JSR).

3) Daten speichern.

BEISPIEL:

```
$1010  20 CF FF  JSR CHRIN
$1013  8D F0 00  STA DATA
```

NAME: **CHROUT**

ZWECK: **Zeichenausgabe**
ADRESSE: **$FFD2**
ÜBERGABE: **A**
VORBEREITUNG: **(CHKOUT, OPEN)**
FEHLERMELDUNG: **0 (siehe READST)**
STACKBEDARF: **8+**
REGISTER: **A**

BESCHREIBUNG:

Über diese Routine wird ein Zeichen zu einem bereits geöffneten Kanal ausgegeben. Bestimmen Sie vor dem Aufruf der Routine den Ausgabekanal durch die KERNAL-Routinen OPEN und CHKOUT. Wird dieser Aufruf ausgelassen, dann erfolgt die Datenübertragung zum Standard-Ausgabegerät (Nr. 3, Bildschirm). Das auszugebende Datenbyte wird in den Akkumulator übertragen und diese Routine aufgerufen. Die Daten werden dann zum angegebenen Ausgabegerät übertragen. Nach dem Aufruf bleibt der Kanal geöffnet.

Besondere Vorsicht ist geboten, wenn diese Routine für die Datenübertragung zu einem speziellen Gerät am seriellen Bus benutzt wird, da alle Daten zu allen offenen Bus-Ausgabekanälen übertragen werden. Ist dies nicht erwünscht, müssen alle Ausgabekanäle des seriellen Bus bis auf den gewünschten Kanal durch die KERNAL-Routine CLRCHN geschlossen werden.

VORGEHEN:

1) Gegebenenfalls KERNAL-Routine CHKOUT benutzen (siehe oben).

2) Auszugebende Daten in den Akkumulator laden.

3) Diese Routine aufrufen.

BEISPIEL:

```
$1010  ; ERSETZE DEN BASIC-BEFEHL: CMD 4, "A";
$1010  A2 04     LDX #4            ; LOGISCHER FILE #4
$1012  20 C9 FF  JSR CHKOUT        ; AUSGABE-KANAL ÖFFNEN
```

```
$1015  A9 41     LDA #"A"
$1017  20 D2 FF  JSR CHROUT       ; ZEICHEN AUSGEBEN
```

NAME: **CIOUT**

ZWECK: **Byte-Ausgabe über den seriellen Bus**
ADRESSE: **$FFA8**
ÜBERGABE: **A**
VORBEREITUNG: **LISTEN, (SECOND)**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **5**
REGISTER: **Keine**

BESCHREIBUNG:

Diese Routine wird für die Informationsübertragung zu einem Gerät am seriellen Bus benutzt. Durch Aufruf dieser Routine wird ein Datenbyte auf den seriellen Bus mit "Handshake" übertragen. Vor dem Aufruf muß die KERNAL-Routine LISTEN benutzt werden, um das Gerät am seriellen Bus für den Datenempfang vorzubereiten. (Wird eine Sekundäradresse benötigt, so muß diese über die KERNAL-Routine SECOND übertragen werden.)

Der Akkumulator wird dann mit einem Byte geladen, das als Datum über den seriellen Bus übertragen wird. Eine Vorrichtung muß für den Datenempfang bereit sein, da sonst das Statuswort ein "Timeout" meldet. Bei dieser Routine wird stets ein Zeichen zwischengespeichert. Ruft man die KERNAL-Routine UNLSN zur Beendigung der Datenübertragung auf, so wird das im Puffer befindliche Zeichen mit einem EOI übertragen. Dann wird der Befehl UNLSN zum Gerät übertragen.

VORGEHEN:

1) KERNAL-Routine LISTEN (und gegebenenfalls SECOND) benutzen.

2) Ein Datenbyte in den Akkumulator laden.

3) Zur Übertragung des Datenbytes diese Routine aufrufen.

BEISPIEL:

```
$1010  A9 58     LDA #"X"          ; SENDE EIN X ZUM SERIELLEN BUS
$1012  20 A8 FF  JSR CIOUT
```

NAME: **CINT**

ZWECK: **Initialisierung von Bildschirmeditor und TED-Chip**
ADRESSE: **$FF81**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **4**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine wird der TED-Chip im Commodore PLUS/4 initialisiert. Auch der KERNAL-Bildschirmeditor wird initialisiert. Diese Routine kann über ein Programm-Modul des PLUS/4 aufgerufen werden.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 81 FF  JSR CINT          ; TED-CHIP INITIALISIEREN
$1013  20 00 20  JMP RUN           ; HAUPTPROGRAMM STARTEN
```

NAME: **CLALL**

ZWECK: **Schließen sämtlicher Dateien**
ADRESSE: **$FFE7**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **11**
REGISTER: **A, X**

BESCHREIBUNG:

Über diese Routine werden alle offenen Dateien geschlossen. Beim Aufruf werden die Zeiger in der Tabelle der offenen Dateien rückgestellt und alle Dateien geschlossen. Die Routine CLRCHN wird automatisch zur Rückstellung der Ein-/Ausgabekanäle aufgerufen.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 E7 FF  JSR CLALL        ; ALLE DATEIEN SCHLIESSEN UND
$1013                             ; STANDARD I/O-KANÄLE SETZEN
$1013  20 00 20  JMP RUN          ; HAUPTPROGRAMM STARTEN
```

NAME: **CLOSE**

ZWECK: **Schließen einer logischen Datei**
ADRESSE: **$FFC3**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **0, 240 (siehe READST)**
STACKBEDARF: **2+**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Diese Routine wird zum Schließen einer logischen Datei benutzt, nachdem alle Ein-/Ausgaben in Bezug auf die Datei beendet sind. Die Routine wird aufgerufen, nachdem der Akkumulator mit der entsprechenden logischen Dateinummer geladen wurde. Es handelt sich hierbei um die gleiche Nummer, die beim Öffnen der Datei über die Routine OPEN benutzt wurde.

VORGEHEN:

1) Entsprechende logische Dateinummer in den Akkumulator laden.

2) Diese Routine aufrufen.

BEISPIEL:

```
$1010                              ; KANAL 15 SCHLIESSEN
$1010  A9 0F     LDA #15
$1012  20 CC FF  JSR CLOSE
```

NAME: **CLRCHN**

ZWECK: **Löschen von Ein-/Ausgabekanälen**
ADRESSE: **$FFCC**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMELDUNG:
STACKBEDARF: **9**
REGISTER: **A, X**

BESCHREIBUNG:

Diese Routine wird zum Löschen aller offenen Kanäle und zur Rückstellung der Ein-/Ausgabekanäle auf die Standardwerte aufgerufen. Normalerweise erfolgt der Aufruf nach Öffnen anderer Ein-/Ausgabekanäle (z.B. Kassette oder Diskette) und zur Beendigung der entsprechenden Ein-/Ausgaben. Die Standard-Eingabegerätenummer ist 0 (Tastatur). Das Standardausgabegerät ist 3 (Bildschirm).

Ist einer der zu schließenden Kanäle der seriellen Bus, so wird zunächst zum Löschen des Eingabekanals ein UNTALK-Signal oder zum Löschen des Ausgabekanals ein UNLISTEN-Signal übertragen. Wird diese Routine nicht aufgerufen (und bleiben die Anschlußgeräte am seriellen Bus empfangsbereit), dann können mehrere Geräte die gleichen Daten vom Commodore PLUS/4 gleichzeitig empfangen. Hierdurch könnte z.B. der Drucker für die Ausgabe und die Diskette für den Datenempfang eingesetzt werden. Auf diese Weise kann eine Diskettendatei direkt ausgedruckt werden.

Beim Ausführen der KERNAL-Routine CLALL wird diese Routine automatisch aufgerufen.

VORGEHEN:

1) Diese Routine über die JSR-Anweisung aufrufen.

BEISPIEL:

```
$1010  20 CC FF  JSR CLRCHN
```

NAME: **GETIN**

ZWECK: **Ein Zeichen lesen**
ADRESSE: **$FFE4**
ÜBERGABE: **A**
VORBEREITUNG: **CHKIN, OPEN**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **7+**
REGISTER: **A (X, Y)**

BESCHREIBUNG:

Handelt es sich bei dem Kanal um die Tastatur, dann nimmt dieses Unterprogramm ein Zeichen aus dem Tastaturpuffer und überträgt es als ASCII-Wert in den Akkumulator. Ist der Puffer leer, dann wird der Wert 0 in den Akku geladen. Zeichen werden automatisch durch eine Tastatur-Abfrageroutine, die die Routine SCNKEY aufruft, in eine "Warteschlange" übertragen. Im Tastaturpuffer können max. 10 Zeichen gespeichert sein. Ist der Puffer gefüllt, dann werden das 11. und alle weiteren Zeichen solange überlesen, bis mindestens ein Zeichen aus der Warteschlange entfernt wurde.

Handelt es sich bei dem Kanal um RS-232, dann wird nur Register A benutzt und ein einzelnes Zeichen wiedergegeben. Zur Überprüfung siehe READST. Handelt es sich bei dem Kanal um den seriellen Bus, die Kassette oder den Bildschirm, dann rufen Sie die BASIN-Routine auf.

VORGEHEN:

1) Diese Routine über eine JSR-Anweisung aufrufen.

2) Prüfen, ob im Akkumulator eine 0 gespeichert ist (leerer Puffer).

3) Daten verarbeiten.

BEISPIEL:

```
$1010                           ; AUF EIN ZEICHEN WARTEN
$1010  20 E4 FF   JSR GETIN
$1013  C9 00      CMP #0
$1015  F0 F9      BEQ $1010
```

NAME: **IOBASE**

ZWECK: **Festlegen des Ein-/Ausgabe-Speicherbereichs**
ADRESSE: **$FFF3**
ÜBERGABE: **X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG:
STACKBEDARF: **2**
REGISTER: **X, Y**

BESCHREIBUNG:

Über diese Routine werden im X- und Y-Register nieder- und höherwertige Bytes der Adresse des Speicherabschnitts abgespeichert, in dem sich die Ein-/Ausgaberegister befinden. Diese Adresse kann dann zusammen mit relativen Adressen für den Zugriff auf die Ein-/Ausgaberegister des Commodore PLUS/4 benutzt werden. Register X enthält das untere Adreßbyte, Register Y das obere Adreßbyte.

Durch diese Routine wird die Kompatibilität zwischen dem VC-20, dem C-64 und dem PLUS/4 gewährleistet. Werden die Ein-/Ausgaberegister für ein Maschinenspracheprogramm durch Aufruf dieser Routine gesetzt, dann sind sie auch mit künftigen Versionen des Commodore PLUS/4, was KERNAL und BASIC angeht, kompatibel.

VORGEHEN:

1) Diese Routine über die JSR-Anweisung aufrufen.

2) Die Register X und Y in aufeinanderfolgenden Plätzen speichern.

3) Die Verschiebung in Register Y laden.

4) Auf diesen Ein-/Ausgabeplatz zugreifen.

BEISPIEl: (nur bei PLUS/4)

```
$1010                          ; SETZE DAS DATENRICHTUNGSRE-
```

```
$1010                           ; GISTER DES USER-PORT AUF 0.
$1010  20 F3 FF   JSR IOBASE
$1013  86 D0      STX POINT     ; SETZE BASIS-REGISTER
$1015  84 D1      STY POINT+1
$1017  A2 02      LDY #2
$1019  A9 00      LDA #0        ; OFFSET FÜR DDR DES USER-PORT
$101B  91 D0      STA (POINT),Y ; SETZE DDR AUF 0
```

NAME: **IOINIT**

ZWECK: **Initialisieren von Ein-/Ausgabegeräten**
ADRESSE: **$FF84**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMELDUNG:
STACKBEDARF: **Keiner**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine werden alle Ein-/Ausgabevorrichtungen und Routinen initialisiert. Sie wird normalerweise als Teil der Initialisierung eines Programmoduls des Commodore PLUS/4 aufgerufen.

BEISPIEL:

```
$1010  20 84 FF  JSR IOINIT
```

NAME: **LISTEN**

ZWECK: **LISTEN-Befehl für ein Gerät am seriellen Bus**
ADRESSE: **$FFB1**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **Keiner**
REGISTER: **A**

BESCHREIBUNG:

Über diese Routine wird Gerät am seriellen Bus für den Datenempfang vorbereitet. Eine Gerätenummer zwischen 0 und 31 wird vor dem Aufruf dieser Routine in den Akkumulator geladen. Über die LISTEN-Anweisung wird die Zahl Bit für Bit ODER-verknüpft und in eine LISTEN-Adresse umgewandelt. Dieses Datum wird dann als Befehl über den seriellen Bus übertragen. Das angegebene Gerät ist dann für den Datenempfang bereit.

VORGEHEN:

1) Die gewünschte Gerätenummer in den Akkumulator laden.

2) Diese Routine über die JSR-Anweisung aufrufen.

BEISPIEL:

```
$1010                              ; GERATE-NR. 8 AUF LISTEN
$1010  A9 08      LDA #8
$1012  20 B1 FF   JSR LISTEN
```

NAME: **LOAD**

ZWECK: **RAM laden von Peripherie**
ADRESSE: **$FFD5**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **SETLFS, SETNAM**
FEHLERMELDUNG: **0, 4, 5, 8, 5, READST**
STACKBEDARF: **Keiner**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine werden Datenbytes direkt von einem beliebigen Eingabegerät in den Speicher des Commodore PLUS/4 geladen. Sie kann auch für einen Vergleich der vom Gerät stammenden Daten mit denen im Speicher benutzt werden, während die im RAM gespeicherten Daten unverändert bleiben (VERIFY).

Zum Laden wird der Akkumulator auf 0 und für VERIFY auf 1 gesetzt. Wird ein OPEN auf das Eingabegerät mit der Sekundäradresse (SA) 0 eingegeben, dann wird die Ladeadresse ignoriert. In diesem Fall müssen die Register X und Y die Startadresse enthalten. Wird die Sekundäradresse 1, 0 oder 2 gewählt, dann werden die Daten ab der durch die Ladeadresse gegebenen Position in den Speicher geladen. Diese Routine ermittelt die Adresse des obersten benutzten RAM-Platzes.

Vor dem Aufruf dieser Routine müssen die KERNAL-Routinen SETLFS und SETNAM aufgerufen werden.

Ein LOAD über Tastatur (0), RS-232 (2) oder Bildschirm (3) ist nicht möglich.

VORGEHEN:

1) Routine SETLFS und SETNAM aufrufen. Wird ein verschobenes Laden gewünscht, Routine SETLFS zum übertragen der Sekundäradresse 0 benutzen.

2) Register A zum Laden auf 0 und zum Überprüfen auf 1 setzen.

3) Wird Laden an eine bestimmte Adresse gewünscht, müssen Register X und Y auf die Lade-Startadresse gesetzt sein.

4) Die Routine über die JSR-Anweisung aufrufen.

BEISPIEL:

```
$1010                              ; LADE EIN PROGRAMM VON KASSETTE
$1010  A9 01     LDA #DEVICE1      ; SETZE GERÄTE-NR.
$1012  A2 01     LDX #FILENO       ; SETZE LOGISCHE FILE-NR.
$1014  A0 01     LDY CMD1          ; SETZE SEKUNDÄRADRESSE
$1016  20 BA FF  JSR SETLFS
$1019  A9 0C     LDA #NAME1-NAME   ; LADE DEN AKKU MIT DER LÄNGE
$1019                              ; DES PROGRAMM-NAMENS (12)
$101B  A2 32     LDX #<NAME        ; LADE X UND Y MIT DER ADRESSE
$101D  A0 10     LDY #>NAME        ; DES PROGRAMM-NAMENS ($1032)
$101F  20 BD FF  JSR SETNAM
$1022  A9 00     LDA #0            ; SETZE FLAG FÜR LOAD
$1024  A2 FF     LDX #$FF
$1026  A0 FF     LDY #$FF
$1028  20 D5 FF  JSR LOAD
$102B  86 2D     STX VARTAB        ; ENDADRESSE DES PROGRAMMS
$102D  84 2E     STY VARTAB+1
$102F  4C 00 20  JMP START
$1032  50 52 4F  .BYT "PROGRAMMNAME"
$1035  47 52 41
$1038  4D 4D 4E
$103B  41 4D 45
```

NAME: **MEMBOT**

ZWECK: **Setzen des Zeigers für das unterer Speicherende**
ADRESSE: **$FF9C**
ÜBERGABE: **X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **Keiner**
REGISTER: **X, Y**

BESCHREIBUNG:

Diese Routine wird zum Setzen des unteren Speicherzeigers benutzt. Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gesetzt, dann wird der Zeiger des untersten RAM-Bytes im Register X und Y wiedergegeben. Bei einem Commodore PLUS/4 ist der Zeigeranfangswert normalerweise $1000 (4096 dezimal). Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gelöscht (=0), dann werden die Werte des Registers X und Y zum nieder- bzw. höherwertigen Byte des Zeigers, der den RAM-Anfang festlegt, übertragen.

VORGEHEN (Lesen des RAM-Anfangs):

1) Übertrags-Flag setzen.

2) Diese Routine aufrufen.

VORGEHEN (Setzen des Speicheranfangs):

1) Übertrags-Flag löschen.

2) Diese Routine aufrufen.

BEISPIEL:

```
$1010                          ; UNTERE SPEICHERGRENZE UM EINE
```

```
$1010                           ; PAGE (256 BYTES) HERAUFSETZEN
$1010  38       SEC             ; SPEICHERANFANG LESEN
$1011  20 9C FF JSR MEMBOT
$1014  E8       INY
$1015  18       CLC             ; SPEICHERANFANG SETZEN
$1016  20 9C FF JSR MEMBOT
```

NAME: **MEMTOP**

ZWECK: **Setzen des Zeigers für das obere Speicherende**
ADRESSE: **$FF99**
ÜBERGABE: **X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **X, Y**

BESCHREIBUNG:

Über diese Routine wird das RAM-Ende gesetzt. Ist beim Aufruf der Routine das Akkumulator-Übertragsbit gesetzt, dann wird der Zeiger des RAM-Endes in das Register X und Y geladen. Ist beim Aufruf der Routine das Akkumulator-Übertragsbit gelöscht, dann werden die Inhalte von Register X und Y in den Speicherendezeiger geladen und so die Speicherendposition geändert.

BEISPIEL:

```
$1010                              ; FREIGABE DES RS-232 BUFFER
$1010  38        SEC
$1011  20 99 FF  JSR MEMTOP        ; SPEICHEROBERGRENZE LESEN
$1014  88        DEX
$1015  18        CLC
$1016  20 99 FF  JSR MEMTOP        ; SPEICHEROBERGRENZE SETZEN
```

NAME: **OPEN**

ZWECK: **Öffnen einer logischen Datei**
ADRESSE: **$FFC0**
ÜBERGABE: **Keine**
VORBEREITUNG: **SETLFS, SETNAM**
FEHLERMELDUNG: **1, 2, 4, 5, 6, 240, READST**
STACKBEDARF: **Keiner**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine kann eine logische Datei geöffnet werden. Nach Einrichten einer logischen Datei kann diese für Ein-/Ausgaben benutzt werden. Bei den meisten KERNAL-Ein-/Ausgaberoutinen wird diese Routine zum Erstellen der entsprechenden logischen Dateien aufgerufen. Für die Verwendung dieser Routine sind keine Parameter erforderlich, es müssen jedoch die KERNAL-Routinen SETLFS und SETNAM zuvor aufgerufen werden.

VORGEHEN:

1) Routine SETLFS benutzen.

2) Routine SETNAM benutzen.

3) Diese Routine aufrufen.

BEISPIEL:

```
$1010                              ; DIESE ROUTINE SIMULIERT DEN
$1010                              ; BASIC-BEFEHL: OPEN 15,8,15,"I0
$1010  A9 02     LDA #NAME1-NAME   ; LÄNGE DES FILENAMENS (2)
$1012  A2 25     LDX #<NAME        ; ADRESSE DES FILENAMENS NACH
$1014  A0 10     LDY #>NAME        ; X UND Y BRINGEN ($1025)
$1016  20 BD FF  JSR SETNAM
$1019  A9 0F     LDA #15
$101B  A2 08     LDX #8
```

```
$101D  A0 0F     LDY #15
$101F  20 BA FF  JSR SETLFS
$1022  20 C0 FF  JSR OPEN
$1025  49 30     .BYT "I0"
```

NAME: **PLOT**

ZWECK: **Cursorposition lesen/setzen**
ADRESSE: **$FFF0**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Ist beim Aufruf dieser Routine das Akkumulator-Übertrag-Flag gesetzt, dann wird die derzeitige Cursorposition auf dem Bildschirm (in X-/Y-Koordinaten) in Register X und Y geladen. X ist die Zeilennummer (0-24) des Cursorplatzes, Y die Spaltennummer (0-39). Ist beim Aufruf das Übertragsbit gelöscht, dann bewegt sich der Cursor in die durch X, Y gegebene Position (entsprechend Register X und Y).

VORGEHEN (Lesen der Cursorposition):

1) Übertrag-Flag setzen.

2) Diese Routine aufrufen.

3) X- und Y-Position aus Register X bzw. Y lesen.

VORGEHEN (Setzen der Cursorposition):

1) Übertrag-Flag löschen.

2) In Register X und Y die gewünschte Cursorposition schreiben.

3) Diese Routine aufrufen.

BEISPIEL:

```
$1010                                   ; SETZE DEN CURSOR IN DIE
$1010                                   ; ZEILE 10, SPALTE 5
$1010  A2 0A      LDX #10
$1012  A0 05      LDY #5
$1014  18         CLC
$1015  20 F0 FF   JSR PLOT
```

NAME: **RAMTAS**

ZWECK: **RAM-Test**
ADRESSE: **$FF87**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine wird das RAM getestet und der obere bzw. untere Speicherzeiger gesetzt. Außerdem werden die Speicherplätze $0000 bis $0101 und $0200 bis $07FF gelöscht. Darüber hinaus wird der Kassettenbuffer initialisiert und der Bildschirmanfang auf $0C00 gesetzt. Normalerweise wird diese Routine als Teil der Initialisierung eines Programmoduls des Commodore PLUS/4 aufgerufen.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 87 FF  JSR RAMTAS
```

NAME: **RDTIM**

ZWECK: **Systemtaktgeber lesen**
ADRESSE: **$FFDE**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Diese Routine wird zum Lesen des Systemtaktgebers benutzt. Die Auflösung beträgt hierbei 1/60 s. Durch diese Routine werden 3 Bytes ermittelt. Der Akkumulator enthält das signifikanteste (höchste) Byte, das X-Register das nächste signifikante Byte und das Y-Register das am wenigsten signifikante Byte.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 DE FF  JSR RDTIM
$1013  84 D0     STY TIME
$1015  86 D1     STX TIME+1
$1017  85 D2     STA TIME+2
```

NAME: **READST**

ZWECK: **Statuswort lesen**
ADRESSE: **$FFB7**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A**

BESCHREIBUNG:

Über diese Routine wird der derzeitige Status der Ein-/Ausgabegeräte im Akkumulator abgelegt. Diese Routine wird normalerweise nach neuer Kommunikation mit einem Ein-/Ausgabegerät aufgerufen. Sie gibt Ihnen Informationen über den Gerätestatus oder Fehler, die während der Ein-/Ausgabe aufgetreten sind.

Die in den Akkumulator übertragenen Bits enthalten folgende Informationen:

ST BIT-POSITION	ST NUMER. WERT	LESEN VON KASSETTE	SERIELLEN BUS LESEN/ SCHREIBEN	KASSETTE LOAD/ VERIFY
0	1		Zeitsperre (Timeout) Schreiben	
1	2		Zeitsperre (Timeout) Lesen	
2	4	Kurzer Satz		Kurzer Satz
3	8	Langer Satz		Langer Satz

4	16	Nicht korrigier-barer Lesefehler		Nicht korrigier-barer Lesefehler
5	32	Prüfsummenfehler		Prüfsummenfehler
6	64	Dateiende	EOI-Leitung	
7	128	Bandende	Gerät nicht vorhanden	Bandende

VORGEHEN:

1) Diese Routine aufrufen.

2) Programminformation in Register A dekodieren.

BEISPIEL:

```
$1010                          ; PRÜFE AUF FILE-ENDE WÄHREND
$1010                          ; DES LESENS
$1010  20 B7 FF  JSR READST
$1013  49 40     AND #64       ; TESTE AUF EOF (END OF FILE)
$1015  D0 25     BNE EOF       ; VERZWEIGE, WENN GESETZT
```

NAME: **RESTOR**

ZWECK: **Normalzustand des Systems einstellen**
ADRESSE: **$FF8A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: 2
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine werden die Standardwerte sämtlicher vom KERNAL, von BASIC-Routinen und vom Interrupt benutzten Systemvektoren rückgestellt. Über die KERNAL-Routine VECTOR können die einzelnen Systemvektoren gelesen und aufbereitet werden.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 8A FF  JSR SAVE
```

NAME: **SAVE**

ZWECK: **Übertragen des RAM-Inhalts auf ein entsprechendes Gerät**
ADRESSE: **$FFD8**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **SETLFS, SETNAM**
FEHLERMELDUNG: **5, 8, 9, READST**
STACKBEDARF: **Keiner**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine wird ein Speicherbereich auf einem externen Speichermedium abgespeichert. Der Speicher wird ab der durch den Akkumulator festgelegten indirekten Adresse in der Zeropage bis zu der in den Registern X und Y festgelegten Adresse abgespeichert. Er wird zu einer logischen Datei eines Ein-/Ausgabegerätes übertragen. Vor dem Aufruf dieser Routine sind die Routinen SETLFS und SETNAM zu verwenden. Zum Sichern auf Gerät Nr. 1 (Kassette) ist jedoch kein Dateiname erforderlich (allerdings empfehlenswert). Wird versucht, auf eine andere Vorrichtung ohne Dateiname zu speichern, kommt es zu einem Fehler.

Es ist nicht möglich, auf Gerät Nr. 0 (Tastatur), 2 (RS-232) und 3 (Bildschirm) zu speichern, da sonst eine Fehlermeldung erfolgt und die Speicherung gestoppt wird.

VORGEHEN:

1) Routine SETLFS und SETNAM ausführen (wenn nicht auf Band ohne Dateiname gespeichert werden soll).

2) Zwei aufeinanderfolgende Plätze der Zeropage mit dem Zeiger laden, der auf den Anfang des abzuspeichernden Bereichs zeigt (standardmäßig kommt beim 7501 das niederwertige Byte zuerst und danach das höherwertige Byte).

3) Die Adresse des Zeigers in der Zeropage in den Akkumulator laden.

4) Das niederwertige Byte bzw. das höherwertige Byte der Endadresse des abzuspeichernden Bereichs in Register X und Y laden.

5) Diese Routine aufrufen.

BEISPIEL:

```
$1010  A9 01     LDA #1           ; GERÄT-NR. 1 (KASSETTE)
$1012  20 BA FF  JSR SETLFS
$1015  A9 00     LDA #0           ; KEIN NAME
$1017  20 BD FF  JSR SETNAM
$101A  A9 00     LDA START        ; LADE STARTADRESSE ($3C00)
$101C  85 2B     STA TXTTAB       ; (LOW BYTE)
$101E  A9 3C     LDA START+1
$1020  85 2C     STA TXTTAB+1     ; (HIGH BYTE)
$1022  A2 00     LDX END+1        ; LADE ENDADRESSE ($4000) (LOW)
$1024  A0 40     LDY END+1        ; (HIGH BYTE)
$1026  A9 2B     LDA #<TXTTAB     ; LADE AKKU MIT ZEROPAGE-ADRESSE
$1029  20 D8 FF  JSR SAVE
```

NAME: **SCNKEY**

ZWECK: **Abfrage der Tastatur**
ADRESSE: **$FF9F**
ÜBERGABE: **Keine**
VORBEREITUNG: **IOINIT**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **5**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Über diese Routine wird die Tastatur des Commodore PLUS/4 abgefragt und so festgestellt, ob und wenn ja, welche Tasten gedrückt wurden. Dies ist die gleiche Routine, die bei jedem Interrupt aufgerufen wird. Nach Drücken einer Taste wird der entsprechende ASCII-Wert in den Tastaturpuffer geschrieben.

Diese Routine wird nur aufgerufen, wenn die normalen IRQ-Unterbrechung übergangen wird.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 9F FF  JSR SCNKEY      ; TASTATUR ABFRAGEN
$1013  20 E4 FF  JSR GETIN       ; ZEICHEN HOLEN
$1016  C9 00     CMP #0          ; KEINE TASTE GEDRÜCKT?
$1018  F0 F6     BEQ GET         ; JA, ERNEUTE ABFRAGE
$101A  20 D2 FF  JSR CHROUT      ; ZEICHEN AUSGEBEN
```

NAME: **SCREEN**

ZWECK: **Ermitteln des Bildschirmformats**
ADRESSE: **$FFED**
ÜBERGABE: **X, Y**
VORBEREITUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **X, Y**

BESCHREIBUNG:

Diese Routine gibt das Bildschirmformat wieder, z.B. 40 Spalten in X und 25 Zeilen in Y. Sie kann benutzt werden, um zu bestimmen, auf welcher Maschine ein Programm läuft. Diese Funktion wurde für den Commodore 64 eingeführt, um Ihre Programme leichter mit anderen Geräten kompatibel machen zu können.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 ED FF  JSR SCREEN
$1013  86 D0     STX MAXCOL
$1015  84 D1     STY MAXROW
```

NAME: **SECOND**

ZWECK: **Übertragen der Sekundäradresse für LISTEN**
ADRESSE: **$FF93**
ÜBERGABE: **A**
VORBEREITUNG: **LISTEN**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **8**
REGISTER: **A**

BESCHREIBUNG:

Über diese Routine wird eine Sekundäradresse nach Aufruf der LISTEN-Routine zu einem Ein-/Ausgabegerät übertragen. Das Gerät ist danach empfangsbereit. Diese Routine kann nicht zur Übertragung einer Sekundäradresse nach dem Aufruf der TALK-Routine benutzt werden.

VORBEREITUNG:

1) Zu übertragende Sekundäradresse in den Akkumulator laden, geODERt mit 96.

2) Diese Routine aufrufen.

BEISPIEL:

```
$1010                              ; ADRESSIERE GERÄT NR. 8
$1010                              ; (FLOPPY) MIT SEKUNDÄR-
$1010                              ; ADRESSE 15
$1010  A9 08     LDA #8
$1012  49 60     EOR #96
$1014  20 B1 FF  JSR LISTEN
$1017  A9 0F     LDA #15
$1019  20 93 FF  JSR SECOND
```

NAME: **SETLFS**

ZWECK: **Einrichten einer logischen Datei**
ADRESSE: **$FFBA**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **Keine**

BESCHREIBUNG:

Über diese Routine wird die logische Dateinummer, Geräteadresse und eine Sekundäradresse (Befehlsnummer) für andere KERNAL-Routinen eingerichtet.

Die logische Dateinummer wird vom System als eine Art Schlüssel für die durch die OPEN-Routine erstellte Dateitabelle benutzt. Für die Geräteadressen stehen die Zahlen 0 bis 31 zur Verfügung. Die entsprechenden Codes der Peripheriegeräte beim Commodore PLUS/4 lauten wie folgt:

ADRESSE:	**VORRICHTUNG:**
0	Tastatur
1	Kassettenrekorder
2	RS-232C
3	Bildschirm
4	Drucker am seriellen Bus
8	Floppy " " "

Eine Gerätenummer von 4 oder darüber bezieht sich automatisch auf Geräte am seriellen Bus.

Ein Gerätebefehl wird als Sekundäradresse über den seriellen Bus übertragen, nachdem die Gerätenummer während des seriellen Handshakes übertragen wurde. Wird keine Sekundäradresse übertragen, dann muß das Y-Register auf 255 gesetzt werden.

VORGEHEN:

1) Logische Dateinummer in den Akkumulator laden.

2) Gerätenummer ins X-Register laden.

3) Befehl ins Y-Register laden.

4) Diese Routine aufrufen.

BEISPIEL:

```
$1010                                ; SETZE LOGISCHEN FILE 32,
$1010                                ; GERÄTE-NR. 4 UND KEIN BEFEHL
$1010  A9 20     LDA #32
$1012  A2 04     LDX #4
$1014  A0 FF     LDY #255
$1016  20 BA FF  JSR SETLFS
```

NAME: **SETMSG**

ZWECK: **Ausgabe von Systemmeldungen**
ADRESSE: **$FF90**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A**

BESCHREIBUNG:

Durch diese Routine werden Anzeigen von Fehler- und Steuermeldungen über das KERNAL gesteuert. Bei Aufruf der Routine werden je nach Inhalt des Akkumulators Fehler oder Steuermeldungen angezeigt. Eine Fehlermeldung ist z.B. FILE NOT FOUND. PRESS PLAY ON TAPE ist z.B. eine Steuermeldung.

VORGEHEN:

1) Den Akkumulator auf den gewünschten Wert setzen.

2) Diese Routine aufrufen.

BEISPIEL:

```
$1010  A9 40     LDA #$40          ; STEUERMELDUNGEN EIN
$1012  20 90 FF  JSR SETMSG
$1015  A9 80     LDA #$80          ; FEHLERMELDUNGEN EIN
$1017  20 90 FF  JSR SETMSG
$101A  A9 00     LDA #0            ; ALLE KERNAL-MELDUNGEN AUS
$101C  20 90 FF  JSR SETMSG
```

NAME: **SETNAM**

ZWECK: **Festlegen des Dateinamens**
ADRESSE: **$FFBD**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **Keine**
STACKBEDARF: **Keiner**
REGISTER: **Keine**

BESCHREIBUNG:

Über diese Routine werden die Dateinamen für die Routinen OPEN, SAVE oder LOAD festgelegt. Die Dateinamenlänge wird in den Akkumulator geladen. Die Adresse des Dateinamens wird in die Register X und Y geladen. Für den 7501 gilt standardmäßig das Format niederwertiges Byte/höherwertiges Byte. Die Adresse kann eine beliebige gültige Systemspeicheradresse sein, ab der der Dateiname als String gespeichert ist. Wird kein Dateiname gewünscht, wird der Akkumulator auf 0 gesetzt. In diesem Fall können die Register X und Y auf eine beliebige Speicheradresse gesetzt werden.

VORGEHEN:

1) Länge des Dateinamens in den Akkumulator laden.

2) Niederwertiges Adreßbyte des Dateinamens ins X-Register laden.

3) Höherwertiges " " " " Y-Register "

4) Diese Routine aufrufen.

BEISPIEL:

```
$1010  A9 04     LDA #NAME1-NAME  ; LADE LÄNGE DES DATEINAMENS
$1012  A2 1A     LDX #<NAME       ; LADE ADRESSE DES DATEINAMENS
$1014  A0 10     LDY #>NAME
$1016  20 BD FF  JSR SETNAM
$1019  RTS
```

```
$101A  54 45 43  .BYT "TEST"
$101D  54
```

NAME: **SETTIM**

ZWECK: **Systemtaktgeber setzen**
ADRESSE: **$FFDB**
ÜBERGABE: **A, X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **Keine**

BESCHREIBUNG:

Eine Uhr wird alle 1/60 s (=1 "Jiffy") von einer Interrupt-Routine aktualisiert. Die Uhr ist 3 Bytes "lang", so daß sie bis zu 5.184.000 "Jiffies" (24 Stunden) anzeigen kann. Dann erfolgt die Rückstellung auf 0. Vor dem Aufruf dieser Routine muß in den Akkumulator das signifikanteste (höchste) Byte, ins X-Register das nächste signifikante Byte und ins Y-Register das am wenigsten signifikante Byte der Ausgangs-Zeiteinstellung (in Jiffies) eingegeben werden.

VORGEHEN:

1) Das signifikanteste Byte der 3-Byte-Zahl in den Akkumulator laden.

2) Das nächste Byte ins X-Register laden.

3) Das am wenigsten signifikante Byte ins Y-Register laden.

4) Diese Routine aufrufen.

BEISPIEL:

```
$1010                              ; SETZE DIE UHR AUF 10 MINUTEN
$1010                              ; (=3600 JIFFIES)
$1010  A9 00     LDA #0            ; SIGNIFIKANTESTES BYTE
$1012  A2 0E     LDX #>3600        ; NÄCHSTES BYTE
```

```
$1014  A0 10     LDY #<3600        ; NIEDERWERTIGSTES BYTE
$1016  20 DB FF  JSR SETTIM
```

NAME: **SETTMO**

ZWECK: **Setzen des Timeout-Flags für den IEC-Bus**
ADRESSE: **$FFA2**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **Keine**

BESCHREIBUNG:

Diese Routine wird ausschließlich mit einer zusätzlichen IEC-Bus-Karte benutzt und ist somit für den Commodore PLUS/4 praktisch ohne Bedeutung.

Durch diese Routine wird das Timeout-Flag für den IEC-Bus gesetzt. Ist dieses Kennzeichen gesetzt, dann wartet der Commodore PLUS/4 64 ms auf die Meldung eines Gerätes am IEC-Bus. Antwortet das Gerät nicht auf das DAV-Signal (gültige Datenadresse) des Commodore PLUS/4 innerhalb dieses Zeitraums, dann erkennt der Computer eine Fehlerbedingung und verläßt die Handshake-Sequenz. Ist beim Aufruf dieser Routine Bit 7 im Akkumulator auf 0 gesetzt, dann sind Timeouts wirksam. Entsprechend sind Timeouts unwirksam, wenn Bit 7 auf 1 gesetzt ist.

VORGEHEN (Setzen des Timeout-Flags):

1) Bit 7 des Akkumulators auf 0 setzen.

2) Diese Routine aufrufen.

VORGEHEN (Löschen des Timeout-Flags):

1) Bit 7 des Akkumulators auf 1 setzen.

2) Diese Routine aufrufen.

BEISPIEL:

```
$1010                                ; TIMEOUT-FLAG SETZEN
$1010  A9 00      LDA #0
$1012  20 A2 FF   JSR SETTMO
```

NAME: **STOP**

ZWECK: **Abfrage der STOP-Taste**
ADRESSE: **$FFE1**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **Keiner**
REGISTER: **A, X**

BESCHREIBUNG:

Wurde während eines UDTIM-Aufrufs die STOP-Taste gedrückt, dann wird nach Aufruf dieser Routine das Zero-Flag gesetzt. Darüber hinaus werden die Kanäle auf die Standardwerte zurückgesetzt. Alle anderen Flags bleiben unverändert. Wurde die STOP-Taste nicht gedrückt, enthält der Akkumulator 1 Byte, das die letzte Reihe der Tastatur-Abfrage wiedergibt. Auf diese Weise kann der Bediener auch prüfen, ob bestimmte andere Tasten gedrückt wurden.

VORGEHEN:

1) Vor dieser Routine muß UDTIM aufgerufen werden.

2) Diese Routine aufrufen.

3) Auf Zero-Flag hin überprüfen.

BEISPIEL:

```
$1010  20 EA FF  JSR UDTIM        ; PRÜFE AUF STOP
$1013  20 E1 FF  JSR STOP
$1016  F0 F8     BNE LOOP         ; AUF STOP-TASTE WARTEN
```

NAME: **TALK**

ZWECK: **TALK-Befehl für ein Gerät am seriellen Bus**
ADRESSE: **$FFB4**
ÜBERGABE: **A**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **8**
REGISTER: **A**

BESCHREIBUNG:

Um mit dieser Routine zu arbeiten, muß zunächst eine Gerätenummer zwischen 0 und 31 in den Akkumulator geladen werden. Nach dem Aufruf wird dann Bit für Bit durch diese Routine ODER-verknüpft, um die Gerätenummer in eine TALK-Adresse umzuwandeln. Dieses Datum wird dann als Befehl über den seriellen Bus übertragen.

VORGEHEN:

1) Gerätenummer in den Akkumulator laden.

2) Diese Routine aufrufen.

BEISPIEL:

```
$1010                              ; TALK-BEFEHL FÜR GERÄTE-NR. 4
$1010  A9 04     LDA #4
$1012  20 B4 FF  JSR TALK
```

NAME: **TKSA**

ZWECK: **Übertragen einer Sekundäradresse zu einem Gerät, das den TALK-Befehl erhalten hat**
ADRESSE: **$FF96**
ÜBERGABE: **A**
VORBEREITUNG: **TALK**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **8**
REGISTER: **A**

BESCHREIBUNG:

Diese Routine überträgt eine Sekundäradresse über den seriellen Bus zu einem TALK-Gerät. Beim Aufruf dieser Routine muß im Akkumulator eine Zahl zwischen 0 und 31 geladen sein. Diese Zahl wird dann als Sekundäradreßbefehl über den seriellen Bus gesandt. Zuvor ist unbedingt die TALK-Routine aufzurufen. TKSA ist nicht nach LISTEN wirksam.

VORGEHEN:

1) TALK-Routine benutzen.

2) Sekundäradresse in den Akkumulator laden, geODERt mit 96.

3) Diese Routine aufrufen.

BEISPIEL:

```
$1010                                ; GERÄT NR. 4 MIT BEFEHL 7
$1010  A9 04     LDA #4
$1012  20 B4 FF  JSR TALK
$1015  A9 07     LDA #7
$1017  49 60     EOR #96
$1019  20 96 FF  JSR TKSA
```

NAME: **UDTIM**

ZWECK: **Aktualisierung des Systemtaktgebers**
ADRESSE: **$FFEA**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A, X**

BESCHREIBUNG:

Über diese Routine wird die Systemuhr aktualisiert. Normalerweise wird sie alle 1/60 s von der normalen KERNAL-Interrupt-Routine aufgerufen. Arbeitet das Benutzer-Programm mit eigenen Interrupts, dann muß zur Zeitaktualisierung diese Routine aufgerufen werden.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 EA FF  JSR UDTIM
```

NAME: **UNLSN**

ZWECK: **Übertragung eines UNLISTEN-Befehls**
ADRESSE: **$FFAE**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMEDLUNG: **Siehe READST**
STACKBEDARF: **8**
REGISTER: **A**

BESCHREIBUNG:

Über diese Routine erhalten alle Geräte am seriellen Bus den Befehl, den Datenempfang vom Commmodore PLUS/4 zu beenden. Durch Aufruf dieser Routine wird ein UNLISTEN-Befehl über den seriellen Bus übertragen. Hierbei werden nur die Geräte beeinflußt, die zuvor einen LISTEN-Befehl erhalten haben. Normalerweise wird diese Routine benutzt, nachdem der Commodore PLUS/4 die Datenübertragung zu einem externen Gerät beendet hat. Nach dem UNLISTEN-Befehl sind die Geräte nicht mehr an den seriellen Bus angeschlossen und stehen für andere Zwecke zur Verfügung.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 AE FF  JSR UNLSN
```

NAME: **UNTLK**

ZWECK: **Übertragung eines UNTALK-Befehls**
ADRESSE: **$FFAB**
ÜBERGABE: **Keine**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Siehe READST**
STACKBEDARF: **8**
REGISTER: **A**

BESCHREIBUNG:

Über diese Routine wird ein UNTALK-Befehl über den seriellen Bus übertragen. Alle Befehle, die zuvor einen TALK-Befehl erhalten hatten, beenden die Datenübertragung.

VORGEHEN:

1) Diese Routine aufrufen.

BEISPIEL:

```
$1010  20 AB FF  JSR UNTLK
```

NAME: **VECTOR**

ZWECK: **Verwaltung der RAM-Vektoren**
ADRESSE: **$FF8D**
ÜBERGABE: **X, Y**
VORBEREITUNG: **Keine**
FEHLERMELDUNG: **Keine**
STACKBEDARF: **2**
REGISTER: **A, X, Y**

BESCHREIBUNG:

Diese Routine verwaltet alle im RAM gespeicherten Sprungvektoren. Ist beim Aufruf dieser Routine das Akkumulator-Übertragsbit gesetzt, dann wird der derzeitige Inhalt der RAM-Vektoren in einer Liste gespeichert, deren Adresse durch die Inhalte der Register X und Y gegeben ist.

Beim Arbeiten mit dieser Routine ist äußerste Vorsicht geboten. Zunächst sollte der gesamte Vektor-Inhalt in den Benutzerbereich gelesen, die gewünschten Vektoren geändert und danach dieser Inhalt zurück in die Systemvektoren übertragen werden.

VORGEHEN (Lesen der System-RAM-Vektoren):

1) Übertrag-Flag setzen.

2) Register X und Y auf die gewünschte Vektor-Adresse zeigen lassen.

3) Diese Routine aufrufen.

VORGEHEN (Laden der System-RAM-Vektoren):

1) Übertrag-Flag löschen.

2) Register X und Y auf die zu ladende RAM-Adreß-Vektorliste zeigen lassen.

3) Diese Routine aufrufen.

BEISPIEL:

```
$1010                                ; INPUT-ROUTINE ANDERN
$1010  A2 30     LDX #<USER
$1012  A0 10     LDY #>USER
$1014  38        SEC
$1015  20 8D FF  JSR VECTOR          ; ALTE VEKTOREN LESEN
$1018  A9 00     LDA #<MYINP         ; EIGENE INPUT-ROUTINE
$101A  8D 3A 10  STA USER+10
$101D  A9 20     LDA #>MYINP
$101F  8D 3B 10  STA USER+11
$1022  A2 30     LDX #<USER
$1024  A0 10     LDY #>USER
$1026  18        CLC
$1027  20 8D FF  JSR VECTOR          ; VEKTOREN ANDERN
```

FEHLERMELDUNGEN DES KERNAL

Nachstehend finden Sie eine Liste der Fehlermeldungen, die beim Arbeiten mit den KERNAL-Routinen auftreten können. Kommt es zu einem dieser Fehler, dann wird das Übertragsbit des Akkumulators gesetzt und die Zahl der Fehlermeldungen in den Akkumulator übertragen.

Einige KERNAL-Ein-/Ausgaberoutinen arbeiten nicht mit diesen Codes der Fehlermeldungen. Die Fehler werden stattdessen durch die KERNAL-Routine READST identifiziert.

NUMMER:	BEDEUTUNG:
0	Routine durch STOP-Taste beendet
1	Zu viele offene Dateien
2	Datei bereits offen
3	Datei nicht offen
4	Datei nicht gefunden
5	Gerät nicht vorhanden
6	Keine Eingabe-Datei
7	Keine Ausgabe-Datei
8	Dateiname fehlt
9	Unzulässige Gerätenummer
240	Speicherende verändert (RS-232C)

4.5 SPEICHERPLAN (MEMORY MAP)

ADRESSE (hex)	ADRESSE (dez)	LABEL	BESCHREIBUNG
$0000	0	PDIR	7501 Datenrichtungsregister
$0001	1	PORT	7501 8 Bit I/O-Port
$0002	2	SRCHTK	Flag für Schleifen
$0003-0004	3-4	ZPVEC1	Neue Startadresse (RENUMBER)
$0005-0006	5-6	ZPVEC2	Schrittweite (RENUMBER)
$0007	7	CHARAC	Suchzeichen
$0008	8	ENDCHR	Flag: Suchen nach einem Anführungszeichen am Ende eines Strings
$0009	9	TRMPOS	Bildschirmspalte ab letztem TAB
$000A	10	VERCK	Flag: 0=LOAD, 1=VERIFY
$000B	11	COUNT	Eingabepufferzeiger, Anzahl der Elemente
$000C	12	DIMFLG	Flag: Standard-Felddimensionierung
$000D	13	VALTYP	Datentyp: $FF=String, $00=Numerisch
$000E	14	INTFLG	Datentyp: $80=Integer, $00=Fließkomma
$000F	15	DORES	Flag für DATAs lesen/LIST auflisten
$0010	16	SUBFLG	Flag: Benutzerfunktionsaufruf
$0011	17	INPFLG	Flag: $00=INPUT, $40=GET, $98=READ
$0012	18	TANSGN	Flag: Vorzeichen des TAN/Flag für Gleichheit bei Vergleich
$0013	19	CHANNL	Flag: INPUT-Kommentar
$0014-0015	20-21	LINNUM	Ganzzahliger Wert
$0016	22	TEMPPT	Zeiger: Temporärer Stringstapel
$0017-0018	23-24	LASTPT	Letzte Stringadresse
$0019-0021	25-33	TEMPST	Stapel für temporäre Strings
$0022-0023	34-35	INDEX1	Bereich für Hilfszeiger 1
$0024-0025	36-37	INDEX2	Bereich für Hilfszeiger 2
$0026	38	RESHO	Bereich für Multiplikationsprodukt
$0027	39	RESMOH	" " "
$0028	40	RESMO	" " "
$0029	41	RESLO	" " "
$002A	42		
$002B-002C	43-44	TXTTAB	Zeiger: Anfang BASIC-Text
$002D-002E	45-46	VARTAB	Zeiger: Anfang BASIC-Variablen

$002F-0030	47-48	ARYTAB	Zeiger: Anfang BASIC-Felder
$0031-0032	49-50	STREND	Zeiger: Ende BASIC-Felder (+1)
$0033-0034	51-52	FRETOP	Zeiger: Anfang der Sringspeicherung
$0035-0036	53-54	FRESPC	Hilfszeiger für Strings
$0037-0038	55-56	MEMSIZ	Zeiger: Oberste BASIC-Adresse
$0039-003A	57-58	CURLIN	Derzeitige BASIC-Zeilennummer
$003B-003C	59-60	TXTPTR	Zeiger: Derzeitiges Byte des BASIC-Textes
$003D-003E	61-62	FNDPNT	Zeiger auf BASIC-Statement für CONT
$003F-0040	63-64	DATLIN	Derzeitige DATA-Zeilennummer
$0041-0042	65-66	DATPTR	Zeiger: Derzeitige DATA-Adresse
$0043-0044	67-68	INPPTR	Vektor: INPUT-Routine
$0045-0046	69-70	VARNAM	Derzeitiger BASIC-Variablenname
$0047-0048	71-72	VARPNT	Adresse der aktuellen Variable
$0049-004A	73-74	FORPNT	Variablenzeiger für FOR/NEXT
$004B-004C	75-76	OPPTR	Zwischenzeiger für BASIC-Zeiger/ Daten
$004D	77	OPMASK	Vergleichs-Maske: 1=größer, 2=gleich, 4=kleiner
$004E-004F	78-79	DEFPNT	Zeiger auf Variable einer DEF FN-Funktion
$0050-0051	80-81	DSCPNT	Zeiger auf String-Deskriptor in einer Variablen-Liste
$0052	82		
$0053	83	HELPER	Flag: HELP oder LIST
$0054-0056	84-86	JMPER	Sprungvektor für Funktionen
$0057-0060	87-96	TEMPF1	Temporärer Zeiger, temporärer Gleitpunktakkumulator
$0061	97	FACEXP	Gleitpunktakkumulator #1 : Exponent
$0062-0065	98-101	FACHO	" #1 : Mantisse
$0066	102	FACSGN	" #1 : Vorzei.
$0067	103	SGNFLG	Zeiger: Polynomauswertung
$0068	104	BITS	Gleitpunktakkumulator #1 : Überlauf
$0069	105	ARGEXP	" #2 : Exponent
$006A-006D	106-109	ARGHO	" #2 : Mantisse
$006E	110	ARGSGN	" #2 : Vorzei.
$006F	111	ARISGN	Vorzeichenvergleich Akku #1 mit Akku #2, $00=gleiches Vorzeichen, $FF=unterschiedlich
$0070	112	FACOV	Gleitpunktakkumulator #1 : Niederwertige Stelle (Rundung)

$0071-0072	113-114	FBUFPT	Zeiger: Kassettenbuffer
$0073-0074	115-116	AUTINC	Inkrement beim AUTO-Befehl, 0=Aus
$0075	117	MVDFLG	Flag: Gesetzt, wenn 10K für Grafik reserviert wurden ($FF=gesetzt)
$0076-0078	118-120	KEYNUM	Temporärer Speicher für MID$-Befehl
$0079-007B	121-123	DSDESC	Deskriptor für DS$
$007C-007D	124-125	TOS	Obergrenze des Run-Time-Stacks
$007E-007F	126-127	TMPTON	Arbeitsbereich (Sound)
$0080	128	VOICNO	" "
$0081	129	RUNMOD	Flag für RUN ($00=Nein, $80=Ja)
$0082	130	POINT	Flag für DOS-Befehle
$0083	131	GRAPHM	Derzeitiger Grafik-Modus ($00=Text, $20=Hires, $60=Split Hires,$A0=Multicolour,$E0=Split Multicolour)
$0084	132	COLSEL	Derzeitige ausgewählte Farbe
$0085	133	MC1	Multicolour-Farbe 1
$0086	134	FG	Vordergrundfarbe
$0087	135	SCXMAX	Maximale Anzahl von Spalten
$0088	136	SCYMAX	" " " Zeilen
$0089	137	LTFLAG	Flag: PAINT links
$008A	138	RTFLAG	Flag: " rechts
$008B	139	STOPNB	Flag: " Umrandung ($00=gleiche Farbe, $80=verschiedene Farbe)
$008C-008D	140-141	GRAPNT	Zeiger: Bit Map Farbinformation
$008E	142	VTEMP1	Temporärer Zwischenspeicher
$008F	143	VTEMP2	" "
$0090	144	STATUS	Kernal-Ein-/Ausgabestatuswort: ST
$0091	145	STKEY	Flag: STOP-Taste/RVS-Taste
$0092	146	SPVERR	Temporärer Speicher
$0093	147	VERFCK	Flag: 0=LOAD, 1=VERIFY
$0094	148	C3PO	Flag: serieller Bus - Zeichen im Puffer ($00=Nein, $80=Ja)
$0095	149	BSOUR	Zeichen im Puffer für seriellen Bus
$0096	150	RSAV	Temporärer Speicher für BASIN
$0097	151	LDTND	Anzahl der offenen Dateien / Dateitabellen-Index
$0098	152	DFLTN	Eingabegerät (Standard: 0)
$0099	153	DFLTO	Ausgabegerät (CMD) (Standard: 3)
$009A	154	MSGFLG	Flag: $80=BASIC-Direktmodus, $C0=Maschinenmonitor, $00=Programm
$009B-009C	155-156	SAL	Zeiger: Kassettenpuffer / Bild-

			schirm scrollen
$009D-009E	157-158	EAL	Zeiger: Kassettenende / Programmende
$009F-00A0	159-160	T1	Temporärer Speicher
$00A1-00A2	161-162	T2	" "
$00A3-00A5	163-165	TIME	Echtzeituhr (ca. 1/60 s)
$00A6	166	R2D2	Register für seriellen Bus
$00A7	167	TPBYTE	Register für Kassettenroutine
$00A8	168	BSOUR1	Register für seriellen Bus
$00A9	169	FPVERR	Temporärer Farb-Vektor
$00AA	170	DCOUNT	Register für Kassettenroutine
$00AB	171	FNLEN	Länge des aktuellen Dateinamens
$00AC	172	LA	Logische Dateinummer
$00AD	173	SA	Aktuelle Sekundär-Adresse
$00AE	174	FA	Aktuelle Gerätenummer
$00AF-00B0	175-176	FNADR	Zeiger: Aktueller Dateiname
$00B1	177	ERRSUM	Zähler für Kassettenfehler
$00B2-00B3	178-179	STAL	I/O-Startadresse
$00B4-00B5	180-181	MEMUSS	Basis-Ladeadresse
$00B6-00B7	182-183	TAPEBS	Ladeendadresse (Kassette)
$00B8-00B9	184-185	TMP2	Adresse für VECTOR
$00BA-00BB	186-187	WRBASE	Zeiger auf Zeichen im Kassettenpuffer
$00BC-00BD	188-189	IMPARM	Zeiger auf String für Primms
$00BE-00BF	190-191	FETPTR	Register für Long-Fetch-Routine
$00C0-00C1	192-193	SEDSAL	Register für Scrolling
$00C2	194	RVS	Flag: RVS-Zeichen ($12=Ja,$00=Nein)
$00C3	195	INDX	Zeiger: Ende der logischen Zeile für Eingabe
$00C4-00C5	196-197	LSXP	Cursor X/Y-Position für Eingabe
$00C6	198	SFDX	Flag: Gedrückte Taste, ($40=keine)
$00C7	199	CRSW	Flag: Eingabe INPUT oder GET über die Tastatur
$00C8-00C9	200-201	PNT	Zeiger: Derzeitige Bildschirmzeile
$00CA	202	PNTR	Cursorspalte in derzeitiger Zeile
$00CB	203	QTSW	Flag: Editor im Anführungszeichen-Modus (0=Nein)
$00CC	204	SEDT1	Länge der aktuellen Bildschirmzeile
$00CD	205	TBLX	Zeile, in der sich der Cursor befindet
$00CE	206	DATAX	Letztes Zeichen (I/O)

$00CF	207	INSRT	Flag: INST-Modus, >0=Anzahl der Einfügungen
$00D0-00D7	208-215		Reserviert für Sprachsynthesizer
$00D8-00E8	218-232		" " Anwendungssoftware
$00E9	233	CIRSEG	Arbeitsbereich
$00EA-00EB	234-235	USER	Zeiger: Derzeitiges Farb-RAM des Bildschirms
$00EC-00EE	236-238	KEYTAB	Vektor: Tastatur-Dekodiertab. $E026
$00EF	239	NDX	Anzahl der Zeichen im Tastaturpuffer
$00F0	240	STPFLG	Flag: Bildschirm-Pause (CTRL-S/T)
$00F1-00F2	241-242	TO	Register für Maschinensprache-Monitor
$00F3	243	CHRPTR	Zero-Page-Adresse für Masch. mon.
$00F4	244	BUFEND	" " " " " "
$00F5	245	CHKSUM	Register für Prüfsumme
$00F6	246	LENGTH	
$00F7	247	PASS	
$00F8	248	TYPE	Block-Typ
$00F9	249	USEKDY	(B.7=1) für Write, (B.6=1) für Read
$00FA	250	XSTOP	Register für X bei Stoptastentest
$00FB	251	CURBNK	Aktuelle Bank-Konfiguration
$00FC	252	XON	Zu sendendes Zeichen für X-On
$00FD	253	XOFF	" " " " X-Off
$00FE	254	SEDT2	Arbeitsspeicher des Editor
$00FF	255	LOFBUF	
$0100-010F	256-271	FBUFFR	
$0110	272	SAVEA	Zwischenspeicher für SAVE & RESTORE
$0111	273	SAVEY	" " " " "
$0112	274	SAVEX	" " " " "
$0113-0122	275-289	COLKEY	Farb-/Luminanz-Tabelle im RAM
$0123	290		
$0124-01FF	291-511	SYSSTK	Prozessorstack
$0200-0258	512-600	BUF	System-Eingabepuffer
$0259-025A	601-602	OLDLIN	Vorherige BASIC-Zeilennummer
$025B-025C	603-604	OLDTXT	Zeiger: BASIC-Anweisung für CONT
$025D-02AC	605-684		BASIC-/DOS-Arbeitsbereich

$025D	605	XCNT	DOS-Schleifenzähler
$025E-026D	606-621	FNBUFR	Bereich für Filenamen
$026E	622	DOSF1L	Länge des 1. Filenamens
$026F	623	DOSDS1	DOS (Laufwerk 1)
$0270-0271	624-625	DOSF1A	Adresse des 1. Filenamens
$0272	626	DOSF2L	Länge des 2. Filenamens
$0273	627	DOSFA	DOS (Laufwerk 2)
$0274-0275	628-629	DOSF2A	Adresse des 2. Filenamens
$0276	630	DOSLA	DOS logische Adresse
$0277	631	DOSFA	" Geräteadresse
$0278	632	DOSSA	" Sekundäradresse
$0279-027A	633-634	DODDID	" Disketten-ID
$027B	635	DIDCHK	ID-Flag
$027C	636	DOSSTR	DOS Ausgabepuffer
$027D-02AC	637-684	DOSSPC	" Arbeitsbereich
$02AD-02AE	685-686	XPOS	Aktuelle X-Position des Grafik-Cursor
$02AF-02B0	687-688	YPOS	" Y- " " " "
$02B1-02B2	689-690	XDEST	X-Zielkoordinate
$02B3-02B4	691-692	YDEST	Y- "
$02B5-02B6	693-694	XABS	Absolutwert von X
$02B7-02B8	695-696	YABS	" " Y
$02B9-02BA	697-698	XSGN	Signumwert von X
$02BB-02BC	699-700	YSGN	" " Y
$02BD-02BE	701-702	FCT1	
$02BF-02C0	703-704	FCT2	
$02C1-02C2	705-706	ERRVAL	
$02C3	707	LESSER	
$02C4	708	GREATR	
$02C5	709	ANGSGN	Vorzeichen des Winkels
$02C6-02C7	710-711	SINVAL	Sinus des Winkels
$02C8-02C9	712-713	COSVAL	Cosinus des Winkels
$02CA-02CB	714-715	ANGCNT	Temporäres Register für Winkelabstand
$02CD	717	BNR	Zeiger auf Beginn (PRINT USING)
$02CE	718	ENR	Zeiger auf Ende
$02CF	719	DOLR	Flag: Dollar
$02D0	720	FLAG	Flag: Komma
$02D1	721	SWE	Zähler
$02D2	722	USGN	Vorzeichen des Exponents

$02D3	723	UEXP	Zeiger auf Exponent
$02D4	724	VN	Anzahl der Zahlen vor dem Dezimal-punkt
$02D5	725	CHSN	Flag: Justierung
$02D6	726	VF	Anzahl der Positionen vor dem Dezi-malpunkt
$02D7	727	NF	Anzahl der Positionen nach dem Dezimalpunkt
$02D8	728	POSP	Flag: +/- (Feld)
$02D9	729	FESP	Flag: Exponent (Feld)
$02DA	730	ETOF	Schalter
$02DB	731	CFORM	Zeichenzähler (Feld)
$02DC	732	SNO	Vorzeichennummer
$02DD	733	BLFD	Flag: Leertaste/*
$02DE	734	BEGFD	Zeiger auf Anfang des Feldes
$02DF	735	LFOR	Länge des Formatstrings
$02E0	736	ENDFD	Zeiger auf Ende des Feldes

$02CC-02CD	716-717	XCENTR	
$02CE-02CF	718-719	YCENTR	
$02D0-02D1	720-721	XDIST1	
$02D2-02D3	722-723	YDIST1	
$02D4-02D5	724-725	XDIST2	
$02D6-02D7	726-727	YDIST2	
$02D8-02D9	728-729	DISEND	
$02DA	730	COLCNT	Spaltenzähler bei CHAR-Befehl
$02DB	731	ROWCNT	Zeilenzähler " " - "
$02DC	732	STRCNT	Für CHAR-Befehl

$02CC-02CD	716-717	XCORD1	Punkt 1 X-Koordinate
$02CE-02CF	718-719	YCORD1	" " Y- "
$02D0-02D1	720-721	BOXANG	Drehwinkel
$02D2-02D3	722-723	XCOUNT	
$02D4-02D5	724-725	YCOUNT	
$02D6-02D7	726-727	BXLENG	Länge einer Seite
$02D8-02D9	728-729	XCORD2	Punkt 2 X-Koordinate
$02DA-02DB	730-731	YCORD2	" " Y- "

$02CC-02CD	716-717	XCIRCL	Kreismitte, X-Koordinate
$02CE-02CF	718-719	YCIRCL	" , Y- "
$02D0-02D1	720-721	XRADUS	X-Radius

$02D2-02D3	722-723	YRADUS	Y- "
$02D4-02D5	724-725	ROTANG	Drehwinkel
$02D6-02D7	726-727		
$02D8-02D9	728-729	ANGBEG	Kreissegment-Winkel Start
$02DA-02DB	730-731	ANGEND	" " Ende
$02DC-02DD	732-733	XRCOS	X-Radius * Cosinus des Drehwinkels
$02DE-02DF	734-735	YRSIN	Y- " * Sinus " "
$02E0-02E1	736-737	XRSIN	X- " * " " "
$02E2-02E3	738-739	YRCOS	Y- " * Cosinus " "
$02CD	717	KEYLEN	
$02CE	718	KEYNXT	
$02CF	719	STRSZ	Länge der Stringvariablen (SHAPE)
$02D0	720	GETTYP	Shape-Modus setzen
$02D1	721	STRPTR	Zähler für Stringposition
$02D2	722	OLDBYT	Altes Bitmap-Byte
$02D3	723	NEWBYT	Variable für neuen String oder Bitmap-Byte
$02D4	724		
$02D5-02D6	725-726	XSIZE	Länge des Shapes in X-Richtung
$02D7-02D8	727-728	YSIZE	" " " " Y- "
$02D9-02DA	729-730	XSAVE	Temporärer Speicher für XSIZE
$02DB-02DC	731-732	STRADR	Speicher für Shape/String-Deskriptor
$02DD	733	BITIDX	Zeiger auf ein Bit in einem Byte
$02DE-02E1	734-737	SAVSIZ	Temporärer Speicher
$02E4	740	CHRPAG	High-Byte einer Character-ROM-Adresse für CHAR-Befehl
$02E5	741	BITCNT	Register für GSHAPE
$02E6	742	SCALEM	Flag: SCALE-Modus ($00=Aus)
$02E7	743	WIDTH	Flag: Doppelte Pixel-Größe
$02E8	744	FILFLG	Flag: Ausmalen eines Rechtecks (BOX-Befehl)
$02E9	745	BITMSK	Temporärer Speicher für Bitmaske
$02EA	746	NUMCNT	Länge des Strings
$02EB	747	TRCFLG	Flag: TRACE-Modus ($00=Aus)
$02EC	748	T3	Zwischenspeicher für DIRECTORY
$02ED-02EE	749-750	T4	" " "
$02EF	751	VTEMP3	Temporärer Speicher für Grafik
$02F0	752	VTEMP4	Anzahl der Grafik-Parameter

$02F1	753	VTEMP5	Parameter: $01=Relativ, $00=Absolut
$02F2-02F3	754-755	ADRAY1	Zeiger auf Konvertierungsroutine Gleitkomma nach Integer
$02F4-02F5	756-757	ADRAY2	Zeiger auf Konvertierungsroutine Integer nach Gleitkomma
$02FE-02FF	766-767	BNKVEC	Vektor für Funktions-Modul

$0300-0301	768-769	IERROR	Vektor: BASIC-Fehlermeldung	$8686
$0302-0303	770-771	IMAIN	" " -Warmstart	$8712
$0304-0305	772-773	ICRNCH	" " -Token generier.	$8956
$0306-0307	774-775	IQPLOP	" " -Text listen	$8B6E
$0308-0309	776-777	IGONE	" " -Befehl ausführ.	$8BD6
$030A-030B	778-779	IEVAL	" " -Tokenauswertung	$9417
$030C-030D	780-781	IESCLK	" User-Tokengenerierung	$896A
$030E-030F	782-783	IESCPR	" Keywort erzeugen	$8B88
$0310-0311	784-785	IESCEX	" User-Token bearbeiten	$8C8B
$0312-0313	786-787	ITIME	" Interrupt (Uhr)	$CE42
$0314-0315	788-789	CINV	" Hardware-Interrupt	$CE0E
$0316-0317	790-791	CBINV	" BRK-Interrupt	$F44C
$0318-0319	792-793	IOPEN	" Kernal-OPEN-Routine	$EF53
$031A-031B	794-795	ICLOSE	" " CLOSE "	$EE5D
$031C-031D	796-797	ICHKIN	" " CHKIN "	$ED18
$031E-031F	798-799	ICKOUT	" " CHKOUT "	$ED60
$0320-0321	800-801	ICLRCH	" " CLRCHN "	$EF0C
$0322-0323	802-803	IBASIN	" " CHRIN "	$EBE8
$0324-0325	804-805	IBSOUT	" " CHROUT "	$EC4B
$0326-0327	806-807	ISTOP	" " STOP "	$F265
$0328-0329	808-809	IGETIN	" " GETIN "	$EBD9
$032A-032B	810-811	ICLALL	" " CLALL "	$EF08
$032C-032D	812-813	USRCMD	" Monitor-Break	$F44C
$032E-032F	814-815	ILOAD	" Kernal-LOAD-Routine	$F04A
$0330-0331	816-817	ISAVE	" " SAVE "	$F1A4
$0332-03F2	818-1010	TAPBUF	Kassettenbuffer	
$0332	818		Filetyp (0=BASIC, 1=Maschinenpgm.)	
$0333-0334	819-820		Startadresse (low/high)	
$0335-0336	821-822		Endadresse (low/high)	
$0337-0346	823-838		Programmname (16 Zeichen)	
$03F3-03F4	1011-1012	WRLEN	Datenzähler (Write)	
$03F5-03F6	1013-1014	RDCNT	" (Read)	

$03F7-0436	1015-1078	INPQUE	RS-232 Eingabe-Puffer (64 Bytes)
$0437-0454	1079-1108	ESTAKL	Kassetten-Fehler-Stack (Low Bytes)
$0455-0472	1109-1138	ESTAKH	" " " (High ")
$0473-0478	1139-1144	CHRGET	Unterroutine: nächstes Byte vom BASIC-Text lesen
$0479-0484	1145-1156	CHRGOT	Erneutes Lesen des gleichen Text-Bytes
$0485-0493	1157-1171	QNUM	
$0494-04A1	1172-1185	INDSUB	Unterprogramm zum Laden aus beliebiger Bank
$04A2-04A4	1186-1188	ZERO	Numerische Konstante für BASIC
$04A5-04AF	1189-1199	INDTXT	Textpointer
$04B0-04BA	1200-1210	INDIN1	Index & Index 1
$04BB-04C5	1211-1221	INDIN2	Index 2
$04C6-04D0	1222-1232	INDST1	String 1
$04D1-04DB	1233-1243	INDLOW	Lowtr
$04DC-04E6	1244-1254	INDFMO	Facmo
$04E7	1255	PUFILL	Füllzeichen bei PRINT USING (Standard: Leertaste)
$04E8	1256	PUCOMA	Kommasymbol
$04E9	1257	PUDOT	Punktsymbol
$04EA	1258	PUMONY	Währungszeichen
$04EB-04EE	1259-1262	TMPDES	Temporärer Speicher für INSTR
$04EF	1263	ERRNUM	Letzte Fehlernummer
$04F0-04F1	1264-1265	ERRLIN	Zeilennummer, in der der letzte Fehler auftrat ($FFFF=Kein Fehler)
$04F2-04F3	1266-1267	TRAPNO	Verweis auf Zeilennummer für ON ERROR GOTO
$04F4	1268	TMPTRP	Temporäres Register für TRAP
$04F5-04F6	1269-1270	ERRTXT	Zwischenspeicher für TRAP
$04F7	1271	OLDSTK	BASIC-Textzeiger auf letzten Fehler
$04F8-04F9	1272-1273	TMPTXT	DO-Speicher vom BASIC-Textzeiger
$04FA-04FB	1274-1275	TMPLIN	DO-Speicher der Zeilennummer
$04FC-04FD	1276-1277	MTIMLO	Low-Byte von Sound 1/2-Länge
$04FE-04FF	1278-1279	MTIMHI	High- " " " " "

$0500	1280	USRPOK	USR-Sprungbefehl
$0501-0502	1281-1282	USRADD	USR-Adresse (low/high)

$0503-0507	1283-1287	RNDX	Startwert für RND
$0508	1288	DEJAVU	Flag: Kalt- oder Warmstart
$0509-0512	1289-1298	LAT	Tabelle der logischen Filenummern
$0513-051C	1299-1308	FAT	" " Gerätenummern
$051D-0526	1309-1318	SAT	" " Sekundäradressen
$0527-0530	1319-1328	KEYD	Tastaturpuffer (FIFO)
$0531-0532	1329-1330	MEMSTR	Zeiger: Startadresse des RAM für Betriebssystem
$0533-0534	1331-1332	MSIZ	Zeiger: Ende des RAM für Betriebssystem
$0535	1333	TIMOUT	Flag: Timeout (Zeitüberschreitung) vom (optionalen) IEC-Bus
$0536	1334	FILEND	1=Ende des Files erreicht, sonst 0
$0537	1335	CTALLY	Anzahl der Zeichen im Puffer (für Read und Write)
$0538	1336	CBUFVA	Anzahl der insgesamt gültigen Zeichen im Buffer (nur Read)
$0539	1337	TPTR	Pointer auf das nächste Zeichen im Puffer (für Read und Write)
$053A	1338	FLTYPE	Typ des Kassettenfiles
$053B	1339	COLOR	Aktives Attribut-Byte (Farbe, Helligkeit, Blinken)
$053C	1340	FLASH	Flag: Zeichen blinkt ($00=Nein)
$053D	1341		unbenutzt
$053E	1342	HIBASE	Video-RAM Anfang (Page)
$053F	1343	XMAX	Größe des Tastaturpuffers
$0540	1344	RPTFLG	Flag: Tastenwiederholung ($80=Alle, $40=Keine, $00=Nur DEL, CRSR, SPC)
$0541	1345	KOUNT	Zählgeschwindigkeit für Wiederholen
$0542	1346	DELAY	Zähler für Wiederholungsverzögerung
$0543	1347	SHFLAG	Flag: Tasten SHIFT, CTRL, C=
$0544	1348	LSTSHF	Letztes SHIFT-Muster der Tastatur
$0545-0546	1349-1350	KEYLOG	Zeiger auf Tastatur-Decodiertabelle
$0547	1351	MODE	Flag: $80=SHIFT unwirksam, $00=Wirksam
$0548	1352	AUTODN	Flag: Automatisches Scrollen (abwärts), 0=Ein, <>0=Aus
$0549	1353	LINTMP	Zwischensp. während Bildschirmausg.
$054A	1354	ROLFLG	" " "
$054B	1355	FORMAT	Arbeitsspeicher vom Maschinensprachemonitor

$054C-054E	1356-1358	MSAL	Für Assembler
$054F	1359	WRAP	Temporärer Speicher für Assembler
$0550	1360	TMPC	" " " "
$0551	1361	DIFF	" " " "
$0552	1362	PCH	Program-Counter (high)
$0553	1363	PCL	" " (low)
$0554	1364	FLGS	Prozessor-Flags
$0555	1365	ACC	Akkumulator des Prozessors
$0556	1366	XR	X-Register " "
$0557	1367	YR	Y- " " "
$0558	1368	SP	Stack-Pointer " "
$0559	1369	INVL	Speicher für Maschinen-Monitor
$055A	1370	INVH	" " " "
$055B	1371	CMPFLG	" " " "
$055C	1372	BAD	Wird von verschiedenen Monitor-Routinen benutzt
$055D	1373	KYNDX	Zähler für Zeichen einer Funktionstaste
$055E	1374	KEYIDX	Zeiger auf String einer Funktionstaste
$055F-0566	1375-1382	KEYBUF	Länge der Strings auf den Funktionstasten
$0567-05E6	1383-1510	PKYBUF	Speicher für Strings der Funktionstasten
$05E7	1511	KDATA	Temporärer Speicher für Daten, die nach Kennedy geschrieben werden
$05E8	1512	KDYCMD	Auswahl, ob Kennedy Read oder Write
$05E9	1513	KDYNUM	Kennedy`s Gerätenummer
$05EA	1514	KDYPRS	$FF=Kennedy präsent, sonst $00
$05EB	1515	KDYTYP	Temporärer Speicher für Kennedy
$05EC-06EB	1516-1771	SAVRAM	1 Page, die von Banking-Routinen benutzt wird
$05EC-05EF	1516-1519	PAT	Physikalische Adressen (Tabelle)
$05F0-05F1	1520-1521	LNGJMP	Long-Jump (Adresse)
$05F2	1522	FETARG	" " (Akkumulator)
$05F3	1523	FETXRG	" " (X-Register)
$05F4	1524	FETSRG	" " (Status-Register)
$05F5-065D	1525-1629	AREAS	RAM-Bereich für Bank-Switching
$065E-06EB	1630-1771	APECH	" " " Sprachsynthesizer

$06EC-07AF	1772-1967	STKTOP	Pseudo-Stack für BASIC-Interpreter
$07B0	1968	WROUT	Byte, das auf Kassette geschrieben werden soll
$07B1	1969	PARITY	Temporärer Speicher für Parity
$07B2	1970	TT1	" " " Kopf schr.
$07B3	1971	TT2	" " " " "
$07B5	1973	RDBITS	Lokaler Index für READBYTE-Routine
$07B6	1974	ERRSP	Pointer auf den Error-Stack
$07B7	1975	FPERRS	Anzahl der Errors beim 1. Pass
$07B8-07B9	1976-1977	DSAMP1	Zeitkonstante
$07BA-07BB	1978-1979	DSAMP2	"
$07BC-07BD	1980-1981	ZCELL	"
$07BE	1982	SRECOV	Stack-Markierung für STOP-Taste
$07BF	1983	DRECOV	" " " DROP- "
$07C0-07C3	1984-1987	TRSAVE	Nach RDBLOK übergebene Parameter
$07C4	1988	RDETMP	Temporärer Speicher für RDBLOK
$07C5	1989	LDRSCN	Anzahl von Shorts im Kopf
$07C6	1990	CDERRM	" der Fehler beim RD Countdown
$07C7	1991	VSAVE	Temporärer Speicher für VERIFY
$07C8-07CB	1992-1995	T1PIPE	Temporärer Speicher für T1
$07CC	1996	ENEXT	Read Error Propagata
$07CD	1997	UOUTQ	Zu sendendes User-Zeichen (RS-232)
$07CE	1998	UOUTFG	0=Leer, 1=Voll
$07CF	1999	SOUTQ	Zu sendendes System-Zeichen
$07D0	2000	SOUNFG	0=Leer, 1=Voll
$07D1	2001	INQFPT	Pointer auf Anfang Eingabepuffer
$07D2	2002	INQRPT	" " Ende "
$07D3	2003	INQCNT	Anzahl der Zeichen im Eingabepuffer
$07D4	2004	ASTAT	Temporärer Status der ACIA
$07D5	2005	AINTMP	" Speicher für INPUT
$07D6	2006	ALSTOP	Flag: Lokale Pause
$07D7	2007	ARSTOP	" Ferngesteuerte Pause
$07D8	2008	APRES	" Kennzeichnet Präsenz der ACIA
$07D9-07E4	2009-2020	KLUDES	Indirekte Routine (downloaded)
$07E5	2021	SCBOT	Bildschirmfenster: Unterer Rand
$07E6	2022	SCTOP	" Oberer "
$07E7	2023	SCLF	" Linker "
$07E8	2024	SCRT	" Rechter "
$07E9	2025	SCRDIS	
$07EA	2026	INSFLG	Flag: Automatisches Einfügen
$07EB	2027	LSTCHR	Letztes ausgegebenes Zeichen

$07EC	2028	LOGSCR	Speicher für Bildschirmverwaltung
$07ED	2029	TCOLOR	Temporäres Register für die Farbe bei INST und DEL
$07EE-07F1	2030-2033	BITABL	Zeilen-Link-Tabelle für Bildschirm
$07F2	2034	SAREG	Akkumulator retten bei SYS-Befehl
$07F3	2035	SXREG	X-Register " " " "
$07F4	2036	SYREG	Y- " " " " "
$07F5	2037	SPREG	SP- " " " " "
$07F6	2038	LSTX	Derzeit gedrückte Taste: CHR$(n), $40=Keine Taste
$07F7	2039	STPDSB	Flag: CTRL-S (0=Offen, 6=Gesperrt)
$07F8	2040	RAMROM	RAM/ROM-Umschaltung für Maschinensprachemonitor ($00=ROM, $80=RAM)
$07F9	2041	COLSW	RAM/ROM-Umschaltung für Farb-/ Luminanztabelle ($00=ROM, $80=RAM)
$07FA	2042	FFRMSK	ROM-Maske für geteilten Bildschirm
$07FB	2043	VMBMSK	Video-RAM-Maske für " "
$07FC	2044	LSEM	Motorsteuerung Kassettenrekorder
$07FD	2045	PALCNT	Hilfszähler für Echtzeituhr bei PAL-System
$07FE-07FF	2046-2047		Unbenutzt

HIRES-GRAFIK WURDE NOCH NICHT AUFGERUFEN :

$0800-0BE7	2048-3047	TEDATR	Farb-RAM (Textmodus)
$0C00-0FE7	3072-4071	TEDSCN	Video-RAM (Textmodus)
$1000-3FFF	4096-16383	BASBGN	BASIC-RAM bei C-116, C-16
$1000-7FFF	4096-32767	BASBGN	" " " " " + 16K RAM
$1000-FCFF	4096-64767	BASBGN	" " " " " + 64K RAM
			oder bei PLUS/4

HIRES-GRAFIK WURDE BEREITS AUFGERUFEN :

$1800-1BE7	6144-7143	TEDATR	Luminanztabelle (Grafikmodus)
$1C00-1FE7	7168-8167	TEDSCN	Farbtabelle (Grafikmodus)
$2000-3F3F	8192-16191	GRBASE	Bitmap-Grafikbildschirm
$1000-17FF	4096-6143	BASBGN	BASIC-RAM bei C-116, C-16
$4000-7FFF	16384-32767	BASBGN	" " " " " + 16K RAM
$4000-FCFF	16384-64767	BASBGN	" " " " " + 64K RAM

oder bei PLUS/4

$8000-CFFF 32768-53247 BASIC-Interpreter

$D000-D3FF 53248-54271 Character-ROM (Groß-/Grafikzeichen)
$D400-D7FF 54272-55295 " " (Klein-/Großzeichen)

$D800-FBFF 55296-64511 Betriebssystem

$FC00-FCFF 64512-64767 ROM-Banking-Routinen

$FD00-FD0F 64768-64783 ACIA (Nur PLUS/4)
$FD10-FD1F 64784-64799 6529 Parallel Port (Nur PLUS/4)
$FDD0-FDDF 64976-64991 Modul Bank Port
$FE00-FEFF 65024-65279 DMA Disk System

TED-CHIP ($FF00-FF3F)

REG.	ADRESSE	ADRESSE	BITS	BESCHREIBUNG
0	$FF00	65280		Timer 1, Reload Low
1	$FF01	65281		Timer 1, Reload High
2	$FF02	65282		Timer 2, Low
3	$FF03	65283		Timer 2, High
4	$FF04	65284		Timer 3, Low
5	$FF05	65285		Timer 3, High
6	$FF06	65286	0-2	Vertikales Scroll-Position (Y)
			3	Wahl von 24/25 Zeilen (1=25)
			4	Bildschirm abschalten
			5	Bitmap-Modus einschalten (1=Ein)
			6	Extended Color-Modus einsch. (1=Ein)
			7	Testbit (muß immer 0 sein)
7	$FF07	65287	0-2	Horizontale Scroll-Position (X)
			3	Wahl von 38/40 Spalten (1=40)
			4	Multicolour-Modus einschalt. (1=Ein)
			5	Freeze-Modus einschalten (1=Ein)
			6	PAL/NTSC-Modus (0=PAL, 1=NTSC)
			7	RVS-Video (0=Hardware, 1=Software)
8	$FF08	65288		Tastatur-Matrix
9	$FF09	65289		Interrupt-Quellen
			0	Nicht benutzt
			1	Raster-Interrupt
			2	(Light Pen, beim PLUS/4 nicht mögl.)
			3	Timer 1 Interrupt
			4	" 2 "
			5	Nicht benutzt
			6	Timer 3 Interrupt

			7	Interrupt Bit
10	FF0A	65290		Interrupt-Maskierung
			0	Bit 8 vom Raster-Vergleich (Reg. 11)
			1	Raster-Interrupt
			2	(Light Pen, beim PLUS/4 nicht mögl.)
			3	Timer 1 Interrupt
			4	" 2 "
			5	Nicht benutzt
			6	Timer 3 Interrupt
			7	Nicht benutzt
11	FF0B	65291		Raster-Vergleich (Bit 0-7)
12	FF0C	65292	0-1	Bit 8-9 der Cursor-Pos. (Reg. 13)
			2-7	Nicht benutzt
13	FF0D	65293		Hardware-Cursor-Position (Bit 0-7)
14	FF0E	65294		Frequenz Stimme 1 (Bit 0-7)
15	FF0F	65295		" " 2 (Bit 0-7)
16	FF10	65296	0-1	" " 2 (Bit 8-9)
			2-7	Nicht benutzt
17	FF11	65297	0-3	Lautstärke (0=Aus, 15=Laut)
			4	Stimme 1 einschalten (1=Ein)
			5	" 2 Rechteck einschalt. (1=Ein)
			6	" " Rauschen " (1=Ein)
			7	Sound Reload Bit
18	FF12	65298	0-1	Bit 8-9 Frequenz Stimme 1 (Reg. 14)
			2	RAM/ROM Bank (0=RAM, 1=ROM)
			3-5	Adresse des Bitmap-RAM (Bit 13-15)
			6-7	Nicht benutzt
19	FF13	65299	0	ROM-Bank Status-Bit (nur Read)
			1	Force Single Clock Bit (1=Verbietet doppelte Taktfrequenz)
			2-7	Adresse d. Zeichensatzes (Bit 10-15)

20	FF14	65300	0-2	Nicht benutzt
			3-7	Adresse des Video-RAM (Bit 11-15)
21	FF15	65301	0-3	Hintergrund 0 Farbe
			4-6	" " Luminanz
			7	Nicht benutzt
22	FF16	65302	0-3	Hintergrund 1 Farbe
			4-6	" " Luminanz
			7	Nicht benutzt
23	FF17	65303	0-3	Hintergrund 2 Farbe
			4-6	" " Luminanz
			7	Nicht benutzt
24	FF18	65304	0-3	Hintergrund 3 Farbe
			4-6	" " Luminanz
			7	Nicht benutzt
25	FF19	65305	0-3	Rahmen-Farbe
			4-6	" Luminanz
			7	Nicht benutzt
26	FF1A	65306	0-1	Bit 8-9 von Bit Map Reload (Reg. 27)
			2-7	Nicht benutzt
27	FF1B	65307		Bit Map Reload der Character-Position (Bit 0-7)
28	FF1C	65308	0	Bit 8 der Raster-Zeile (Reg. 29)
			1-7	Nicht benutzt
29	FF1D	65309		Aktuelle Raster-Zeile (Bit 0-7)
30	FF1E	65310		" " Spalte (Bit 1-8)
31	FF1F	65311	0-2	Vertikale Subadresse
			3-6	Blink-Rate
			7	Nicht benutzt

62	FF3E	65342	ROM-Select (Nur Write)
63	FF3F	65343	RAM-Select (Nur Write)

KERNAL-SPRUNGTABELLE ($FF49-FFFF)

ADRESSE (hex)	ADRESSE (dez)	CODE	NAME	BESCHREIBUNG
$FF49	65353	JMP $B7C2		Funktionstaste definieren (Nr. nach $76, Adresse nach $22-23, Länge in Akkumul.)
$FF4C	65356	JMP $DC49		Print
$FF4F	65359	JMP $FBD8		Meldung ausgeben
$FF52	65362	JMP $F445		Maschinen-Monitor aufrufen
$FF55-FF7E	65365-65406	$FF,$FF,... ...,$FF,$FF		Nicht benutzt
$FF7F	65407	$2A		Nicht benutzt
$FF80	65408	$84		" "
$FF81	65409	JMP $D84E	CINT	Editor initialisieren
$FF84	65412	JMP $F30B	IOINIT	I/O "
$FF87	65415	JMP $F352	RAMTAS	RAM " , Kassettenpuffer einrichten, Bildschirm nach $0C00
$FF8A	65418	JMP $F2CE	RESTOR	Vektoren wiederherstellen
$FF8D	65421	JMP $F2D3	VECTOR	Abspeichern von RAM
$FF90	65424	JMP $F41A	SETMSG	KERNAL-Meldungen steuern
$FF93	65427	JMP $EE4D	SECOND	Sekundäradresse nach LISTEN übertragen
$FF96	65430	JMP $EE1A	TKSA	Sekundäradresse nach TALK übertragen
$FF99	65433	JMP $F427	MEMTOP	Oberen RAM-Pointer lesen/setzen
$FF9C	65436	JMP $F436	MEMBOT	Unteren RAM-Pointer lesen/setzen
$FF9F	65439	JMP $DB11	SCNKEY	Tastatur abfragen
$FFA2	65442	JMP $F423	SETTMO	Zeitsperre für (optionalen) IEC-Bus setzen
$FFA5	65445	JMP $EC8B	ACPTR	Byte-Eingabe zum seriellen Port
$FFA8	65448	JMP $ECDF	CIOUT	Byte-Ausgabe über den seriellen Bus
$FFAB	65451	JMP $EF3B	UNTLK	UNTALK-Befehl für seriellen Bus

$FFAE	65454	JMP $EF23	UNLSN	UNLISTEN-Befehl für seriellen Bus	
$FFB1	65457	JMP $EE2C	LISTEN	LISTEN-Befehl für Geräte am seriellen Bus	
$FFB4	65460	JMP $EDFA	TALK	TALK-Befehl für Geräte am seriellen Bus	
$FFB7	65463	JMP $F41C	READST	I/O-Statuswort lesen	
$FFBA	65466	JMP $F413	SETLFS	Logische, Primär- und Sekundär-Adresse setzen	
$FFBD	65469	JMP $F40C	SETNAM	Dateinamen festlegen	
$FFC0	65472	JMP ($0318)	OPEN	Öffnen einer logischen Datei	$EF53
$FFC3	65475	JMP ($031A)	CLOSE	Schließen einer logischen Datei	$EE5D
$FFC6	65478	JMP ($031C)	CHKIN	Kanal für Eingabe öffnen	$ED18
$FFC9	65481	JMP ($031E)	CHKOUT	Kanal für Augabe öffnen	$ED60
$FFCC	65484	JMP ($0320)	CLRCHN	Schließen der I/O-Kanäle	$EF0C
$FFCF	65487	JMP ($0322)	CHRIN	Zeicheneingabe	$EBE8
$FFD2	65490	JMP ($0324)	CHROUT	Zeichenausgabe	$EC4B
$FFD5	65493	JMP $F043	LOAD	Von Peripherie laden	
$FFD8	65496	JMP $F194	SAVE	Nach " speichern	
$FFDB	65499	JMP $CF2D	SETTIM	Uhrzeit setzen	
$FFDE	65502	JMP $CF26	RDTIM	" lesen	
$FFE1	65505	JMP ($0326)	STOP	STOP-Taste abfragen	$F265
$FFE4	65508	JMP ($0328)	GETIN	Zeichen aus Tastaturpuffer lesen	$EBD9
$FFE7	65511	JMP ($032A)	CLALL	Schließen aller Kanäle und Dateien	$EF08
$FFEA	65514	JMP $CEF0	UDTIM	Uhrzeit inkrementieren	
$FFED	65517	JMP $D834	SCREEN	X-, Y-Bildschirmaufbau ermitteln	
$FFF0	65520	JMP $D839	PLOT	X-, Y-Cursorposition lesen/setzen	
$FFF3	65523	JMP $FC19	IOBASE	Basisadress-Rückmeldung der I/O-Geräte	
$FFF6	65526	STA $FF3E		ROM einschalten	
$FFF9	65529	JMP $F2A4		Sprung zur Reset-Routine	

$FFFC	65532	$FFF6	Prozessor-Reset
$FFFE	65534	$FCB3	" Interrupt

4.6 VERGLEICHSTABELLE C-64 UND PLUS/4

In vielen Zeitungen werden Programme für den C-64 abgedruckt, die z.T. auch für PLUS/4-Besitzer interessant sein dürften. Während sich die Programme teilweise damit beschäftigen, dem C-64 BASIC-Befehle beizubringen, die der PLUS/4 sowieso schon eingebaut hat (z.B. Grafik-Befehle) und somit für Sie uninteressant sind, gibt es doch sicher einige Programme, die Sie gerne auch auf Ihrem PLUS/4 haben möchten. Um Ihnen das Umschreiben zu erleichtern, haben wir eine Vergleichstabelle aller identischen Adressen zusammengestellt. Da viele C-64-Programme von PEEK- und POKE-Befehlen wimmeln, können Sie hier die entsprechenden PLUS/4-Adressen heraussuchen. Alles kann man natürlich nicht umschreiben (z.B. Programme mit Sprites), aber was vergleichbar ist, haben wir in der nachfolgenden Tabelle erfasst. Die Erklärung der Bedeutung der Adresse ist sehr kurz; eine ausführlichere Beschreibung finden Sie im vorherigen Kapitel.

ADRESSE C64 hex	ADRESSE C64 dez	ADRESSE +4 hex	ADRESSE +4 dez	LABEL	BEDEUTUNG
$0000	0	$0000	0	D6510	Datenrichtungsregister
$0001	1	$0001	1	R6510	I/O-Port
$0003-	3-	$02F2-	754-	ADRAY1	Zeiger auf Konv. GK>IN
$0005-	5-	$02F4-	756-	ADRAY2	" " " IN>GK
$0007	7	$0007	7	CHARAC	Suchzeichen
$0008	8	$0008	8	ENDCHR	F: Suchen nach Anführ.
$0009	9	$0009	9	TRMPOS	Bildsch.sp. ab 1. TAB
$000A	10	$000A	10	VERCK	F: 0=LOAD, 1=VERIFY
$000B	11	$000B	11	COUNT	Eingabepufferzeiger
$000C	12	$000C	12	DIMFLG	F: Standard-Felddimen.
$000D	13	$000D	13	VALTYP	Datentyp: $FF=String
$000E	14	$000E	14	INTFLG	Datentyp: $80=Integer
$0010	16	$0010	16	SUBFLG	F: Benutzerfunktionsa.
$0011	17	$0011	17	INPFLG	F: $00=INPUT, $40=GET,
$0012	18	$0012	18	TANSGN	F: Vorzeichen TAN
$0013	19	$0013	19	CHANNL	F: INPUT-Kommentar
$0014-	20-	$0014-	20-	LINNUM	Ganzzahliger Wert
$0016	22	$0016	22	TEMPPT	Z: Temp. Stringstapel
$0017-	23-	$0017-	23-	LASTPT	Letzte Stringadresse

$0019-	25-	$0019-	25-	TEMPST	Stapel für temp. Stri.
$0022-	34-	$0022-	34-	INDEX	Bereich f. Hilfszeiger
$0026-	38-	$0026-	38-	RESHO	Bereich für Multiplik.
$002B-	43-	$002B-	43-	TXTTAB	Z: Anfang BASIC-Text
$002D-	45-	$002D-	45-	VARTAB	Z: Anfang BASIC-Varia.
$002F-	47-	$002F-	47-	ARYTAB	Z: Anfang BASIC-Felder
$0031-	49-	$0031-	49-	STREND	Z: Ende BASIC-Fe. (+1)
$0033-	51-	$0033-	51-	FRETOP	Z: Anfang der Strings.
$0035-	53-	$0035-	53-	FRESPC	Hilfszeiger f. Strings
$0037-	55-	$0037-	55-	MEMSIZ	Z: Oberste BASIC-Adre.
$0039-	57-	$0039-	57-	CURLIN	Derzeitige BASIC-Zeile
$003B-	59-	$0259-	601-	OLDLIN	Vorherige BASIC-Zeile
$003D-	61-	$025B-	602-	OLDTXT	Z: BASIC-Anw. für CONT
$003F-	63-	$003F-	63-	DATLIN	Derzeitige DATA-Zeile
$0041-	65-	$0041-	65-	DATPTR	Z: Derz. DATA-Adresse
$0043-	67-	$0043-	67-	INPPTR	V: INPUT-Routine
$0045-	69-	$0045-	69-	VARNAM	Derz. BASIC-Varia.name
$0047-	71-	$0047-	71-	VARPNT	Adresse der akt. Vari.
$0049-	73-	$0049-	73-	FORPNT	Variabl.z. f. FOR/NEXT
$004B-	75-	$004B-	75-	OPPTR	Zwischenz. für BASIC
$0061	97	$0061	97	FACEXP	Gleitpunktakku 1 Expo.
$0062-	98-	$0062-	98-	FACHO	" 1 Mant.
$0066	102	$0066	102	FACSGN	" 1 Vorz.
$0067	103	$0067	103	SGNFLG	Z: Polynomauswertung
$0068	104	$0068	104	BITS	Gleitpunktakku 1 Über.
$0069	105	$0069	105	ARGEXP	" 2 Expo.
$006A-	106-	$006A-	106-	ARGHO	" 2 Mant.
$006E	110	$006E	110	ARGSGN	" 2 Vorz.
$006F	111	$006F	111	ARISGN	Vorz.vergleich Akku1/2
$0070	112	$0070	112	FACOV	Gleitpunktakku 1 Nied.
$0071-	113-	$0071-	113-	FBUFPT	Z: Kassettenbuffer
$0073-	115-	$0473-	1139-	CHRGET	Unterrout. n. Byte le.
$0079	121	$0479	1145	CHRGOT	Ern. Lesen des gl. Te.
$007A-	122-	$003B-	59-	TXTPTR	Z: Derz. Byte des Te.
$008B-	139-	$0503-	1283-	RNDX	Startwert für RND
$0090	144	$0090	144	STATUS	Kernal-I/O-Statuswort
$0091	145	$0091	145	STKEY	F: STOP-/RVS-Taste
$0093	147	$0093	147	VERFCK	F: 0=LOAD, 1=VERIFY
$0094	148	$0094	148	C3PO	F: Ser. Bus, Zeichen
$0095	149	$0095	149	BSOUR	Zeichen im Puffer
$0098	152	$0097	151	LDTND	Anzahl offene Dateien

$0099	153	$0098	152	DFLTN	Eingabegerät (Std.: 3)
$009A	154	$0099	153	DFLTO	Ausgabegerät (Std.: 0)
$009D	157	$009A	154	MSGFLG	F: $80=Direkt, $00=Pr.
$00A0-	160-	$00A3-	163-	TIME	Echtzeituhr (ca. 1/60)
$00AC-	172-	$009B-	155-	SAL	Z: Kass.b./Bild scrol.
$00AE-	174-	$009D-	157-	EAL	Z: Kass.ende/Pgm.ende
$00B7	183	$00AB	171	FNLEN	Länge des akt. Datein.
$00B8	184	$00AC	172	LA	Logische Dateinummer
$00B9	185	$00AD	173	SA	Aktuelle Sekundäradr.
$00BA	186	$00AE	174	FA	Aktuelle Gerätenummer
$00BB-	187-	$00AF-	175-	FNADR	Z: Aktueller Dateiname
$00C1-	193-	$00B2-	178-	STAL	I/O-Startadresse
$00C3-	195-	$00B4-	180-	MEMUSS	Basis-Ladeadresse
$00C5	197	$07F6	2038	LSTX	Derz. gedrückte Taste
$00C6	198	$00EF	239	NDX	Anz. Zei. in Ta.puffer
$00C7	199	$00C2	194	RVS	F: RVS-Zeichen (1=Ja)
$00C8	200	$00C3	195	INDX	Z: Ende der log. Zeile
$00C9-	201-	$00C4-	196-	LSXP	Cursor X/Y für Eingabe
$00CB	203	$00C6	198	SFDX	F: Gedrückte Taste
$00D0	208	$00C7	199	CRSW	F: Eingabe INPUT o GET
$00D1-	209-	$00C8-	200-	PNT	Z: Derz. Bildschirmze.
$00D3	211	$00CA	202	PNTR	Cursorsp. in derz. Ze.
$00D4	212	$00CB	203	QTSW	F: Anführungsze.-Modus
$00D6	214	$00CD	205	TBLX	Cursorzeile
$00D8	216	$00CF	207	INSRT	F: INST-Modus
$00F3-	243-	$00EA-	234-	USER	Z: Derz. Farb-RAM
$00F5-	245-	$00EC-	236-	KEYTAB	V: Tastatur-Dekodier.
$0100-	256-	$0100-	256-		Prozessorstack
$0200-	512-	$0200-	512-	BUF	System-Eingabepuffer
$0259-	601-	$0509-	1289-	LAT	Tab. der log. Filennr.
$0263-	611-	$0513-	1299-	FAT	" " Gerätenummern
$026D-	621-	$051D-	1309-	SAT	" " Sekundäradre.
$0277-	631-	$0527-	1319-	KEYD	Tastaturpuffer (FIFO)
$0281-	641-	$0531-	1329-	MEMSTR	Z: Startadresse RAM
$0283-	643-	$0533-	1331-	MSIZ	Z: Endadresse RAM
$0285	645	$0535	1333	TIMOUT	F: Zeitüberschrei. IEC
$0286	646	$053B	1339	COLOR	Zeichenfarbe
$0288	648	$053E	1342	HIBASE	Video-RAM Anfang, Page
$0289	649	$053F	1343	XMAX	Größe des Tast.puffers
$028A	650	$0540	1344	RPTFLG	F: Tastenwiederholung
$028B	651	$0541	1345	KOUNT	Zählgeschw. für Wiede.

$028C	652	$0542	1346	DELAY	Zähler für Wied.verzö.
$028D	653	$0543	1347	SHFLAG	F: Tasten SHIFT, CTRL,
$028E	654	$0544	1348	LSTSHF	Letztes SHIFT-Muster
$028F-	655-	$0545-	1349-	KEYLOG	Zeiger auf Tast.Decod.
$0291	657	$0547	1351	MODE	F: $80=SHIFT unwirksam
$0292	658	$0548	1352	AUTODN	F: Automat. Scrollen
$0300-	768-	$0300-	768-	IERROR	V: BASIC-Fehlermeldung
$0302-	770-	$0302-	770-	IMAIN	V: " -Warmstart
$0304-	772-	$0304-	772-	ICRNCH	V: " -Token generi.
$0306-	774-	$0306-	774-	IQPLOP	V: " -Text listen
$0308-	776-	$0308-	776-	IGONE	V: " -Befehl ausfü.
$030A-	778-	$030A-	778-	IEVAL	V: " -Tokenauswert.
$030C	780	$07F2	2034	SAREG	Akku bei SYS-Befehl
$030D	781	$07F3	2035	SXREG	X-Regi. bei SYS-Befehl
$030E	782	$07F4	2036	SYREG	Y- " " " "
$030F	783	$07F5	2037	SPREG	SP- " " " "
$0310	784	$0500	1280	USRPOK	USR-Sprung
$0311-	785-	$0501-	1281-	USRADD	USR-Adresse (low/high)
$0314-	788-	$0314-	788-	CINV	V: Hardware-Interrupt
$0316-	790-	$0316-	790-	CBINV	V: BRK-Interrupt
$031A-	794-	$0318-	792-	IOPEN	V: Kernal-OPEN
$031C-	796-	$031A-	794-	ICLOSE	V: " CLOSE
$031E-	798-	$031C-	796-	ICHKIN	V: " CHKIN
$0320-	800-	$031E-	798-	ICKOUT	V: " CHKOUT
$0322-	802-	$0320-	800-	ICLRCH	V: " CLRCHN
$0324-	804-	$0322-	802-	IBASIN	V: " CHRIN
$0326-	806-	$0324-	804-	IBASOUT	V: " CHROUT
$0328-	808-	$0326-	806-	ISTOP	V: " STOP
$032A-	810-	$0328-	808-	IGETIN	V: " GETIN
$032C-	812-	$032A-	810-	ICLALL	V: " CLALL
$032E-	814-	$032C-	812-	USRCMD	Benutzer-IRQ (Monitor)
$0330-	816-	$032E-	814-	ILOAD	V: Kernal-LOAD
$0332-	818-	$0330-	816-	ISAVE	V: " SAVE
$033C-	828-	$0332-	818	TAPBUF	Kassettenbuffer
$0400-	1024-	$0C00-	3076	VICSCN	Video-RAM
$0800-	2048-	$1000-	4096-	BASBGN	BASIC-RAM
$D011	53265	$FF06	65286		Bit 7 ist verschieden!
$D012	53266	$FF0B	65291		Raster-Interrupt
$D016	53270	$FF07	65287		Bits 5-7 verschieden!
$D020	53280	$FF19	65305		Rahmenfarbe
$D021	53281	$FF15	65301		Hintergrund 0 Farbe

$D022	53282	$FF16	65302		"	1	"
$D023	53283	$FF17	65303		"	2	"
$D024	53284	$FF18	65304		"	3	"
$D800	55296	$0800	2048	COLSCN	Farb-RAM		

MEMORY MAP DES C-64

ADRESSE (hex)	ADRESSE (dez)	BESCHREIBUNG
$0000-00FF	0- 255	Zero-Page
$0100-01FF	256- 511	Prozessor-Stack
$0200-03FF	512- 1023	Verschiedene Variablen, Kassettenbuffer
$0400-07E7	1024- 2023	Video-RAM
$0800-9FFF	2048-40959	BASIC-RAM
$A000-BFFF	40960-49151	BASIC-Interpreter
$C000-CFFF	49152-53247	Freies RAM (für Unterprogramme usw.)
$D000-D3FF	53248-54271	Video-Controller (Register bis $D02E)
$D400-D7FF	54272-55295	Sound-Controller (Register bis $D41C)
$D800-DBE7	55296-56295	Farb-RAM
$DC00-DCFF	56320-56575	CIA #1 : Tastatur, Joystick, Timer
$DD00-DDFF	56576-56831	CIA #2 : Serieller Bus, RS-232, Timer
$DE00-DFFF	56832-57343	Nicht belegt; frei für I/O-Erweiterungen
$E000-FFFF	57344-65535	Kernal-ROM

5. UTILITIES

5.1 ZUFALLSZAHLENGENERATOR IN MASCHINENSPRACHE

Das folgende Beispiel soll demonstrieren, wie man in Maschinensprache Zufallswerte erzeugen kann. Dies ist von besonderem Interesse u.a. für Spiele-Programmierer, die häufig Zufallswerte brauchen. In unserem Beispiel wird ein Wurf mit zwei Würfeln simuliert. Wenn Sie die Leertaste drücken, wird gewürfelt und am oberen Bildschirmrand erscheint das Ergebnis unseres Wurfes. Dabei wird als zufälliger Startwert der Rasterstrahl benutzt.

```
>1000 00 0B 10 00 00 9E 34 31 :......41
>1008 31 32 00 00 00 00 00 00 :12......
>1010 A9 93 20 D2 FF 20 42 10 :). B.
>1018 A9 0F 8D 11 08 8D 13 08 :).......
>1020 20 E4 FF C9 20 D0 F9 20 : I Py
>1028 35 10 8D 11 0C 20 35 10 :5.... 5.
>1030 8D 13 0C D0 EB 20 4C 10 :...Pk L.
>1038 F0 FB C9 07 B0 F7 18 69 :p(I.0w.i
>1040 30 60 AD 0B FF A2 04 95 :0`-"..
>1048 D0 CA 10 FB A9 E0 25 D4 :PJ.()`%T
>1050 09 20 85 D4 18 A2 04 B5 :. .T.".5
>1058 D0 75 D0 95 D5 CA 10 F7 :PuP.UJ.w
>1060 18 A2 04 B5 D0 75 D5 95 :.".5PuU.
>1068 D0 CA 10 F7 18 A2 02 B5 :PJ.w.".5
>1070 D0 75 D7 95 D0 CA 10 F7 :PuW.PJ.w
>1078 A5 D0 60 00 00 00 00 00 :%P`.....
```

```
S"RANDOM",01,1001,107B
```

```
. 1010  A9 93    LDA #$93
. 1012  20 D2 FF JSR $FFD2
. 1015  20 42 10 JSR $1042
. 1018  A9 0F    LDA #$0F
. 101A  8D 11 08 STA $0811
. 101D  8D 13 08 STA $0813
. 1020  20 E4 FF JSR $FFE4
. 1023  C9 20    CMP #$20
. 1025  D0 F9    BNE $1020
```

```
. 1027  20 35 10 JSR $1035
. 102A  8D 11 0C STA $0C11
. 102D  20 35 10 JSR $1035
. 1030  8D 13 0C STA $0C13
. 1033  D0 EB    BNE $1020
. 1035  20 4C 10 JSR $104C
. 1038  F0 FB    BEQ $1035
. 103A  C9 07    CMP #$07
. 103C  B0 F7    BCS $1035
. 103E  18       CLC
. 103F  69 30    ADC #$30
. 1041  60       RTS
. 1042  AD 0B FF LDA $FF0B
. 1045  A2 04    LDX #$04
. 1047  95 D0    STA $D0,X
. 1049  CA       DEX
. 104A  10 FB    BPL $1047
. 104C  A9 E0    LDA #$E0
. 104E  25 D4    AND $D4
. 1050  09 20    ORA #$20
. 1052  85 D4    STA $D4
. 1054  18       CLC
. 1055  A2 04    LDX #$04
. 1057  B5 D0    LDA $D0,X
. 1059  75 D0    ADC $D0,X
. 105B  95 D5    STA $D5,X
. 105D  CA       DEX
. 105E  10 F7    BPL $1057
. 1060  18       CLC
. 1061  A2 04    LDX #$04
. 1063  B5 D0    LDA $D0,X
. 1065  75 D5    ADC $D5,X
. 1067  95 D0    STA $D0,X
. 1069  CA       DEX
. 106A  10 F7    BPL $1063
. 106C  18       CLC
. 106D  A2 02    LDX #$02
. 106F  B5 D0    LDA $D0,X
. 1071  75 D7    ADC $D7,X
. 1073  95 D0    STA $D0,X
```

```
. 1075  CA       DEX
. 1076  10 F7    BPL $106F
. 1078  A5 D0    LDA $D0
. 107A  60       RTS
```

5.2 JOYSTICK-ABFRAGE IN MASCHINENSPRACHE

Mit der nachfolgenden Routine können Sie einen Joystick in Maschinensprache abfragen. Wir haben dieses Programm schon im Grafik-Kapitel mehrmals als Unterprogramm benutzt, es läßt sich aber auch in Ihr eigenes Programm einbauen.

```
>1000 00 0C 10 00 00 9E 34 31 :.......41
>1008 31 32 00 00 00 00 00 00 :12......
>1010 A9 FD 8D 08 FF AD 08 FF :)}.-
>1018 A0 00 A2 00 4A B0 01 88 : .".J0..
>1020 4A B0 01 C8 4A B0 01 CA :J0.HJ0.J
>1028 4A B0 01 E8 86 D0 84 D1 :J0.h.P.Q
>1030 29 08 60 AA AA AA AA AA :).`*****
```

```
S"JOYSTICK",01,1001,1033
```

```
. 1010  A9 FD     LDA #$FD
. 1012  8D 08 FF  STA $FF08
. 1015  AD 08 FF  LDA $FF08
. 1018  A0 00     LDY #$00
. 101A  A2 00     LDX #$00
. 101C  4A        LSR
. 101D  B0 01     BCS $1020
. 101F  88        DEY
. 1020  4A        LSR
. 1021  B0 01     BCS $1024
. 1023  C8        INY
. 1024  4A        LSR
. 1025  B0 01     BCS $1028
. 1027  CA        DEX
. 1028  4A        LSR
. 1029  B0 01     BCS $102C
. 102B  E8        INX
. 102C  86 D0     STX $D0
. 102E  84 D1     STY $D1
. 1030  29 08     AND #$08
. 1032  60        RTS
```

5.3 TURBO-MODUS FÜR DEN PLUS/4

Für alle, denen der PLUS/4 zu langsam ist, können wir an dieser Stelle einen Trick verraten, mit dem er rund 30% schneller wird. Da wir in einem Turbo-Zeitalter leben, nennen wir es halt den Turbo-Modus. Der Trick besteht darin, daß der Bildschirm abgeschaltet wird. Das ist natürlich unpraktikabel, wenn Sie Grafiken erstellen und diese gleichzeitig sehen wollen. Wenn Sie aber rechenintensive Aufgaben haben, bei denen Sie nicht ständig auf den Bildschirm schauen müssen. Der Geschwindigkeitsvorteil ergibt sich dadurch, daß bei abgeschaltetem Bildschirm der Videochip TED den Mikroprozessor nicht mehr verlangsamt. Ein kleines Demo-Programm, das erst im Turbo-Modus und dann im normalen Modus läuft, zeigt Ihnen, wie´s geht und wieviel Zeit Sie sparen können.

```
100 POKE 65286,PEEK(65286) AND 239
110 GOSUB 150
120 POKE 65286,PEEK(65286) OR 16
130 GOSUB 150
140 END
150 T=TI
160 FOR I=1 TO 1000
170 : B=B+1
180 NEXT I
190 PRINT USING "##.##";(TI-T)/60
200 RETURN
```

5.4 OLD (WIEDERHERSTELLEN EINES PROGRAMMS NACH NEW)

Kennen Sie das: Sie haben ein Programm im Computer und geben NEW ein oder drücken den Reset-Knopf, obwohl Sie das Programm noch brauchen und abspeichern wollen? Wenn ja, dann können Sie unser OLD-Programm gut gebrauchen. Nach NEW oder einem Reset brauchen Sie einfach nur: SYS 1630 eingeben und Ihr Programm ist wieder da. Das Maschinenprogramm liegt in einem Bereich, der bei einem Reset nicht gelöscht wird. Am besten, Sie laden sich das OLD-Programm kurz ein, bevor Sie anfangen, zu programmieren. Es steht Ihnen dann jederzeit zur Verfügung.

Zur Vorgehensweise: tippen Sie das nachfolgende BASIC-Programm ein und speichern Sie es ab, bevor Sie es starten. Dann können Sie es starten, wobei eine Prüfsumme auf eine fehlerhafte Eingabe hinweist. Wenn alles ok ist, löschen Sie probeweise das BASIC-Programm mit: NEW und geben Sie: SYS 1630 ein. Es sollte jetzt wieder da sein (mit LIST ausprobieren). Das einzige, was evtl. nicht stimmt, ist die erste Zeilennummer. Nachdem Sie sich vom ordnungsgemäßen Zustand der OLD-Routine überzeugt haben (es geht ohne den Notar vom Lotto), gehen Sie in den Monitor und speichern das Programm ab. Wenn Sie es nochmal brauchen, können Sie es direkt so wieder einladen (der BASIC-Vorspann wird nicht mehr benötigt).

```
100 FOR I=1630 TO 1630+106
110 : READ A
120 : POKE I,A
130 : C=C+A
140 NEXT I
150 IF C<>10560 THEN PRINT "PRUEFSUMMENFEHLER" : STOP
160 DATA 165,43,133,3,165,44,133,4,160,4,177,3,240,3,200,208
170 DATA 249,200,152,32,189,6,160,0,165,3,145,43,200,165,4,145
180 DATA 43,200,169,0,145,43,200,145,43,165,43,133,3,165,44,133
190 DATA 4,160,0,169,4,32,189,6,177,3,240,7,32,187,6,169
200 DATA 0,240,245,32,187,6,177,3,240,4,169,0,240,234,32,187
210 DATA 6,32,187,6,165,3,133,45,165,4,133,46,96,169,1,24
220 DATA 101,3,133,3,169,0,101,4,133,4,96

S"OLD",01,065E,06C9
```

```
. 065E  A5 2B     LDA $2B
. 0660  85 03     STA $03
. 0662  A5 2C     LDA $2C
. 0664  85 04     STA $04
. 0666  A0 04     LDY #$04
. 0668  B1 03     LDA ($03),Y
. 066A  F0 03     BEQ $066F
. 066C  C8        INY
. 066D  D0 F9     BNE $0668
. 066F  C8        INY
. 0670  98        TYA
. 0671  20 BD 06  JSR $06BD
. 0674  A0 00     LDY #$00
. 0676  A5 03     LDA $03
. 0678  91 2B     STA ($2B),Y
. 067A  C8        INY
. 067B  A5 04     LDA $04
. 067D  91 2B     STA ($2B),Y
. 067F  C8        INY
. 0680  A9 00     LDA #$00
. 0682  91 2B     STA ($2B),Y
. 0684  C8        INY
. 0685  91 2B     STA ($2B),Y
. 0687  A5 2B     LDA $2B
. 0689  85 03     STA $03
. 068B  A5 2C     LDA $2C
. 068D  85 04     STA $04
. 068F  A0 00     LDY #$00
. 0691  A9 04     LDA #$04
. 0693  20 BD 06  JSR $06BD
. 0696  B1 03     LDA ($03),Y
. 0698  F0 07     BEQ $06A1
. 069A  20 BB 06  JSR $06BB
. 069D  A9 00     LDA #$00
. 069F  F0 F5     BEQ $0696
. 06A1  20 BB 06  JSR $06BB
. 06A4  B1 03     LDA ($03),Y
. 06A6  F0 04     BEQ $06AC
. 06A8  A9 00     LDA #$00
. 06AA  F0 EA     BEQ $0696
```

```
. 06AC  20 BB 06 JSR $06BB
. 06AF  20 BB 06 JSR $06BB
. 06B2  A5 03    LDA $03
. 06B4  85 2D    STA $2D
. 06B6  A5 04    LDA $04
. 06B8  85 2E    STA $2E
. 06BA  60       RTS
. 06BB  A9 01    LDA #$01
. 06BD  18       CLC
. 06BE  65 03    ADC $03
. 06C0  85 03    STA $03
. 06C2  A9 00    LDA #$00
. 06C4  65 04    ADC $04
. 06C6  85 04    STA $04
. 06C8  60       RTS
```

5.5 MERGE (ANEINANDERHÄNGEN VON PROGRAMMEN)

Obwohl das Basic des PLUS/4 sehr umfangreich ist und kaum Wünsche offen läßt, fehlt leider ein Befehl, der sehr praktisch ist: MERGE. Dieser erlaubt es, zwei BASIC-Programme aneinanderzuhängen. Wenn Sie sich z.B. eine kleine Bilbliothek von Unterprogrammen aufgebaut haben, so können Sie diese mit MERGE zu Ihrem aktuellen Programm stets dazuladen.

Zur Vorgehensweise: nachfolgendes BASIC-Programm eingeben und abspeichern. Wenn nach dem Starten kein Prüfsummenfehler gemeldet wird, gehen Sie in den Monitor und speichern es ab. Es liegt übrigens genau wie OLD in einem Bereich, der bei einem Reset nicht gelöscht wird. Wenn Sie zwei Programme aneinanderhängen wollen, gehen Sie jetzt so vor:

1. Wenn sich MERGE noch nicht im Computer befindet, einladen.

2. Laden Sie Ihr erstes Programm von Kassette oder Diskette.

3. Geben Sie: **SYS 1737** ein.

4. Laden Sie Ihr zweites Programm ein.

5. Geben Sie: **SYS 1759** ein.

Ihre beiden Programme sind jetzt zusammengebunden. Eventuell müssen Sie das neue Programm jetzt RENUMBER`n. Vermeiden Sie es, gleiche Zeilennummern zu verwenden. Am besten, Sie geben Ihren Unterprogrammen hohe Zeilennummern (z.B. 62000, 63000).

```
100 FOR I=1737 TO 1737+30
110 : READ A
120 : POKE I,A
130 : C=C+A
140 NEXT I
150 IF C<>3786 THEN PRINT "PRUEFSUMMENFEHLER" : STOP
160 DATA 165,43,133,232,165,44,133,233,56,165,45,233,2,133,43,165
170 DATA 46,233,0,133,44,96,165,232,133,43,165,233,133,44,96
```

```
S"MERGE",01,06C9,06E8

. 06C9  A5 2B     LDA $2B
. 06CB  85 E8     STA $E8
. 06CD  A5 2C     LDA $2C
. 06CF  85 E9     STA $E9
. 06D1  38        SEC
. 06D2  A5 2D     LDA $2D
. 06D4  E9 02     SBC #$02
. 06D6  85 2B     STA $2B
. 06D8  A5 2E     LDA $2E
. 06DA  E9 00     SBC #$00
. 06DC  85 2C     STA $2C
. 06DE  60        RTS
. 06DF  A5 E8     LDA $E8
. 06E1  85 2B     STA $2B
. 06E3  A5 E9     LDA $E9
. 06E5  85 2C     STA $2C
. 06E7  60        RTS
```

5.6 VARLIST (AUSDRUCK ALLER VERWENDETEN VARIABLEN)

Wenn Sie ein Programm in BASIC erstellt haben und die Übersicht verloren haben, welche Variablen Sie bereits verwendet haben, dann leistet dieses Hilfs-Programm gute Dienste. Es erstellt eine Liste aller bisher verwendeten Variablen und gibt ihren Typ aus (Fließkomma (FLO), Integer (INT) oder String (STR)).

Zur Vorgehensweise: geben Sie das BASIC-Programm ein und speichern Sie es ab. Wenn die Prüfsumme ok ist, können Sie es ausprobieren. Geben Sie: **SYS 16100** ein und es werden die in dem BASIC-Lader verwendeten Variablen I, A und C ausgegeben (alles Fließkommavariable). Beachten Sie bitte, daß nur die Variablen ausgegeben werden, die bisher aufgerufen wurden. Lassen Sie also Ihr Programm erst ein wenig laufen, bevor Sie es unterbrechen und VARLIST aufrufen.

```
100 POKE 55,227 : POKE 56,62 : CLR
110 FOR I=16100 TO 16100+273
120 : READ A
130 : POKE I,A
140 : C=C+A
150 NEXT I
160 IF C<>30643 THEN PRINT "PRUEFSUMMENFEHLER" : STOP
170 DATA 165,45,133,2,165,46,133,3,165,2,197,47,208,6,165,3
180 DATA 197,48,240,40,160,0,177,2,141,227,63,201,128,176,30,200
190 DATA 177,2,201,128,176,65,32,106,63,32,211,63,169,7,24,101
200 DATA 2,133,2,169,0,101,3,133,3,76,236,62,96,56,233,128
210 DATA 141,227,63,200,177,2,201,128,240,24,56,233,128,141,228,63
220 DATA 32,134,63,32,147,63,174,238,63,32,164,63,32,205,63,76
230 DATA 13,63,169,32,76,49,63,240,24,56,233,128,141,228,63,32
240 DATA 134,63,32,147,63,174,240,63,32,164,63,32,205,63,76,13
250 DATA 63,169,32,76,80,63,201,0,240,19,141,228,63,32,134,63
260 DATA 32,147,63,174,239,63,32,164,63,32,205,63,96,169,32,76
270 DATA 110,63,173,227,63,32,183,63,173,228,63,32,183,63,96,162
280 DATA 15,142,241,63,169,32,32,183,63,174,241,63,202,208,242,96
290 DATA 160,3,142,242,63,189,229,63,32,183,63,174,242,63,232,136
300 DATA 208,240,96,141,243,63,140,245,63,142,244,63,32,210,255,174
310 DATA 244,63,172,245,63,173,243,63,96,169,13,32,183,63,96,32
```

```
320 DATA 228,255,201,32,240,1,96,32,228,255,201,32,208,249,96,0
330 DATA 0,73,78,84,70,76,79,83,84,82,0,3,6,0,0,0,0,0

. 3EE4  A5 2D    LDA $2D
. 3EE6  85 02    STA $02
. 3EE8  A5 2E    LDA $2E
. 3EEA  85 03    STA $03
. 3EEC  A5 02    LDA $02
. 3EEE  C5 2F    CMP $2F
. 3EF0  D0 06    BNE $3EF8
. 3EF2  A5 03    LDA $03
. 3EF4  C5 30    CMP $30
. 3EF6  F0 28    BEQ $3F20
. 3EF8  A0 00    LDY #$00
. 3EFA  B1 02    LDA ($02),Y
. 3EFC  8D E3 3F STA $3FE3
. 3EFF  C9 80    CMP #$80
. 3F01  B0 1E    BCS $3F21
. 3F03  C8       INY
. 3F04  B1 02    LDA ($02),Y
. 3F06  C9 80    CMP #$80
. 3F08  B0 41    BCS $3F4B
. 3F0A  20 6A 3F JSR $3F6A
. 3F0D  20 D3 3F JSR $3FD3
. 3F10  A9 07    LDA #$07
. 3F12  18       CLC
. 3F13  65 02    ADC $02
. 3F15  85 02    STA $02
. 3F17  A9 00    LDA #$00
. 3F19  65 03    ADC $03
. 3F1B  85 03    STA $03
. 3F1D  4C EC 3E JMP $3EEC
. 3F20  60       RTS
. 3F21  38       SEC
. 3F22  E9 80    SBC #$80
. 3F24  8D E3 3F STA $3FE3
. 3F27  C8       INY
. 3F28  B1 02    LDA ($02),Y
. 3F2A  C9 80    CMP #$80
. 3F2C  F0 18    BEQ $3F46
. 3F2E  38       SEC
```

```
. 3F2F  E9 80     SBC #$80
. 3F31  8D E4 3F  STA $3FE4
. 3F34  20 86 3F  JSR $3F86
. 3F37  20 93 3F  JSR $3F93
. 3F3A  AE EE 3F  LDX $3FEE
. 3F3D  20 A4 3F  JSR $3FA4
. 3F40  20 CD 3F  JSR $3FCD
. 3F43  4C 0D 3F  JMP $3F0D
. 3F46  A9 20     LDA #$20
. 3F48  4C 31 3F  JMP $3F31
. 3F4B  F0 18     BEQ $3F65
. 3F4D  38        SEC
. 3F4E  E9 80     SBC #$80
. 3F50  8D E4 3F  STA $3FE4
. 3F53  20 86 3F  JSR $3F86
. 3F56  20 93 3F  JSR $3F93
. 3F59  AE F0 3F  LDX $3FF0
. 3F5C  20 A4 3F  JSR $3FA4
. 3F5F  20 CD 3F  JSR $3FCD
. 3F62  4C 0D 3F  JMP $3F0D
. 3F65  A9 20     LDA #$20
. 3F67  4C 50 3F  JMP $3F50
. 3F6A  C9 00     CMP #$00
. 3F6C  F0 13     BEQ $3F81
. 3F6E  8D E4 3F  STA $3FE4
. 3F71  20 86 3F  JSR $3F86
. 3F74  20 93 3F  JSR $3F93
. 3F77  AE EF 3F  LDX $3FEF
. 3F7A  20 A4 3F  JSR $3FA4
. 3F7D  20 CD 3F  JSR $3FCD
. 3F80  60        RTS
. 3F81  A9 20     LDA #$20
. 3F83  4C 6E 3F  JMP $3F6E
. 3F86  AD E3 3F  LDA $3FE3
. 3F89  20 B7 3F  JSR $3FB7
. 3F8C  AD E4 3F  LDA $3FE4
. 3F8F  20 B7 3F  JSR $3FB7
. 3F92  60        RTS
. 3F93  A2 0F     LDX #$0F
. 3F95  8E F1 3F  STX $3FF1
. 3F98  A9 20     LDA #$20
```

```
. 3F9A  20 B7 3F JSR $3FB7
. 3F9D  AE F1 3F LDX $3FF1
. 3FA0  CA       DEX
. 3FA1  D0 F2    BNE $3F95
. 3FA3  60       RTS
. 3FA4  A0 03    LDY #$03
. 3FA6  8E F2 3F STX $3FF2
. 3FA9  BD E5 3F LDA $3FE5,X
. 3FAC  20 B7 3F JSR $3FB7
. 3FAF  AE F2 3F LDX $3FF2
. 3FB2  E8       INX
. 3FB3  88       DEY
. 3FB4  D0 F0    BNE $3FA6
. 3FB6  60       RTS
. 3FB7  8D F3 3F STA $3FF3
. 3FBA  8C F5 3F STY $3FF5
. 3FBD  8E F4 3F STX $3FF4
. 3FC0  20 D2 FF JSR $FFD2
. 3FC3  AE F4 3F LDX $3FF4
. 3FC6  AC F5 3F LDY $3FF5
. 3FC9  AD F3 3F LDA $3FF3
. 3FCC  60       RTS
. 3FCD  A9 0D    LDA #$0D
. 3FCF  20 B7 3F JSR $3FB7
. 3FD2  60       RTS
. 3FD3  20 E4 FF JSR $FFE4
. 3FD6  C9 20    CMP #$20
. 3FD8  F0 01    BEQ $3FDB
. 3FDA  60       RTS
. 3FDB  20 E4 FF JSR $FFE4
. 3FDE  C9 20    CMP #$20
. 3FE0  D0 F9    BNE $3FDB
. 3FE2  60       RTS

>3FE3 00 00 49 4E 54 46 4C 4F :..INTFLO
>3FEB 53 54 52 00 03 06 00 00 :STR.....
>3FF3 00 00 00 00 00 00 00 00 :........
```

5.7 CROSS (ERSTELLT CROSS REFERENZ FÜR BASIC-BEFEHLE)

Dieses Hilfsprogramm erstellt eine Liste aller BASIC-Schlüsselwörter und gibt an, in welchen Zeilen sie verwendet wurden. Dabei können Sie in den Zeilen ab 63460 die Schlüsselwörter angeben, die Sie interessieren und deren Code (die Codes oder Tokens der BASIC-Schlüsselwörter finden Sie in der anschließenden Tabelle). Wenn Sie ein eigenes dazu geben möchten, tippen Sie es ab 63480 zusammen mit dem Token ein und erhöhen den Wert in Zeile 63450.

Zur Vorgehensweise: Geben Sie das Programm ein und speichern Sie es ab. Laden Sie das Sie interessierende Programm ein und MERGEn Sie das CROSS-Programm dazu (siehe 5.5 MERGE). Geben Sie: **RUN 63000** ein und beantworten Sie die Frage, ob die Ausgabe auf Bildschirm oder Drucker erfolgen soll. Haben Sie etwas Geduld, denn das Programm muß für jedes Schlüsselwort das gesamte BASIC-Programm durchkämmen.

```
63000 RESTORE 63450 : READ NU : DIM KW$(NU),K(NU)
63010 FOR I=1 TO NU : READ KW$(I),K(I) : NEXT I
63020 SCNCLR : INPUT "BILDSCHIRM ODER DRUCKER (B/D)";Q$
63030 IF Q$="D" THEN OPEN 4,4 : CMD 4 : GOTO 63060
63040 IF Q$<>"B" THEN 63020
63050 :
63060 FOR I=1 TO NU
63070 : K=K(I)
63080 : PRINT KW$(I) : CN=0 : CO=0
63090 : PT=PEEK(43)+PEEK(44)*256
63100 : LN=0
63110 : DO WHILE LN<>63000
63120 :   NL=PEEK(PT)+PEEK(PT+1)*256
63130 :   LN=PEEK(PT+2)+PEEK(PT+3)*256
63140 :   IF LN=63000 THEN EXIT
63150 :   E=0
63160 :   J=PT+4
63170 :   DO WHILE J<>PT+80
63180 :     CH=PEEK(J)
63190 :     IF CH=34 THEN GOSUB 63320 : IF E THEN EXIT
63200 :     IF CH=K THEN GOSUB 63390
```

```
63210 :      IF CH=0 THEN EXIT
63220 :      J=J+1
63230 :    LOOP
63240 :    PT=NL
63250 : LOOP
63260 : IF CN<>0 THEN PRINT
63270 : PRINT "ANZAHL =";CN : PRINT
63280 NEXT I
63290 IF 0$="D" THEN PRINT#4 : CLOSE 4
63300 END
63310 :
63320 DO WHILE CH<>0
63330 : J=J+1 : CH=PEEK(J)
63340 : IF CH=34 THEN EXIT
63350 : IF CH=0 THEN E=1 : EXIT
63360 LOOP
63370 RETURN
63380 :
63390 IF CO=6 THEN CO=0 : PRINT
63400 CO=CO+1 : LN$=STR$(LN) : LN$=MID$(LN$,2,LEN(LN$))
63410 IF LEN(LN$)<5 THEN LN$=MID$("00000",1,(5-LEN(LN$)))+LN$
63420 PRINT LN$;"    "; : CN=CN+1
63430 RETURN
63440 :
63450 DATA 10
63460 DATA DO,235,EXIT,237,FOR,129,GOSUB,141,GOTO,137,IF,139
63470 DATA LOOP,236,NEXT,130,RETURN,142,WHILE,253
```

6.1 DIE TOKENS DER BASIC-SCHLÜSSELWÖRTER

Wie Sie vielleicht wissen, wird jeder BASIC-Befehl im Programm als sogenanntes "Token" abgespeichert, d.h. es erscheint im Speicher des PLUS/4 als ein Byte und nicht als gesamtes Wort, was natürlich Speicherplatz spart. Es kann bei der Analyse von BASIC-Programmen interessant sein, diese Tokens zu kennen (siehe voriges Kapitel). Hier folgt eine Übersicht aller Schlüsselwörter und deren Tokens:

TOKEN (dez)	TOKEN (hex)	BASIC-SCHLÜSSELWORT
128	$80	END
129	$81	FOR
130	$82	NEXT
131	$83	DATA
132	$84	INPUT#
133	$85	INPUT
134	$86	DIM
135	$87	READ
136	$88	LET
137	$89	GOTO
138	$8A	RUN
139	$8B	IF
140	$8C	RESTORE
141	$8D	GOSUB
142	$8E	RETURN
143	$8F	REM
144	$90	STOP
145	$91	ON
146	$92	WAIT
147	$93	LOAD
148	$94	SAVE
149	$95	VERIFY
150	$96	DEF
151	$97	POKE
152	$98	PRINT#
153	$99	PRINT

154	$9A	CONT
155	$9B	LIST
156	$9C	CLR
157	$9D	CMD
158	$9E	SYS
159	$9F	OPEN
160	$A0	CLOSE
161	$A1	GET
162	$A2	NEW
163	$A3	TAB(
164	$A4	TO
165	$A5	FN
166	$A6	SPC(
167	$A7	THEN
168	$A8	NOT
169	$A9	STEP
170	$AA	+
171	$AB	-
172	$AC	*
173	$AD	/
174	$AE	^
175	$AF	AND
176	$B0	OR
177	$B1	>
178	$B2	=
179	$B3	<
180	$B4	SGN
181	$B5	INT
182	$B6	ABS
183	$B7	USR
184	$B8	FRE
185	$B9	POS
186	$BA	SQR
187	$BB	RND
188	$BC	LOG
189	$BD	EXP
190	$BE	COS
191	$BF	SIN
192	$C0	TAN
193	$C1	ATN
194	$C2	PEEK

195	$C3	LEN
196	$C4	STR$
197	$C5	VAL
198	$C6	ASC
199	$C7	CHR$
200	$C8	LEFT$
201	$C9	RIGHT$
202	$CA	MID$
203	$CB	GO
204	$CC	RGR
205	$CD	RCLR
206	$CE	RLUM
207	$CF	JOY
208	$D0	RDOT
209	$D1	DEC
210	$D2	HEX$
211	$D3	ERR$
212	$D4	INSTR
213	$D5	ELSE
214	$D6	RESUME
215	$D7	TRAP
216	$D8	TRON
217	$D9	TROFF
218	$DA	SOUND
219	$DB	VOL
220	$DC	AUTO
221	$DD	PUDEF
222	$DE	GRAPHIC
223	$DF	PAINT
224	$E0	CHAR
225	$E1	BOX
226	$E2	CIRCLE
227	$E3	GSHAPE
228	$E4	SSHAPE
229	$E5	DRAW
230	$E6	LOCATE
231	$E7	COLOR
232	$E8	SCNCLR
233	$E9	SCALE
234	$EA	HELP
235	$EB	DO

236	$EC	LOOP
237	$ED	EXIT
238	$EE	DIRECTORY
239	$EF	DSAVE
240	$F0	DLOAD
241	$F1	HEADER
242	$F2	SCRATCH
243	$F3	COLLECT
244	$F4	COPY
245	$F5	RENAME
246	$F6	BACKUP
247	$F7	DELETE
248	$F8	RENUMBER
249	$F9	KEY
250	$FA	MONITOR
251	$FB	USING
252	$FC	UNTIL
253	$FD	WHILE
254	$FE	unbenutzt
255	$FF	

ACHTUNG!

Wir haben alles rund um die Computer C-116, C-16 und Plus/4 zu taschengeldfreundlichen Preisen. Fordern Sie unbedingt unseren aktuellen großen Gesamt-Katalog an mit über 50 hervorragenden Spielen (z.B. BONGO CONSTRUCTION SET, BRIDGHEAD, GALAXY, GHOST TOWN, GRANDMASTER, KARATE KING, LEGIONNAIRE, SOMMER OLYMPIADE, SPACE PILOT, TOM, WINTER OLYMPIADE), Anwendungsprogrammen (z.B. GRAFIK DESIGNER, MICRO DATEI, MICRO KALK, MICRO TEXT, MUSIC MASTER, PAINT BOX, TURBO TAPE) und Hardware (SPEICHERERWEITERUNGEN AUF 32K und 64K, JOYSTICKS (anschlußfertig), JOYSTICK-ADAPTER). Postkarte an:

KINGSOFT
Fritz Schäfer
Schnackebusch 4
D-5106 Roetgen