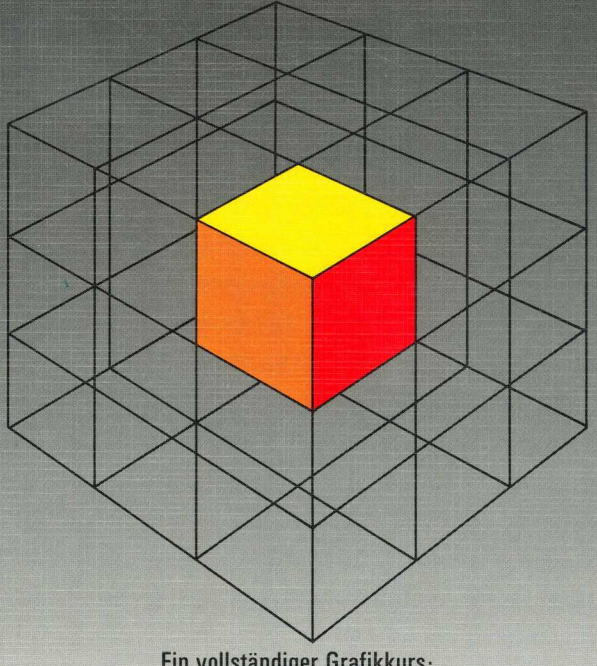
Markt&Technik

Heimo Ponnath

C 64: Wunderland der Grafik



Ein vollständiger Grafikkurs:
● Spriteprogrammierung ● 2D- und 3D-Effekte
● Maschinenspracheroutinen ● Animation.
Mit Sonderteil: 70 Farben auf dem C 64.

Mit allen Beispielen
auf beigelegter Diskette

C 64 – Wunderland der Grafik

Heimo Ponnath

C 64 – Wunderland der Grafik

Ein vollständiger Grafikkurs:
Spriteprogrammierung
2D- und 3D-Effekte
Maschinenspracheroutinen
Animation

Mit Sonderteil:
70 Farben auf dem C 64

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Ponnath, Heimo:
C 64: Wunderland der Grafik : e. vollst. Grafikkurs:
Spriteprogrammierung, 2D- u. 3D-Effekte, Maschinenspracheroutinen,
Animation ; mit Sonderteil: 70 Farben auf dem C 64 / Heimo Ponnath. —
Haar bei München : Markt-und-Technik-Verlag, 1985.
ISBN 3-89090-363-0

15 14 13 12 11 10 9 8
89

ISBN 3-89090-363-0

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Vorwort

Grafik und der C 64 - ein schier unerschöpfliches Thema! Die vergleichsweise schwache Unterstützung der erstaunlich leistungsfähigen C 64-Hardware bringt immer wieder Programmierer und Autoren dazu, interessante und raffinierte Programmzusätze zu erdenken. Dieses Buch ist ein weiterer Beweis dafür, daß der Umgang mit Technik durchaus erstaunlich kreativ sein kann.

Das Buch ist in zwei Teile untergliedert. Im ersten Teil wird zunächst anhand zahlreicher Programmbeispiele gezeigt, was mit dem Grafikchip alles möglich ist. Daran anschließend finden Sie ein zuladbares Assemblerprogramm, das die schnelle und komfortable Erstellung von Grafiken unterstützt und darüber hinaus noch einige zusätzliche nützliche BASIC-Befehle anbietet.

Der zweite Teil enthält eine Programmsammlung, die zeigt, wie man mit dem VIC-Chip bis zu 70 verschiedene Farben erzeugen kann. Einige Anwendungsbeispiele hierzu runden den zweiten Teil ab.

Inhaltsverzeichnis

TEIL I Reise durch die Wunderwelt der Grafik

1	Grafik-Grundlagen	11
1.1	Die Grafikmöglichkeiten des C 64	14
1.2	Die Speicherorganisation des C 64	15
1.3	Der VIC-II-Chip	22
2	Zahlensysteme und Speicherorganisation	31
2.1	Die Begegnung mit den Zweifingerlingen: Das Binärsystem	33
2.2	Die Enttarnung der Sechszehnfingerlinge: Das Hexadezimalsystem	35
2.3	Wir führen den C 64 hinters Licht: Eigene Änderungen an der Speicherorganisation	40
2.4	Und oder? Oder und? Die Befehle AND, OR	45
2.5	Frankensteins freundliche Monster: Eigenen Zeichen	48
3	Die ersten Schritte	55
3.1	Für Farbenkünstler: Der Mehrfarben-Zeichen-Modus	57
3.2	Nachspeise für Farb-Gourmets: Zeichen mit verändertem Hintergrund	61
3.3	Wir betreten Dornröschens Schloß: Das Bit-Mapping-Prinzip	63
3.4	Jetzt kommt Farbe ins Bild	66
3.5	Dornröschen lernt zeichnen: Punkte setzen im Hochauflösungs-Modus	69
4	In Riesenschritten zum Ziel: Die hochauflösende Grafik in BASIC-Programmen	73
4.1	Wir krempeln den Commodore 64 um: Speicherveränderungen für hochauflösende Grafik	75
4.2	Bildschirm-Experimente	79
4.3	Die Grafik bekennt Farbe: Der Bit-Map-Mehrfarben-Modus	83
4.4	Langsam - aber zuverlässig: Eine kleine Grafik-Unterprogramm-Bibliothek in BASIC	85
4.4.1	Erläuterungen zu Programm 2 (Zeilenbereich 49990 bis 51500)	88
4.4.2	Ein Beispiel für die Möglichkeiten der Grafik-Programmbibliothek	92

5	Sprites	97
5.1	Na, wo ist er denn? Sprite-Zeiger	104
5.2	Einschalten der Sprites	104
5.3	Schon wieder ein Koordinatensystem: Ort der Sprites	105
5.4	Mal wieder Farbe: Diesmal die von Sprites	109
5.5	Anormale Sprites? Gequetschte und gezerrte MOBs	113
5.6	Wer hat Vorfahrt? Prioritäten	115
5.7	Zusammenstöße: Kollisionsregister	116
6	Zwei- und dreidimensionale Darstellung	119
6.1	Wir schaffen es, nichts zu sehen: Abschalten des Bildschirms	121
6.2	Das sogenannte Scrollen: Bildverschiebung	122
6.3	Dornröschen ist wieder dran: 2D-Grafik ist besser	126
6.4	Flacher Raum: 3D-Grafik, wie funktioniert das?	132
7	Das Ziel ist erreicht	143
7.1	Das Hinterschneidungsproblem: Wie verhindert man, daß verdeckte Kanten gezeichnet werden?	145
7.2	Grafik und Schrift in einem Bild	150
7.2.1	Der erste "weiche" Weg	151
7.2.2	Der zweite "weiche" Weg	152
7.3	Grafik und Maschinensprache	154
7.4	Bewegte Grafik	155
8	HIRES-3 - die Super-Grafikerweiterung	157
8.1	Hilfsfunktionen	160
8.1.1	Funktionstastenbelegung	160
8.1.2	Hilfsfunktionen als neue Basic-Befehle	162
8.2	Grafik-Befehle	163
8.2.1	Einrichten der Grafik	163
8.2.2	Zeichnen im Bildschirmsystem	164
8.2.3	Löschen im Bildschirmsystem	168
8.3	Abspeichern/Laden von Hochauflösungsbildern	169
8.4	Grafik-Befehle für frei wählbare Koordinatensysteme	169
8.4.1	Einrichten eines frei wählbaren Koordinatensystems	169
8.4.2	Zeichnen im selbstdefinierten System	171
8.4.3	Löschen im selbstdefinierten System	173
8.5	Noch zwei Kleinigkeiten	174

TEIL II 70 Farben für den Commodore 64

Einleitung		195
1	Das Mischen von Farben	197
1.1	Grundlagen	199
1.2	Senkrechte Linien	199
1.3	Horizontale Linien	201
1.4	Tabelle der angewählten Kombinationen	201
2	Die Verwendung von Mischfarben in der Praxis	207
2.1	Mischfarben im HIRES-Modus	209
2.2	Gemischte Farben im Zeichen-Modus	211
2.3	Weitere Konsequenzen	212
Anhang		213
Stichwortverzeichnis		229
Übersicht weiterer Markt & Technik Bücher		233

TEIL I

1

Grafik-Grundlagen

1 Grafik-Grundlagen

Ist es Ihnen auch so ergangen? Sie lesen Testberichte über den Commodore 64, seine hervorragenden Grafikmöglichkeiten, eine Auflösung von 320 x 200 Punkten, und Ihnen schweben die phantastischen Abbildungen von Computergrafiken vor, die Sie nun auch alle selbst realisieren können, wenn Sie diesen Computer in Händen halten. Dann, nach mehr oder weniger vielen Anstrengungen, sitzen Sie vor Ihrem eigenen Commodore 64, neben sich das 170 Seiten dicke Handbuch und arbeiten sich durch alles hindurch. Aber finden Sie diese schöne Grafik? Nachdem Sie frustriert ein paar Sprites über den Bildschirm ziehen ließen und die Ballspiele aus dem Handbuch anfangen, Sie zu langweilen, geht die Suche los, wie man denn nun eine hübsche dreidimensionale Grafik auf den Bildschirm zaubern kann: Im Handbuch ist nichts zu finden. Dann gibt es nur zwei Möglichkeiten: Entweder Sie lassen die Grafik Grafik sein, oder Sie beginnen eine Odyssee durch Buchläden, Computerzeitschriften und auch durch die Speicherzellen Ihres Computers, um sie nach langen Irrwegen endlich zu finden: die hochauflösende Grafik.

Wenn Ihre Barschaft es erlaubt, können Sie sich natürlich einiges an Schweiß ersparen: Inzwischen wird eine Reihe von mehr oder weniger brauchbarer Grafik-Software angeboten. Aber was Sie nicht bezahlen können, ist eine Menge von Erkenntnissen über die Möglichkeiten, die - verborgen hinter dornigen POKE-Hecken - darauf warten, von Ihnen wie Dornröschen wachgeküßt zu werden. Glauben Sie mir, diese Küsse sind es wert, sich in das Byte-Gewirr zu stürzen, zumal ich versuchen werde, Ihnen dazu den von mir schon gebahnten Weg in diesem Buch zu zeigen. Bei der Gelegenheit werden Sie feststellen, daß Sie nicht nur Dornröschen (die hochauflösende Grafik) wachgeküßt haben, sondern - erinnern Sie sich an die Gebrüder Grimm - auch das ganze Volk im Schloß fängt an zu leben. Mit nüchternen Worten: Sie machen sich dabei eine Menge anderer, meist unbekannter Eigenschaften Ihres Computers zunutze.

Damit Sie nicht über Begriffe stolpern, sind sie hier erklärt:

RAM	Random Access Memory = Speicher für beliebigen Zugriff, also Schreiben und Lesen (POKE und PEEK) möglich.
ROM	Read Only Memory = Speicher kann nur gelesen werden (PEEK).
Speicher	kann man sich vorstellen als lange Straße mit meist ebenerdigen Häusern und Hausnummern von 0 bis 65535.
Byte	ein Haus dieser Straße mit acht Zimmern. Man numeriert sie durch von 0 bis 7.
Bit	ein Zimmer eines solchen Hauses. Es ist entweder etwas drin (Bit gesetzt, also = 1) oder nichts drin (Bit gelöscht, also = 0).

Adreßbus	eine Art Kabinentaxi, das alle 65535 Häuser durch Angabe der Hausnummer ansteuern kann. Eine höhere Zahl als 65535 kann nicht angegeben werden.
1 KByte	einmal 1024 Bytes.
1 page	eine Seite = ein Viertel von 1 KByte = 256 Bytes.

1.1 Die Grafikmöglichkeiten des C 64

Noch einige Worte, bevor wir an die Arbeit gehen: Wissen Sie eigentlich, wie vielfältig die Grafik des Commodore 64 ist? In Bild 1.1 ist sie aufgeführt.

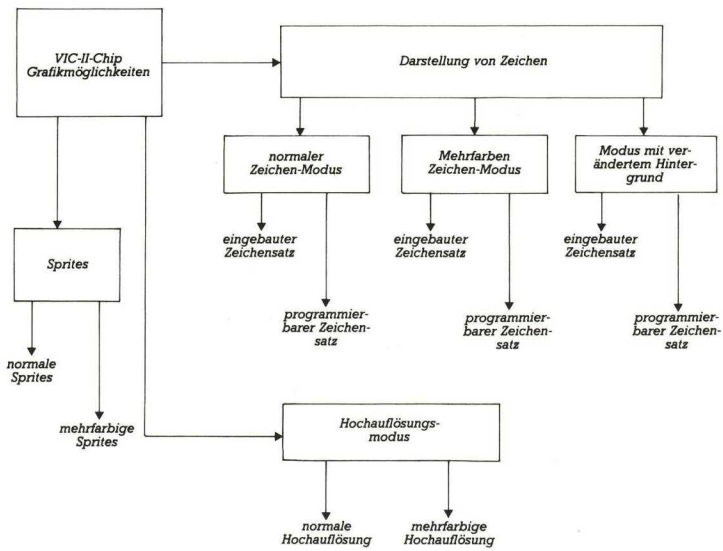


Bild 1.1 Die Vielfalt der Grafikmöglichkeiten des Commodore 64

Im Handbuch finden Sie nur zwei davon: Den "normalen" Zeichensatz und die "normalen" Sprites. Zu dem Schema in Bild 1.1 gehören eigentlich noch einige Kleinigkeiten, auf die wir noch zurückkommen werden. So kann beispielsweise der Bildschirm auf verschiedene Grafikarten aufgeteilt werden. Aber um so weit zu gelangen, müssen wir uns erst eine Weile durch die Byte-Dornen hauen.

1.2 Die Speicherorganisation des C 64

Sie dürfen schon Ihren Computer einschalten, denn wir werden bei der nun folgenden Reise durch das Grafik-Land einiges ausprobieren. Allerdings werden wir uns vorübergehend dabei von den Gebrüdern Grimm trennen müssen, denn die Landschaft, durch die wir uns dabei bewegen, paßt besser zum Wunderland der kleinen Alice: Ganze Landschaftsteile sind da und doch nicht da, Gebäude verschwinden und andere tauchen wieder auf, die Zeit wird gedehnt, Spiegelbilder erscheinen.

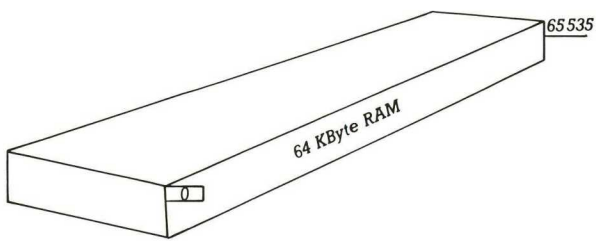


Bild 1.2 Die 64 KByte RAM pur

Wir fangen zunächst damit an, den Ast, auf dem wir sitzen, abzusägen. Damit Sie sich trotzdem keinen Schaden zufügen, sollten Sie vorher alle Programme, die Sie eventuell noch im Computer haben, auf einer Kassette oder einer Diskette abspeichern. Erledigt? Dann geben Sie folgendes ein:

```
POKE 1,PEEK(1) AND 252 <RETURN>
```

Jetzt ist Ihr Computer scheintot. Kein Cursor mehr, keine Reaktion auf Tasteneingaben. Aber dafür haben Sie jetzt tatsächlich die 64 KBytes RAM, die in der Kaltstartmeldung des C 64 angekündigt sind, zur freien Verfügung. Nur ist nichts damit anzufangen! Die 65536 freien Bytes Speicherkapazität liegen wie jungfräulicher Ackerboden vor uns und wir Benutzer sind ihnen völlig egal: Es muß also außer dem, was über einen Adreßbus von 16 Bit normalerweise erreichbar ist, noch etwas anderes vorhanden sein, etwas, das uns die Kommunikation mit unserem Computer erlaubt.

Natürlich ist dieses etwas auch jetzt vorhanden, nur sieht es der C 64 nicht. Das zwingt uns leider dazu, einige für ihn verschwundene Gebäude durch Aus- und Einschalten schlagartig wieder sichtbar zu machen. Welche Gebäude sieht der Computer jetzt wieder?

Da ist zunächst einmal das Betriebssystem (auch Kernal-ROM genannt). Alle Hausnummern unserer Byte-Straße von 57344 bis 65535 haben außer dem RAM-Erdgeschoß noch eine Etage: Im ersten Stock liegt dort das Kernal-ROM:

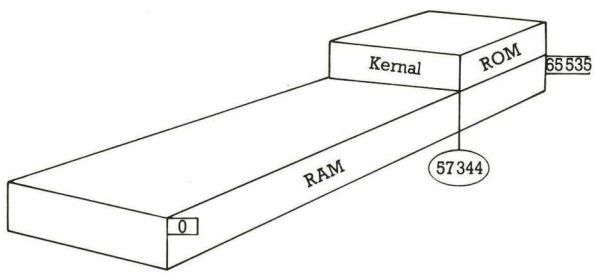


Bild 1.3 So ist das Kernal-ROM plaziert

Dieses Kernal-ROM ist sozusagen der Organisator unseres Computers - ohne es geht nichts mehr, wie wir ja eben, als es weggeschaltet war, gesehen haben. Allerdings braucht auch der beste Organisator noch einige andere lebenswichtige Partner. Damit wir überhaupt mit dem Computer in Verbindung treten können, sind noch einige weitere Hausnummern zumindest einstöckig vorhanden:

53248 bis 57343: Ein-und Ausgabebausteine.
40960 bis 49151: Basic-ROM.

Es gibt sogar Häuser mit einem zweiten Stock:

53248 bis 57343: Zeichen-ROM.

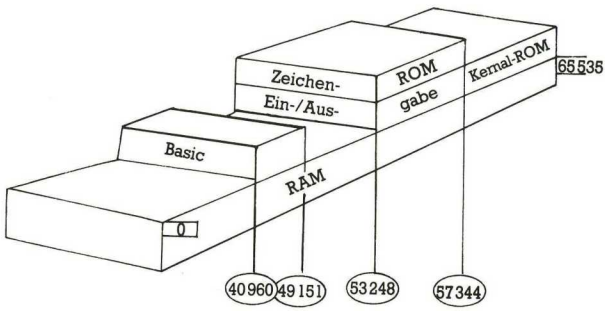


Bild 1.4 Die Ein-/Ausgabebausteine, das Zeichen- und Basic-ROM mit der entsprechenden Speicherbelegung

Zum Zeichen-ROM werden wir später kommen und die Ein- und Ausgabebausteine werden uns eine ganze Weile beschäftigen. Ohne Basic-ROM könnten wir unseren C 64 nur in Maschinensprache und nicht in Basic programmieren.

Wie kann unser Computer diese anderen Etagen nutzen? Es sind ja insgesamt statt 64 KByte jetzt 88 KByte oder exakt 90112 Zimmerfluchten zu je 8 Bit, an die man herankönnen muß. Man kann sich das so vorstellen, daß zum Beispiel zwischen den Hausnummern 53248 und 57343 einen Augenblick lang die Ein- und Ausgabebausteine stehen, dann verschwinden sie und im nächsten Augenblick steht das Zeichen-ROM dort, dann wieder an einer anderen Stelle und so weiter. Also tatsächlich ein Wunderland, das wir an dieser Byte-Straße finden. Gesteuert wird dieses Auftauchen und Verschwinden vom Betriebssystem. In Wirklichkeit bleibt alles an seinem Platz.

Man sollte meinen, daß der Commodore 64 durch diese ganzen Zaubereien, denen er sich da widmen muß, wenig Zeit für uns Benutzer hat! Aber weit gefehlt, unser Computer ist so schnell, daß für uns seine Zeit gedehnt aussieht. Der Puls des Computers rast mit zirka 1 Million Schlägen pro Sekunde, während unser Puls rund einhalbmal in der Sekunde schlägt: In der Zeit, in der unser Augenlid einmal zwinkert, hat der Computer schon tausende von Operationen vorgenommen und steht gewissermaßen mit

den Fingern trommelnd bereit, unser Kommando endlich zu empfangen. Eigentlich langweilt er sich die meiste Zeit. Wie man seine Leistungsfähigkeit effektiver als mit Basic-Programmen ausnutzen kann, werden wir später noch überlegen

Zunächst wollen wir uns ein wenig in unserem Speicher umsehen. Dazu kann das Programm "Speichertest" benutzt werden (siehe Listing). Ziemlich primitiv, für unsere Zwecke zunächst aber schon ausreichend, ist dieses kleine Programm:

```
10 INPUT"STARTADRESSE";A
20 FOR I=A TO A + 255:PRINT PEEK(I);:NEXT
30 GET A$:IF A$=""THEN 30
40 IF A$="<-"THEN A=I:GOTO 20
```

Geben Sie nach RUN als Startadresse 6000 ein. Es erscheinen Blöcke von Nullen und Blöcke von 255ern (meistens). Wenn Sie "<-" drücken, kommen die nächsten 256 Bytes auf den Bildschirm. So sieht ein leerer Speicher aus. Drücken Sie irgendeine Taste (außer "<-") und starten Sie mit RUN erneut. Mit der Eingabe von 2048 blicken wir in die ersten 256 Bytes unseres Basic-RAMs. Der wüste Zahlensalat in der oberen Hälfte des Bildschirms ist die Computer-Version unseres Programms. Danach ist dann wieder leerer Speicher zu sehen.

Wieso eigentlich 2048 als Start des Basic-RAMs? Warum nicht 0? Sehen wir uns mit ein bißchen Geduld den RAM-Bereich von 0 bis 2048 genauer an. Geben Sie Null ein und starten Sie das Programm: Wir sehen einen nahezu vollen Speicher. Das ist die sogenannte Zeropage, zu deutsch Null-Seite. Die Seite ist voll, weil sie uns das Betriebssystem abgezwackt hat, um dort eine Reihe wichtiger Werte zu speichern. Wie wichtig das ist, haben wir gesehen, als wir den Wert 55 des ersten Bytes (auf dem Bildschirm jetzt die zweite Zahl oben links) durch unser Astabsägen verändert haben. Wenn wir jetzt "<-" drücken, sehen wir die nächste Seite (page 1) auf dem Bildschirm. Auch diese Seite - obwohl sie jetzt größtenteils leer ist (Nullen und 255er-Blöcke) - gehört dem Betriebssystem: Es ist der sogenannte Prozessorstapelspeicher (auch als Stack bekannt).

Drücken wir nochmals "<-", erscheint Seite 2 (Adresse 512 bis 767). Hier ist zwar auch vieles leer (viele Nullen), aber wenn wir uns recht erinnern, sah der 'normale' leere Speicher anders aus. Auch diese Seite hat der Computer unserem Zugriff durch das normale Basic entzogen und speichert dort einige wichtige Angaben.

```
ready.


1 rem**********vorbereitungen***
2 poke52,48:poke56,48:poke53280,11:poke53281,11:poke646,0:goto 140
5 rem**up*******cursorsteuerung*
6 poke211,s:poke214,z:sys58640:return
10 rem**up******* hex ← dez******
12 he$=" ":h=int(de/16):he$=mid$(h$,h+1,1):h=de-h*16:he$=he$+mid$(h$,h+1,1)
14 return
20 rem*up*******bin ← dez ******
22 bi$=" ":di=de
24 di=di/2:d$="0":ifdi<>int(di)thend$="1"
26 di=int(di):bi$=d$+bi$:ifdi>0then 24
28 iflen(bi$)<8thenbi$="0"+bi$:goto28
29 return
30 rem*up******* dez ← bin******
32 de=0:fori=1to28:ifmid$(a$,i,1)="1"thende=de+2↑(8-i)
34 nexti:return
40 rem*up******* ausgabe *******
42 printtab(7-len(str$(ad))) ad;tab(13-len(str$(de)))de;tab(16)he$;tab(21)bi$;
44 fori=1to8:w$=mid$(bi$,i,1):ifw$="1"then printtab(30+i);chr$(113);:goto48
46 printtab(30+i)chr$(119);
48 nexti:print:return
50 rem*up******* rom kopieren **
52 printchr$(147):z=8:s=2:gosub5:print"kopieren des zeichen-rom in den ram"
54 print:print"  ab  12288. das dauert ca. 1 minute."
56 print:print"   der bildschirm wird solange leer":fori=0to2000:nexti
58 poke53265,peek(53265)and239:poke56334,peek(56334)and254:poke1,peek(1)and251
60 fori=0to4095:poke12288+i,peek(53248+i):nexti
62 poke1,peek(1)or4:poke56334,peek(56334)or1:poke53265,peek(53265)or16
64 poke53272,(peek(53272)and240)or12:k=1:return
70 rem*up******** speicherinhalt*
72 printchr$(147):z=5:s=3:gosub5:print"wir sehen in einen speicherplatz:"
74 print:print" bitte adresse (0 - 65535) angeben!":print
76 print"zurueck zum menue m. zahl <0 - 65535>"
78 input"adresse   :";ad:ifad<0orad>65535then return
80 de=peek(ad):print:print:gosub 10:gosub 20:printk$:print:gosub 40
82 print:print:goto 76
90 rem*up******* zeichen zeigen*
92 ifk=0then  gosub 50
94 printchr$(147):z=5:s=5:print"so sind die zeichen gespeichert:"
96 print:print"   bitte bildschirmcode des zeichens"
98 print"   eingeben (handbuch s.133)":print
100 print" zurueck zum menue m. zahl <0 oder >511"
102 print:input"bildschirmcode = ";a:ifa<0 ora>511then return
104 print:print:printk$:print:forad=12288+8*ato12288+8*a+7
106 de=peek(ad):gosub10:gosub20:gosub40:nextad:print:goto102
110 rem*up****** zeichen aendern *
112 ifk=0thengosub50
114 printchr$(147):z=2:s=10:gosub10:print"wir aendern zeichen:":print:print
116 print"  dazu geben sie zuerst den code des zu"
118 print"  aendernden zeichens ein. danach kann"
120 print"  der binaerausdruck veraendert werden.":print:print
122 print"zurueck zum menue mit code <0 oder >511.":print
124 input"bildschirmcode =";a:ifa<0ora>511then return
126 printchr$(147):forad=12288+8*ato12288+8*a+7
128 de=peek(ad):gosub10:gosub20:gosub40:nextad:printchr$(19):ad=ad-9
130 forj=1to8:printtab(19);:inputa$:ad=ad+1:gosub 30:gosub 10:gosub20
132 printchr$(145);"                                        ":rem 39 leerzeichen
134 printchr$(145);chr$(145):gosub 40:pokead,de:nextj
136 print:print:goto 124
140 rem
142 rem******* hauptprogramm *******
144 rem
146 rem******* variablenliste*******
```

```
148 s=1:z=1:h=0:de=0:di=0:i=0:j=0:k=0:ad=2:a=0
150 he$=" ":h$="0123456789abcdef":bi$=" ":d$="0":w$=" ":a$=" "
152 k$= "adresse  dez. hex.   bin.          bild"
154 rem****** menue *******
156 printchr$(147):z=2:s=10:gosub5:printchr$(18)"*** speichertest ***":print
158 print:print:print:printtab(5)"inhalt einer speicherzelle"tab(35)"1"
160 print:printtab(5)"zeichen im speicher zeigen"tab(35)"2"
162 print:printtab(5)"zeichen veraendern"tab(35)"3"
164 printtab(5)"programm beenden"tab(35)"4"
166 print:print:printtab(10)"ihre wahl ?";
170 rem****** abfrage *******
172 geta$:ifa$<"1"ora$>"4"then 172
174 print"    "a$
176 if val(a$)=4then180
178 onval(a$)gosub70,90,110:goto154
180 rem   *** ende ***
182 printchr$(147):z=10:s=7:gosub 5:printchr$(18)"a"chr$(146)"lten zeichensatz o
der"
184 z=12:s=7:gosub5:printchr$(18)"n"chr$(146)"euen weiterverwenden ?"
186 z=20:s=5:gosub5
187 print"bitte "chr$(18)"a"chr$(146)" oder "chr$(18)"n"chr$(146)" tippen!"
188 geta$:ifa$<>"a"anda$<>"n"then188
190 ifa$="a"then poke53272,21:poke52,160:poke56,160:end
200 end
210 rem ****************************
211 rem *                          *
212 rem *  das programm speichertest *
213 rem *                          *
214 rem *  wurde aus dem von m.thoma *
215 rem *                          *
216 rem *  und mir gebauten programm *
217 rem *                          *
218 rem *  speilu weiterentwickelt  *
219 rem *                          *
220 rem *                          *
221 rem * heimo ponnath hamburg 1985 *
222 rem *                          *
223 rem ****************************

ready.
```

Listing: Das Programm "Speichertest"

Die Seite 3 erfüllt einen ähnlichen Zweck und außerdem befindet sich dort von 828 bis 1019 noch der Kassettenpuffer. Damit hat uns das Betriebssystem unseres Computer also schon das erste KByte des Speichers gemopst. Wenn Sie sich entsinnen, habe ich vorhin erwähnt, daß das Basic-RAM bei 2048 beginnt. Wie sieht es nun im Bereich des zweiten KByte aus? Wenn wir uns mittels "<-" die nächsten vier Seiten ansehen, marschieren stramme Zahlenkolonnen mit Werten zwischen 48 und 57 (und viele 32) auf. Das sind die Bildschirm-Codes für Zahlen und Leerstellen: Hier haben wir den Bildschirmspeicher mit insgesamt 1000 Bytes und dazu noch einige Bytes, die uns bei den Sprites beschäftigen werden. Jetzt sind wir wieder beim Basic-RAM ab 2048 angelangt. Haben Sie Lust? Dann probieren Sie noch ein bißchen weiter und sehen Sie sich zum Beispiel das Basic-ROM zwischen 40960 und 49151 an, oder das Betriebssystem oder...

Dabei werden Sie dann nochmal einen freien RAM-Bereich zwischen 49152 und 53247 finden, der aber normalerweise nicht für Basic erreichbar ist. Jetzt kennen wir - bis auf einige weitere Merkwürdigkeiten, zum Beispiel die versprochenen Spiegelbilder - unseren Computerspeicher schon ganz gut und können uns dem für die Grafik wichtigsten Speicherteil zuwenden: Den Ein- und Ausgabebausteinen.

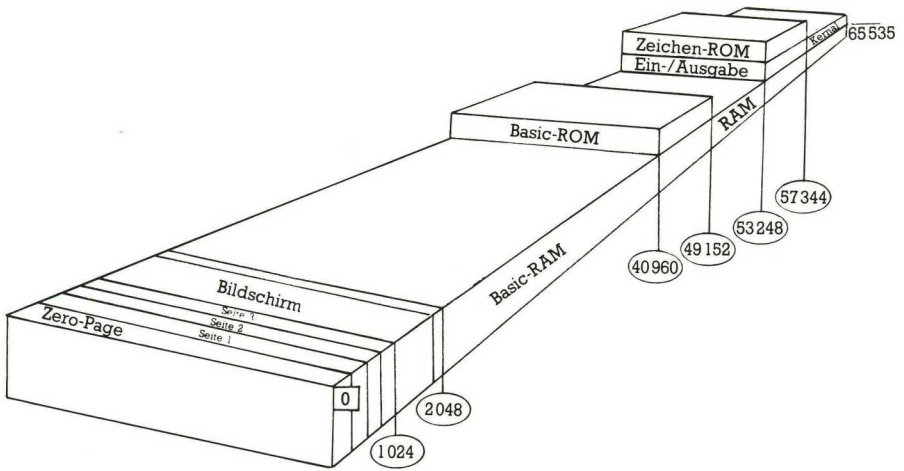


Bild 1.5 Die unteren 2 KByte des RAM-Bereichs sind ebenfalls durch das Betriebssystem belegt

1.3 Der VIC-II-Chip

Verschaffen wir uns zunächst einen Überblick:

- Der Video-Interface-Controller 6567 (VIC-II) liegt zwischen den Hausnummern 53248 und 54271. Genaugenommen ist die letzte Register-Hausnummer allerdings 53294. Alle ansprechbaren Register liegen tatsächlich nur auf 47 von den 1024 Hausnummern.

- An den VIC-II-Chip schließt sich das Sound Interface Device 6581 (SID) an, welches von Hausnummer 54272 bis 55295 reicht. Das ist ein sehr verlockendes Nachbaranwesen (Musikliebhaber kommen hier auf ihre Kosten), welches wir bei dieser Gelegenheit aber nicht besuchen wollen.

- Von 55296 bis 56319 (genauer eigentlich nur bis 56295) liegt das Farb-RAM, Dornröschens hauseigene Malerei, die wir noch bemühen werden.

- Stippvisiten werden wir uns beim Pförtner des Schlosses erlauben, der seine Wache bei den Hausnummern 56320 bis 56575 stehen hat, dem sogenannten Complex Interface Adapter 6526 (CIA Nr. 1). Die 1 rührt daher, daß er noch einen Kollegen hat, der das Revier von Adresse 56576 bis 56831 bewohnt und logischerweise CIA Nr. 2 heißt.

- Sozusagen Baugrund für Erweiterungen findet man noch zwischen den Speicheradressen 56832 und 57343.

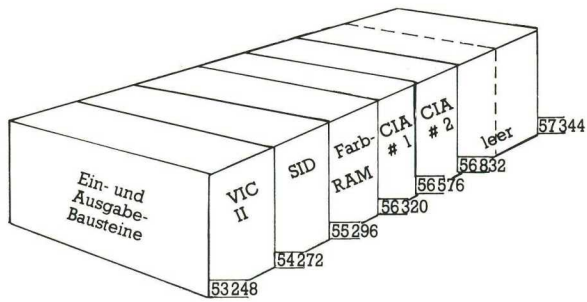


Bild 1.6 Die Ein-/Ausgabebausteine sind in weitere Untergruppen eingeteilt

Von nun an wird uns Dornröschens Schloß, der VIC-II-Chip, ständig beschäftigen. Damit die Orientierung leichter fällt, ist die Tabelle 1 abgebildet, in der alle Registerinhalte wie auf einem Grundriß verzeichnet sind. Auf den ersten Blick sieht das zugegebenermaßen reichlich verwirrend aus - lassen Sie sich nicht erschrecken.

Sie stehen jetzt sozusagen mitten im Dornengestrüpp, und wenn wir gemeinsam den Weg hindurchgefunden haben, werden Sie alles verstehen, was da steht. Fangen wir mit der Hausnummer 53272 an: Wenn Sie Lust haben, können Sie sich den Inhalt der Adresse 53272 mit dem beigefügten Programm "Speichertest" anschauen. In der Registerübersicht werden Sie beim Nachschlagen sehen, daß die Bits (Zimmer) 7-4 den Ort des Bildschirms im Speicher anzeigen und die Bits 3-0 im Normalfall etwas damit zu tun haben, wo die Zeichen (Buchstaben, Zahlen, Grafikzeichen etc.) abrufbar sind. Bevor wir daran gehen, dieses Byte zu verändern, wollen wir es im Urzustand erstmal unter die Lupe nehmen. Wenn Sie

```
PRINT PEEK(53272)
```

eingeben, werden Sie den Wert 21 ausgedruckt bekommen. Haben Sie sich diesen Speicherplatz mittels "Speichertest" angesehen, fanden Sie etwas wie

```
00010101
```

was der Binärausdruck der Dezimalzahl 21 ist. Auch dazu kommen wir noch. Das Betriebssystem setzt nach dem Einschalten das Byte 53272 automatisch auf diesen Wert, und wenn Sie sich an die Speicherreise mit dem kleinen Vierzeilen-Programm erinnern, wissen Sie auch noch, daß der Bildschirmspeicher damit auf die Startadresse 1024 festgelegt ist.

Wie Sie ebenfalls wissen, hat der Commodore 64 im Normalfall 40 Zeichen pro Zeile und 25 Zeilen, was insgesamt 1000 Zeichen pro Bild ergibt. Deswegen hat der Bildschirmspeicher eine Größe von 1000 Bytes: von 1024 bis 2023. Die Aufteilung dieser 1000 Speicherplätze auf den Bildschirm ersehen Sie aus dem Handbuch auf Seite 138. Was ist nun drin in den Bildschirmspeicherstellen? Probieren wir es aus! Tippen Sie doch mal ein:

```
<shift + clear home> ABC <Return>
```

Natürlich taucht jetzt eine Fehlermeldung auf, die uns aber nicht weiter kümmern soll. Jetzt steht ganz links oben (in Speicherplatz 1024) das A, dann B (1025) und C (1026). Nun wollen wir sehen, was der Computer sich merkt:

```
PRINT PEEK(1024), PEEK(1025), PEEK(1026) <Return>
```

Es erscheint: 1 2 3

Register	Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	53248	X-Position des Sprite Nr. 0. Dazu muß Register 16 beachtet werden							
1	53249	Y-Position des Sprite Nr. 0							
2	53250	X-Position des Sprite Nr. 1. Auch dazu, wie zu allen folgenden Sprites, muß Register 16 beachtet werden.							
3	53251	Y-Position des Sprite Nr. 1							
4	53252	X-Position des Sprite Nr. 2. s. o.							
5	53253	Y-Position des Sprite Nr. 2							
6	53254	X-Position des Sprite Nr. 3. s. o.							
7	53255	Y-Position des Sprite Nr. 3							
8	53256	X-Position des Sprite Nr. 4. s. o.							
9	53257	Y-Position des Sprite Nr. 4							
10	53258	X-Position des Sprite Nr. 5. s. o.							
11	53259	Y-Position des Sprite Nr. 5							
12	53260	X-Position des Sprite Nr. 6. s. o.							
13	53261	Y-Position des Sprite Nr. 6							
14	53262	X-Position des Sprite Nr. 7. s. o.							
15	53263	Y-Position des Sprite Nr. 7							
16	53264	Spr. 7, msb X-Pos.	Spr. 6, msb X-Pos.	Spr. 5, msb X-Pos.	Spr. 4, msb X-Pos.	Spr. 3, msb X-Pos.	Spr. 2, msb X-Pos.	Spr. 1, msb X-Pos.	Spr. 0, msb X-Pos.
17	53265	msb des Raster-registers (Reg. 18)	Schaltbit für veränderten Hintergrund-farbmodus 1 = einge-schaltet	Schaltbit für Hochauflö-sungsmodus 1 = einge-schaltet	Schaltbit für Bildschirm »aus« 0 = normaler Bildschirm 1 = Bild-schirmfarbe gleich Hinter-grundfarbe	Schaltbit für Zeilenzahl 0 = 24 Zeilen 1 = 25 Zeilen	Wert der Zeilenverschiebung in Y-Richtung beim Smooth Scrolling		

Register	Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
18	53266	Rasterregister. Dazu kommt das msb in Bit 7, Register 17							
19	53267	Lichtgriffel X-Position							
20	53268	Lichtgriffel Y-Position							
21	53269	Ein- und Ausschalten von Sprites. 0 = Sprite aus. 1 = Sprite an Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
22	53270			Reset-Bit, muß 0 sein, damit VIC-II-Chip arbeitet	Schaltbit für Mehrfarbmodus 1 = eingeschaltet	Schaltbit für Spaltenzahl 0 = 38 Spalten 1 = 40 Spalten	Wert der Spaltenverschiebung in X-Richtung beim Smooth Scrolling		
23	53271	Sprite-Vergrößerung in Y-Richtung. 0 = normale Größe. 1 = doppelte Größe. Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
24	53272	Startadresse des Bildschirmspeichers				— Startadresse des Speicherbereichs, in dem die Zeichen als Punktmatrizen abzurufen sind. — Startadresse der Bit-Map			
25	53273	Interrupt-Flaggen-Register Interrupt				Lichtgriffel-Interrupt-Flagge	Sprite/Sprite-Kollision	Sprite/Hintergrund-Kollision	Raster-Interrupt-Flagge
26	53274	Interrupt-Masken-Register Interrupt				Lichtgriffel-Interrupt-Maske	Sprite/Sprite-Koll.-Maske	Sprite/Hintergrund-Kollision Maske	Raster-Interrupt-Maske
27	53275	Sprite/Hintergrund-Prioritätenregister. 0 = Sprite hat Priorität. 1 = Hintergrund hat Priorität Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
28	53276	Sprite-Mehrfarbmodus-Register. 0 = Normaldarstellung. 1 = Mehrfarbmodus-Darstellung Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
29	53277	Sprite-Vergrößerung in X-Richtung. 0 = normale Größe. 1 = doppelte Größe Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
30	53278	Sprite/Sprite-Kollision. 0 = keine Berührung. 1 = Berührung Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
31	53279	Sprite/Hintergrund-Kollision. 0 = keine Berührung. 1 = Berührung Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0

Register	Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
32	53280		unbenutzt			Farbe des Bildschirmrahmens			
33	53281		unbenutzt			Hintergrundfarbe Nr. 0 (normale Hintergrundfarbe)			
34	53282		unbenutzt			Hintergrundfarbe Nr. 1			
35	53283		unbenutzt			Hintergrundfarbe Nr. 2			
36	53284		unbenutzt			Hintergrundfarbe Nr. 3			
37	53285		unbenutzt			Sprite-Mehrfarben-Register Nr. 0			
38	53286		unbenutzt			Sprite-Mehrfarben-Register Nr. 1			
39	53287		unbenutzt			Sprite 0, Farbe			
40	53288		unbenutzt			Sprite 1, Farbe			
41	53289		unbenutzt			Sprite 2, Farbe			
42	53290		unbenutzt			Sprite 3, Farbe			
43	53291		unbenutzt			Sprite 4, Farbe			
44	53292		unbenutzt			Sprite 5, Farbe			
45	53293		unbenutzt			Sprite 6, Farbe			
46	53294		unbenutzt			Sprite 7, Farbe			

Tabelle 1.1 Registerübersicht des VIC-II-Chips

Wenn auf dem Bildschirm ein A vorhanden ist, hat der Computer in der dazugehörigen Stelle seines Speichers eine 1 stehen, bei B eine 2, bei C eine 3 und so weiter. Lassen Sie uns den Bildschirm mit ‹shift + clear home› nochmal löschen. Dann gehen wir mit dem Cursor etwas abwärts und poken diese Kennzahlen in den Bildschirmspeicher:

```
POKE 1024,1:POKE 1025,2:POKE 1026,3 ‹Return›
```

Nach dem Return ist anscheinend nichts passiert. Erst wenn Sie mit dem Cursor dorthin fahren, wo eigentlich ABC stehen sollte, tauchen diese Buchstaben unter dem Cursor auf. Der Grund für dieses Verhalten liegt sicherlich darin, daß die Kennzahl 1 in der Speicherzelle 1024 alleine nicht genügt, das A sichtbar zu machen. Es hat automatisch die Farbe des Hintergrundes. Einem Zwilling des Bildschirmspeichers sind wir bei den Ein- und Ausgabebausteinen schon begegnet: Dem Farb-RAM zwischen 55296 und 56295. Wie er aufgeteilt ist (siehe Bild 1.7) steht im Handbuch Seite 139 zusammen mit den Farbkennzahlen.

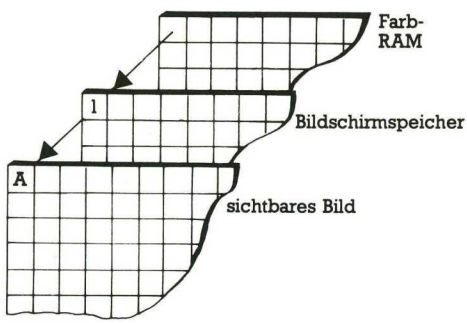


Bild 1.7 Ein Zeichen auf dem Bildschirm setzt sich aus der Bildschirm- und der Farbinformation zusammen

Wenn wir jetzt zum Beispiel noch eingeben:

```
POKE 55296,1:POKE 55297,3:POKE 55298,7‹Return›
```

sehen wir ein weißes A, ein cyanfarbenes B und ein gelbes C. Übrigens, wenn Ihnen die aktuelle Farbe des Cursors oder der gerade verwendeten Zeichen nicht gefällt, dann probieren Sie doch mal

```
POKE 646, Farbkennzahl.
```

Und weil wir gerade bei den Farben sind: die Speicherzellen 53280 und 53281 steuern, mit Farbkennzahlen belegt, die Rahmen- und die Hintergrundfarbe. Mir persönlich gefällt zum Beispiel folgende Kombination sehr gut (auf Schwarzweiß-Bildschirm):

```
POKE 53280,11:POKE 53281,11:POKE 646,0<Return>
```

Nun zu den Zeichen. Woher weiß der Computer, daß er ein A drucken muß, wenn eine 1 im Bildschirmspeicher steht? Das sagt ihm das Betriebssystem. Es teilt ihm mit, daß im Byte 53272 des VIC-II-Chips eine Kennzahl steht (Bit 1-3, Bit 0 wird nicht beachtet), die ihm wiederum sagt, wo die Muster für alle Zeichen zu finden sind. Merkwürdigerweise deutet diese Kennzahl auf eine Startadresse der Zeichenmuster von 4096! Das Zeichen-ROM, das wir bei unserer anfänglichen Speicherbegehung als 2. Stock im Bereich 53248 bis 57343 kennengelernt haben, ist davon meilenweit entfernt! 4096 liegt außerdem mitten in einem Bereich, der ständig von Basic-Programmen überschrieben wird.

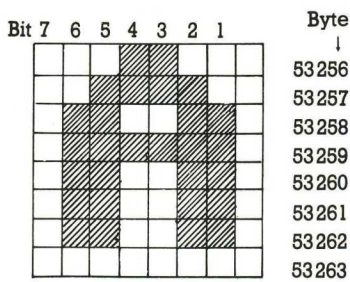


Bild 1.8 Das Zeichenmuster des Buchstaben A

Die genaue Lösung des Rätsels soll erst später gegeben werden. Aufgrund einer technischen Eigenart des VIC-II-Chips werden vom Zeichen-ROM zwei Geisterbilder im Bereich 4096 bis 8191 und im Bereich 36864 bis 40959 erzeugt. Der VIC-II-Chip

"meint", er hole Zeichen-Muster aus diesen Bereichen. In Wirklichkeit bezieht er sie im Normalfall immer aus dem Zeichen-ROM. Das ist eine Eigenart, die so recht in Alices Wunderland paßt!

Wie sehen diese Zeichen Muster aus? Auch dazu können Sie das Programm "Speichertest" benutzen. Wenn Sie sich damit beispielsweise mal das A ansehen, finden Sie ein Muster, wie es in Bild 1.8 dargestellt ist.

Dieses 8 x 8-Gitter (auch Matrix genannt) enthält das Abbild des Zeichens A. Alle Zeichen sind auf diese Weise als Punktmuster in jeweils acht Speicherzellen (hier also von 53256 bis 53263) gespeichert. Ein dunkles Feld bedeutet ein gesetztes Bit (=1; im Zimmer ist etwas drin), ein helles Feld ein gelöschtes Bit (=0; das Zimmer enthält nichts).

Das Zeichen-ROM hat an nullter Stelle von 53248 bis 53255 das Zeichen mit dem Bildschirmcode 0 (den Klammeraffen @), an erster Stelle von 53256 bis 53263 - wie wir sehen - das Zeichen mit dem Bildschirmcode 1 (also das A) nacheinander in Form von je acht Bytes als Bit-Muster gespeichert. Wenn Sie im Handbuch die Seite 133 aufschlagen, können Sie die nachfolgende Tabelle 2 mit dem Inhalt des Zeichengenerator-ROMs besser verstehen.

Block	Adressenbereich	Zeichen	Muster abrufbar im Programm mit Code
0	53248 — 53759	Satz 1 von 0 bis 63 (0 bis ?)	0 — 63
	53760 — 54271	Satz 1 von 64 bis 127 (Grafikz.)	64 — 127
	54272 — 54783	Satz 1 von 0 bis 63 (Reversed)	128 — 191
	54784 — 55295	Satz 1 von 64 bis 127 (Reversed)	192 — 255
1	55296 — 55807	Satz 2 von 0 bis 63 (kleine Buchst.)	256 — 319
	55808 — 56319	Satz 2 von 64 bis 127 (Großbuchst. + Grafikz.)	320 — 383
	56320 — 56831	Satz 2 von 0 bis 63 (Reversed)	384 — 447
	56832 — 57343	Satz 2 von 64 bis 127 (Reversed)	448 — 511

Tabelle 1.2 Inhalt des Zeichen-ROMs

Probieren Sie aus, wie sich die einzelnen Zeichen mit dem Programm "Speichertest" durch Eingabe der Bildschirmcodes (im Handbuch bis 127 als Pokes bezeichnet) abbilden lassen.

Das Programm "Speichertest" enthält noch einige für Sie bislang noch geheimnisvolle Einzelheiten: die Hexadezimal- und die Binärzahlen, das Interrupt-System, das Kopie-

ren des Zeichen-ROMs. Dies alles hängt mit der Frage zusammen: Wie kann man sich eigene Zeichen bauen und verwenden? Wir werden sie in den nächsten Kapiteln gemeinsam beantworten.

2

Zahlensysteme und Speicherorganisation

2 Zahlensysteme und Speicherorganisation

2.1 Die Begegnung mit den Zweifingerlingen: Das Binärsystem

Im Grunde genommen haben sie uns schon eine ganze Weile ungesehen begleitet: Die Zweifingerlinge. Um sie für uns sichtbar zu machen, bedarf es eigentlich nur einiger Gedankenübungen. Beobachten Sie einmal kleine Kinder beim Zählen oder Rechnen: Das läuft Finger für Finger. Wir haben zehn davon (im allgemeinen) und haben deswegen wohl auch neun Ziffern und die Null:

1, 2, 3, 4, 5, 6, 7, 8, 9, 0

Um eine Zahl auszudrücken, die größer als 9 ist, zum Beispiel 9+1, setzen wir einfach zwei von diesen Ziffern zusammen und fangen wieder bei der kleinsten Ziffer 1 an und hängen eine Null dran: 10. Auf diese Weise können wir jeder Anzahl von Dingen eine Zahl zuordnen. Was wäre, wenn wir nur zwei Finger hätten? Wir hätten dann - wie die im Computer herumwimmelnden Zweifingerlinge - nur zwei Ziffern: 1 und 0.

Um nun eine Zahl auszudrücken, die größer als unsere größte Ziffer (1) ist, würden wir auch so verfahren wie die Zehnfingerwesen. Also fangen wir wieder bei der kleinsten Ziffer an (die hier auch gleichzeitig die größte und einzige ist), also der 1 und hängen eine Null dran: 10. Wir zählen jetzt 1, 10, 11, 100, 101, 110, 111, 1000, 1001 und so weiter.

Die Zehnfingerlingzahlen dafür sind: 1, 2, 3, 4, 5, 6, 7, 8, 9 und so weiter.

Die Zweifingerlinge würden zu meinem guten alten R4 sagen: "Dieser R100 hat 100 Zylinder und 100010 PS". Leider - oder von der Steuer her betrachtet Gottseidank - hat sich dadurch aber an der Tatsache nichts geändert, daß er genauso schwach den Berg hinaufklettert wie vorher, nur das Zahlensystem, das ist jetzt binär. Sehen wir uns das nochmal genauer an. Wissen Sie noch, was in der Mathematik Potenzen sind? Falls nicht, 10 hoch 3 (10^3) heißt 10 mal 10 mal 10, also die Zehn dreimal mit sich selbst multipliziert. 10 ist allerdings 10 hoch 1 (10^1) und 10 hoch 0 (10^0) ist 1. Wenn wir nun eine normale Dezimalzahl (eine Zahl der Zehnfingerlinge) vor uns haben, zum Beispiel 255, kann man dafür auch schreiben:

$$255 = 2*10^2+3*10^1+5*10^0 = 2*100+5*10+5*1$$

Rechnen Sie nach: Es stimmt! Genauso ist auch eine Binärzahl aufgebaut:

$$1001 = 1*2^3+0*2^2+0*2^1+1*2^0$$

Deswegen ist es auch relativ einfach, die Zahlen der Zweifingerlinge in unser Zehnfinger-System umzurechnen:

$$1001 = 1*2^3+0*2^2+0*2^1+1*2^0 = 1*8+0*4+0*2+1*1 = 9$$

Die bequemste Methode ist es, ein Schema wie in Bild 2.1 zu benutzen.

Bit	7	6	5	4	3	2	1	Ø	
	2^7 =	2^6 =	2^5 =	2^4 =	2^3 =	2^2 =	2^1 =	2^0 =	
binär	128	64	32	16	8	4	2	1	dezimal
1 1 1 1 1 1 1 1	1	1	1	1	1	1	1	1	255
1 0 0 0 0 1 1 1	1	0	0	0	0	1	1	1	135

Bild 2.1 Schema zur Umrechnung des Dual- in das Dezimalsystem

Andersherum kann man auch ganz einfach unsere Zahlen in die der Zweifingerlinge umrechnen, nämlich wie in Bild 2.2 gezeigt.

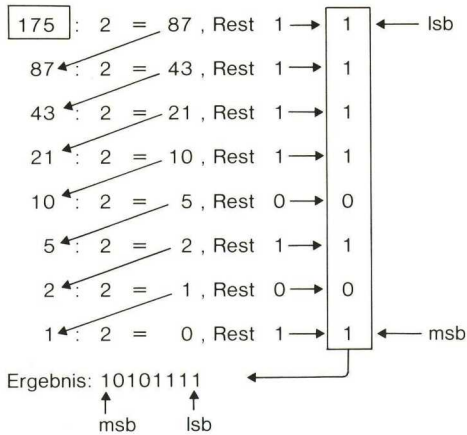


Bild 2.2 Schema zur Umrechnung des Dezimal- in das Dualsystem

Dabei bedeutet lsb "least significant bit" und msb "most significant bit", also zu deutsch etwa "bedeutsamstes Bit" und "am wenigsten bedeutsames Bit". Das ist leicht zu verstehen: Es macht keinen so großen Unterschied, ob mir jemand 1001 Mark oder 1002 Mark schenkt. Mehr berührt es mich wenn es um 1001 oder 2001 Mark geht. Für ökonomisch Denkende sei noch bemerkt, daß das Programm Speichertest ein schönes Unterprogramm (Zeilen 20 bis 29) enthält, welches beliebige Dezimalzahlen in Binärzahlen umrechnet.

Vielleicht haben Sie Lust, zur Übung noch ein bißchen zu rechnen (aber bitte nicht mit dem Unterprogramm):

a) Umrechnung dez -> binär 25, 16, 47, 128
b) Umrechnung binär -> dez 10001, 1110, 11110000

Die Lösungen finden Sie am Ende dieses Kapitels.

Wir haben die Zweifingerlinge ausgiebig kennengelernt und werden mit ihrer Hilfe später einige Hürden nehmen können. Jetzt aber zu den Sechzehnfingerlingen.

2.2 Die Enttarnung der Sechzehnfingerlinge: Das Hexadezimalsystem

Diese Sechzehnfingerlinge begleiten uns auch ständig und zwar als Zehnfingerlinge getarnt. Doch bevor wir sie enttarnen, müssen wir etwas mehr über sie wissen. Die Zählweise der Sechzehnfingerlinge mutet uns, die wir nur zehn Finger haben (also auch nur 9 Ziffern und die Null) etwas merkwürdig an, denn was nimmt man - für Ziffern größer als 9 - ohne eine zweistellige Zahl zu benutzen? Wir bemühen das Alphabet:

0, 1, 2, 3, 4, 5 , 6, 7, 8, 9, A, B, C, D, E, F

Als Zehnfingerlinge würden wir dafür schreiben:

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

Sobald es um eine Anzahl größer als F (dez.=15) geht, zum Beispiel F+1, kommt auch hier die zweite Stelle dazu: F+1=10 (dezimal wäre das: 15+1=16). Ebenso wie die Dezimalzahlen sind auch die Hexadezimalzahlen auf Potenzen aufgebaut:

$$\$831 = 8*16^2+3*16^1+1*16^0 = 8*256+3*16+1*1 = 2097$$

Dabei steht das Dollarzeichen dafür, daß es sich um eine Hexadezimalzahl handelt, was allgemein üblich ist. Die Umrechnung von Hex-Zahlen in Dezimalzahlen und um-

gekehrt ist etwas beschwerlicher als die von Binärzahlen. Man behilft sich am besten mit der nachfolgenden Tabelle.

$	dez.	$	dez.	$	dez.	$	dez.
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

Tabelle 2.1 Umrechnungstabelle von Hex- in Dezimalzahlen

Damit geht die Umrechnung einer Hex-Zahl in eine Dezimalzahl relativ einfach.

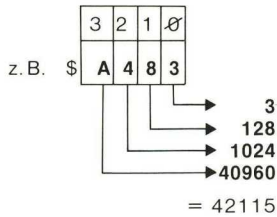


Bild 2.3 Umrechnung einer Hex- in eine Dezimalzahl

Man sucht sich aus der Tabelle in der Spalte, die dem Stellenwert der Hex-Ziffer entspricht, die dazu gehörige Dezimalzahl. Das führt man für alle Hex-Stellen durch und addiert dann die so gefundenen Dezimalwerte wie es im Beispiel gezeigt wird.

Auch die Umwandlung einer Dezimalzahl in die Hex-Zahl ist jetzt nicht mehr so tragisch. Nehmen wir die Dezimalzahl 40959 (siehe Bild 2.4). Wir suchen aus der Tabelle die größte Dezimalzahl heraus, die gerade noch kleiner als unsere Zahl ist: 36864. Ihr entspricht ein $9 an der 3. Stelle der zu findenden Hex-Zahl. Wir ziehen dann diese Zahl 36864 von unseren 40959 ab. Es bleiben 4095. Wieder suchen wir jetzt eine Stelle tiefer (also in Spalte 2) die größte Dezimalzahl heraus. Jetzt ist das 3840. Erneut folgt eine Subtraktion und so weiter wie im Beispiel im Bild 2.4 gezeigt wird.

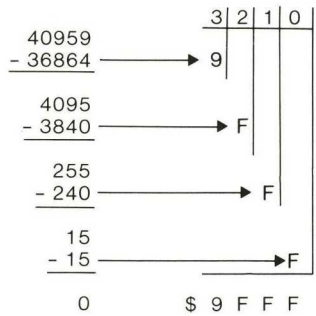


Bild 2.4 Umrechnung einer Dezimal- in eine Hex-Zahl

Ich bewundere Ihren Mut, daß Sie mit mir auf dieser Etappe soweit gefolgt sind. Wir stecken jetzt anscheinend total im Dornendickicht fest . Aber nur noch eine letzte Anstrengung und wir kommen zu einer kleinen Lichtung, auf der wir uns etwas ausruhen können. Dazu werden wir nun die Sechzehnfingerlinge enttarnen.

Sehen wir uns dazu die Zahl $FFFF an, die größte mit vier Stellen darstellbare Hex-Zahl. Wenn Sie per Tabelle umrechnen, werden Sie feststellen, daß wir 65535 vor uns haben. Erinnern Sie sich an Kapitel 1, wo diese Zahl die Obergrenze unseres gesamten Speichers war? Dann erinnern Sie sich sicherlich auch noch daran, daß in einer Speicher-Hausnummer (Byte) acht Zimmer (Bits) sind, die entweder leer oder voll sein konnten (0 und 1). In Bild 2.1 ist die Bit-Numerierung zu sehen und wie voll man so eine Hausnummer machen kann. Wie man aber auch dabei erkennt, ist die größte Zahl, die in einem Byte Platz findet 1111 1111 oder dezimal 255 oder - rechnen Sie nach - $FF im Hex-System.

Nun kann man sich ja vorstellen, daß zum Beispiel das Betriebssystem häufig im Verlauf der Benutzung irgendwelche Hausnummern angeben muß: zum Beispiel wo eine Routine zu finden ist, ein Text und so weiter. Aber auch das Betriebssystem ist darauf festgelegt, daß in jeder Hausnummer nur 8 Bits vorhanden sind! Wie kann es dann aber zum Beispiel sagen, daß die Basic-Warmstartadresse bei 42115 liegt, wenn es nur bis 255 zählen kann? Ganz einfach: Es gibt sozusagen Wohngemeinschaften mit zwei Hausnummern. Wenn alle Bits von 2 Bytes voll sind, haben wir: 1111 1111 1111 1111 = dezimal 65535. Wenn alle leer sind, haben wir 0. Mit 2 Byte-Adressen kann der ganze Bereich erfaßt werden.

In Bild 2.3 haben wir berechnet, daß der Adresse 42115 die Hex-Zahl $A483 entspricht. Jetzt verteilen wir diese Hex-Zahl auf zwei Bytes, von denen das eine MSB (most significant Byte = bedeutsamstes Byte) und das andere LSB (least significant Byte = am wenigsten bedeutsames Byte), genannt wird, wie vorhin msb und lsb bei den Bits.

MSB <- -> A4
LSB <- -> 83

Beide sind kleiner als $FF und können deswegen im Speicher untergebracht werden. Das Betriebssystem notiert sie sich in den Hausnummern 770 und 771 auf page 3. Sehen wir doch einfach mal nach! Geben Sie ein:

```
PRINT PEEK(770), PEEK(771) <RETURN>
```

Wir erhalten: 131 164

Lassen Sie sich nicht verwirren! Rechnen wir diese Angaben in Hex-Zahlen um. Wir finden die folgenden Werte:

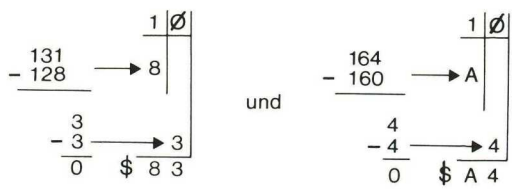


Bild 2.5 Umrechnung von 131 (LSB) und 164 (MSB) in Hex-Zahlen

Das sieht alles komplizierter aus als es ist. Mit etwas Übung, die wir uns jetzt zulegen wollen, werden Sie feststellen, daß Sie allerhand damit anfangen können. Die Sechzehnfingerlinge sind als Dezimalzahlen getarnt gewesen.

Speicher:	770	771
dezimal:	131	164
hex:	$ 83	$ A4
also	LSB	MSB
$ A483 = dez. 42115		

Bild 2.6 Inhalt der Speicherstellen 770 und 771

Auch hier zur Übung einige Aufgaben:

c) Umrechnung hex -> dez $92, $D728, $A001
d) Umrechnung dez -> hex 65534, 2048, 21235
e) Tun Sie so, als müßten Sie diese letzte Zahl in die Speicherzellen 770 und 771 eingeben. Welche POKEs sind nötig?

So, jetzt wo wir die Hex-Zahlen erkennen können, werden wir uns ihrer kräftig bedienen.

2.3 Wir führen den C 64 hinters Licht: Eigene Änderungen an der Speicherorganisation

Jetzt können wir geistig etwas ausspannen. Falls Sie Ihren Computer schon in Betrieb haben, speichern Sie darauf befindliche Programme ab, schalten Sie ihn aus und wieder ein. Wir wollen den Computer im Ursprungszustand etwas untersuchen und dann einige Änderungen vornehmen. Auf der Zeropage gibt es einige nützliche Hausnummern, die wir uns ansehen wollen. Tippen Sie doch mal ein:

```
PRINT PEEK(43), PEEK(44) <RETURN>
```

Wir erhalten: 1 8

Die Umrechnung mit der Tabelle ergibt $ 801 = dez. 2049. Sehen wir in das Handbuch, Anhang Q auf Seite 160. Dort ist zu lesen, hier sei die Startadresse vom Basic-Text gespeichert. Jetzt machen wir uns das etwas komfortabler. Wir geben im Direktmodus (also ohne Programmzeilennummer) ein:

```
A = 45:PRINT PEEK(A), PEEK(A+1), PEEK(A) + PEEK(A+1)*256 <RETURN>
```

Wir erhalten: 3 8 2051

Dies ist die Adresse, von der an Variable gespeichert werden. Gleichzeitig erfährt man so, wo ein Basic-Programm aufhört, denn die einfachen Variablen werden direkt hinter dem Basic-Programmtext gespeichert. Jetzt fahren wir den Cursor hoch auf die 45 in der zuletzt eingegebenen Zeile und ändern sie in 47, und drücken dann <RETURN>. Es erscheint:

```
10  8  2058
```

Ab 2058 beginnen jetzt die indizierten Variablen. Normalerweise fangen sie im Leerzustand auch bei 2051 an. Wir haben aber eine Variable A definiert und die verschiebt die indizierten Variablen um 7 Bytes. Als Nebeneffekt sehen wir so, daß eine Variable in 7 Bytes 'gelagert' wird. Zur Kontrolle geben wir nochmal ein:

```
CLR:PRINT PEEK(47), PEEK(48) <RETURN>
```

und erhalten:

```
3        8
```

na also!

Diese Untersuchung können Sie noch weiterführen, wenn Sie wollen. Sie erhalten so die Werte in Tabelle 2.2.

Adresse			Normaler Inhalt im Leerzustand			
Commodore Name (Label)	LSB	MSB	LSB	MSB	Dezimal	Bedeutung
TXTTAB	43	44	1	8	2049	Anfang Basic-Text
VARTAB	45	46	3	8	2051	Variablenstart
ARYTAB	47	48	3	8	2051	Arraystart
STREND	49	50	3	8	2051	Arrayende + 1
FRETOP	51	52	0	160	40960	Stringstart
MEMZIZ	55	56	0	160	40960	höchste Basic-Adresse
MEMSTR	641	642	0	8	2048	RAM-Start für Betriebssystem
MEMSIZ	643	644	0	160	40960	RAM-Ende für Betriebssystem

Tabelle 2.2 Die wichtigsten Adressen mit ihren Inhalten in der Zeropage.

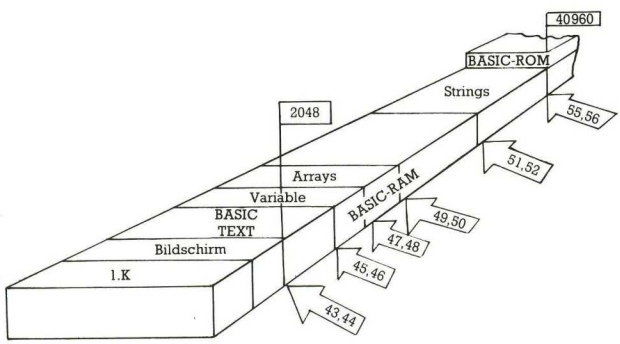


Bild 2.7 Der freie Basicbereich mit den entsprechenden Zeigern

Bevor wir jetzt an die erste Änderung gehen, sehen wir nach, wieviel freies Basic-RAM wir zur Verfügung haben:

```
PRINT FRE(0) + 65536 <RETURN>
```

Es sollte auch bei Ihnen erscheinen: 38909.

Nun wollen wir unserem Computer einreden, sein Basic-Speicher sei schon bei 12288 statt bei 40960 zu Ende. Zunächst müssen wir umrechnen. Wir sehen in die Tabelle und rechnen entsprechend Bild 2.8 um.

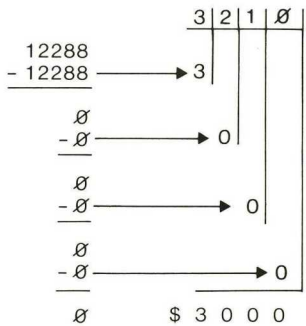


Bild 2.8 12288 dezimal in eine Hex-Zahl umgewandelt

Also ist das MSB = $30 und das LSB = $00. Jetzt rechnen wir wieder ins Dezimalsystem um:

3*16 = 48		0*16 = 0
0* 1 = 0	und	0* 1 = 0
---		---
MSB = 48		LSB = 0

Die höchste Basic-Adresse ist (siehe Tabelle 2.2) in MEMSIZ gespeichert und deshalb geben wir ein:

```
POKE 55,0:POKE 56,48 <RETURN>
```

Nun sehen wir mit

```
PRINT FRE(0)
```

nach und erhalten 10237.

Es ist geglückt. Der noch freie RAM-Bereich oberhalb von 12288 wird für Basic vom Computer nicht mehr wahrgenommen. Das nennt man "schützen" eines Speicherbereiches vor dem Überschreiben durch Basic. Gleichzeitig sollte man, falls im Basic-Programm auch Strings verwendet werden, auch noch FRETOP berücksichtigen:

```
POKE 51,0:POKE 52,48 <RETURN>.
```

In "Speichertest" wird dieser Schutz in Zeile 2 vollzogen. Jedesmal, wenn man einen Teil des Basic-Speichers für andere Dinge verwenden will als für Basic, muß man diesen Teil in der gezeigten Weise schützen.

Sie werden vielleicht sagen, daß Sie soooo lange Basic-Programme kaum verwenden und nie in Regionen über 20000 oder 25000 geraten werden. Leider ist das ein Irrtum. Denn die Speicherung von Strings geschieht ab der Adresse 40960 abwärts (siehe Bild 2.7). Ein "sicherer" Bereich im Basic-Speicher (ohne ihn schützen zu müssen) könnte höchstens irgendwo ungewiß mitten drin sein, wo weder von unten das Basic-Programm mit seinen Variablen noch von oben die Strings anstoßen würden. Darauf würde ich mich aber lieber nicht verlassen. Schützen ist besser. Wir werden im Verlauf der weiteren Folgen noch eine Reihe weiterer Möglichkeiten benutzen, um die Speicherorganisation umzukrempeln.

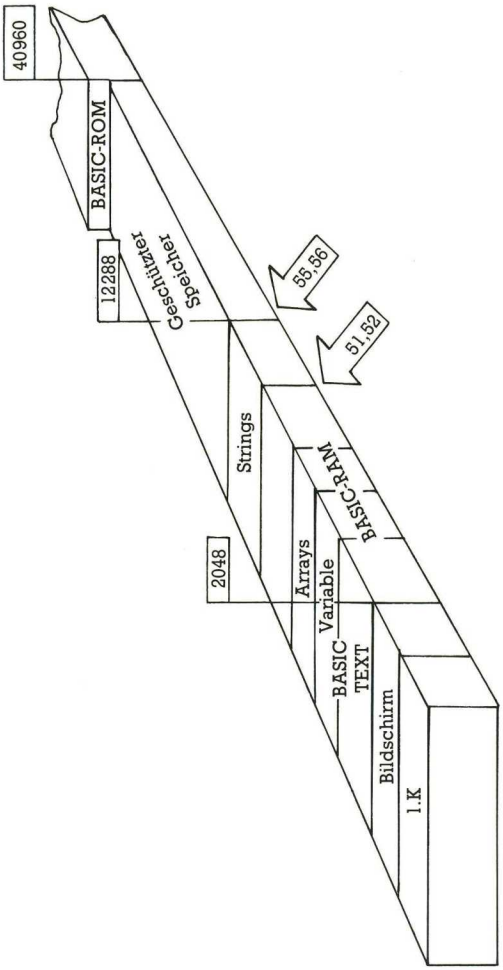


Bild 2.9 Der geschützte Bereich von 12288 bis 40960

2.4 Und oder? Oder und? Die Befehle AND, OR

Jetzt haben wir beinahe alles Handwerkszeug beieinander, um unabhängig von irgendwelchen Fertigprogrammen uns selbst neue Zeichen zu definieren und auch zu bestimmen, woher der Computer sie dann holen soll. Nur eine Tatsache stört noch. Wir wissen jetzt zwar, wie wir in unserem C 64 ganze Bytes ändern können indem wir Adressen POKE-fertig umrechnen und dann benutzen. Was tun wir aber, wenn - was uns häufig beschäftigen wird - nicht das ganze Byte, sondern nur ein halbes (ein sogenanntes Nibble) oder gar nur ein einziges Bit verändert werden soll? Natürlich gibt es dann fast immer die Möglichkeit, durch ein PEEK nachzusehen, was im Byte drin ist, das dann ins Binärsystem umzurechnen und dann schließlich einzuPOKEn.

Sehr umständlich! Basic sei Dank gibt es da zwei Befehle, die uns den Aufwand verringern helfen: AND und OR. Es handelt sich um sogenannte logische Operatoren, die zwei Dinge oder Aussagen miteinander verbinden und daraus ein Ergebnis produzieren. Zunächst mal zu AND. Wir kennen das von Basic her zum Beispiel in der IF...THEN...-Verzweigung:

```
5 IF A = 2 AND B = 200 THEN 10
```

Nur dann, wenn A = 2 und B = 200 ist, erfolgt ein Sprung nach Zeile 10, das heißt, wenn beide miteinander verknüpften Bedingungen erfüllt sind, ist das Ergebnis der Verzweigung erzielt. Bei binären Zahlen ist das einfacher:

1 AND 1 = 1.

Wenn beide Ziffern 1 sind, ist das Ergebnis 1. Man faßt das gerne in einer Tabelle zusammen.

AND	1	0
1	1	0
0	0	0

Tabelle 2.3 Die AND-Verknüpfung

Wie wendet man das an? Nehmen wir an, ein Byte sähe binär so aus: 1111 1011. Wir möchten es so verändern, daß es zu 00001011 wird. Dazu verwenden wir die AND-Operation:

```
                1111 1011 unser Byte
                0000 1111 eine sogenannte Maske
AND             ---------
                0000 1011 unser Ergebnis
```

Das heißt, alle Bits, die mit einer 1 AND-verknüpft worden sind, bleiben unverändert. Alle Bits, die dagegen mit einer 0 AND-verknüpft wurden, sind jetzt 0. Anstelle der ganzen Rechnerei muß jetzt nur die Maske umgerechnet werden: 00001111 = 15 dezimal. Nehmen wir an, unser Byte wäre die Adresse 770, dann sähe die Änderung jetzt so aus:

```
POKE 770,PEEK(770) AND 15
```

Halt! Geben Sie das aber nicht wirklich ein, denn damit verändern Sie den Basic-Warmstart-Vektor und den brauchen wir noch. Sollten Sie es schon getan haben, erfreuen Sie sich noch ein wenig des Effektes und ziehen Sie dann die Notbremse: Computer aus- und wieder einschalten.

Nun zu OR. Auch das kennen wir vom Basic her, zum Beispiel:

```
5 IF A = 2 OR B = 200 THEN 10
```

Wenn A = 2 oder wenn B = 200 ist oder wenn beide Bedingungen erfüllt sind, erfolgt der Sprung nach 10. Ebenso wie für AND kann man auch hier die Verhältnisse am besten mit einer Tabelle übersehen.

OR	1	0
1	1	1
0	1	0

Tabelle 2.4 Die OR-Verknüpfung

Die Anwendung sieht dann so aus (wir nehmen unser Ergebnis von vorhin 0000 1011):

```
                0000 1011
                1111 0000 eine Maske
OR              ---------
                1111 1011 das neue Ergebnis
```

Überall dort, wo mindestens eine 1 steht, ergibt sich im Endergebnis auch eine 1.

Während man mit AND gezielt Bits löschen kann, vermag man mit OR gezielt Bits zu setzen. Beide Operationen können natürlich auch miteinander kombiniert werden. Es gilt die alte Jungprogrammiererregel: Probieren, probieren, ...

Ein Beispiel stelle ich Ihnen nochmal genau vor. Erinnern Sie sich, daß wir das Byte 53272 etwas genauer angesehen haben? Die unteren 4 Bits (genau genommen ohne Bit 0) geben an, wo die Punktmuster für die Zeichen abrufbereit stehen. Durch PEEK(53272) fanden wir den Dezimalwert 21. Das entspricht dem Binärwert

0001 [010] 1,

wobei der eingerahmte Teil für den Ort der Zeichen zuständig ist. In der Tabelle 2.5 sehen Sie, welche Kombinationen auf welche Speicherorte als Startadressen unserer Zeichenmuster deuten.

Zahlenwert Bits 0-3 (Bit 0 = 0)	Bitmuster des Byte 53272	Startadresse Zeichenspeicher dez.	hex	
0	XXXX000X	0	$0000	
2	XXXX001X	2048	$0800	
4	XXXX010X	4096	$1000	Einschaltzustand
6	XXXX011X	6144	$1800	
8	XXXX100X	8192	$2000	
10	XXXX101X	10240	$2800	
12	XXXX110X	12288	$3000	
14	XXXX111X	14336	$3800	

Tabelle 2.5 Das Bitmuster und die Startadresse des Zeichenspeichers im Byte 53272

Wenn wir nun einen anderen Ort eingeben wollen, dürfen wir nur die Bits 1 bis 3 verändern. Lassen Sie uns die Zeichen nicht mehr von 4096 an, sondern von 6144 an gespeichert haben! Zunächst einmal müssen die Bits 4 bis 7 vor jeder Änderung geschützt sein und die Bits 0 bis 3 gelöscht werden:

```
dez. 21         0001 0101 das ist unser PEEK(53272)
dez. 240        1111 0000 eine Maske, die AND-verknüpft wird
------------------------
dez. 16         0001 0000 das ist: PEEK(53272) AND 240
```

Jetzt können wir gezielt Bits setzen. Wir brauchen die Kombination 011X. Wenn wir für X einfach 0 annehmen (das geht, weil Bit 0 hier nicht beachtet wird), ergibt das einen Dezimalwert von 6 (siehe Tabelle 1.5).

```
dez. 16         0001 0000 unser Zwischenwert
dez. 6          0000 0110 Maske wird OR-verknüpft
------------------------
dez. 22         0001 0110 unser Endwert
```

Alles in allem geben wir ein:

```
POKE 53272,(PEEK(53272) AND 240) OR 6
```

Das können Sie gefahrlos eingeben und sich am Ergebnis freuen. Wenn Sie danach mit PEEK(53272) abfragen, werden Sie nicht 22, sondern 23 erhalten, was an Bit 0 liegt, das wir so nicht beeinflussen können.

2.5 Frankensteins freundliches Monster: Eigene Zeichen

Wieso kann man eigentlich eigene Zeichen definieren, wo es sich doch um ein Zeichen-ROM handelt, woraus der C 64 seine Zeichen bezieht? Zum Umbauen der Zeichen muß man in die Punktmatrix hineinschreiben und das geht nur ins RAM. Na, dann kopieren wir einfach das Zeichen-ROM in den RAM-Bereich. Dort können wir dann nach Herzenslust herumPOKEn. Dazu muß man allerdings wissen, daß drei Dinge zu beachten sind:

a) Sobald wir dem Computer gesagt haben, wo er seine Zeichen herholen soll, kennt er alle die Zeichen nicht mehr, die nicht mit kopiert worden sind. Man muß sich also vorher überlegen, welche Zeichen man braucht und erst dann kopieren (siehe dazu in Kapitel 1 die Tabelle 1.2).

b) In Kapitel 1 ist erwähnt worden, daß der Computer so gebaut ist, daß er ständige Wechsel zwischen den Etagen unseres Speichers durchführt. Außerdem verrichtet er noch eine Reihe anderer Tätigkeiten nach einem schnell vor sich gehenden System von lauter Unterbrechungen. Beim Kopiervorgang sollte keine Unterbrechung stattfinden, weil sich der Computer solange auf das Zeichen-ROM konzentrieren soll. Man muß dazu das sogenannte Interrupt-System während des Kopierens abschalten.

c) Wir kopieren unsere Zeichen ins RAM, müssen den dafür verwendeten Speicherraum vor dem Überschreiben durch ein Basic-Programm schützen.

Es empfiehlt sich folgende Vorgehensweise:

A. Schützen des RAMs. Dazu wollen wir den Bereich verwenden, der auch in "Speichertest" eine Rolle spielt:

```
10 POKE 52,48: POKE 56,48
```

Für das folgende sollten Sie sich auch das Unterprogramm ab Zeile 50 von "Speichertest" ansehen.

B. Abschalten des Interrupt. Das macht einen Besuch beim CIA #1, Hausnummer 56334 nötig. Mit einer AND-Operation knipsen wir die Unterbrechung ab:

```
20 POKE 56334, PEEK(56334) AND 254
```

C. Nachdem der Computer nicht mehr per Interrupt die Etagen abläuft, muß er behutsam zum Zeichen-ROM geführt werden:

```
30 POKE 1, PEEK(1) AND 251
```

Behutsam deswegen, weil wir Byte 1 aus der ersten Folge noch in unguter Erinnerung haben. Wir werden es später besser kennenlernen.

D. Nun steht dem Kopieren nichts mehr im Wege. Wir kopieren alles:

```
40 FOR I=0 TO 4095:POKE 12288+I,PEEK(53248+I):NEXT
```

Das dauert allerdings eine Weile.

E. Nun muß das Interrupt-System wieder in den Ausgangszustand zurückversetzt werden:

```
50 POKE 1,PEEK(1) OR 4
60 POKE 56334,PEEK(56334) OR 1
```

F. Jetzt teilen wir dem Computer mit, daß er in Zukunft seine Zeichen ab 12288 und nicht mehr im Zeichen-ROM findet:

```
70 POKE 53272, (PEEK(53272)AND 240)OR 12
```

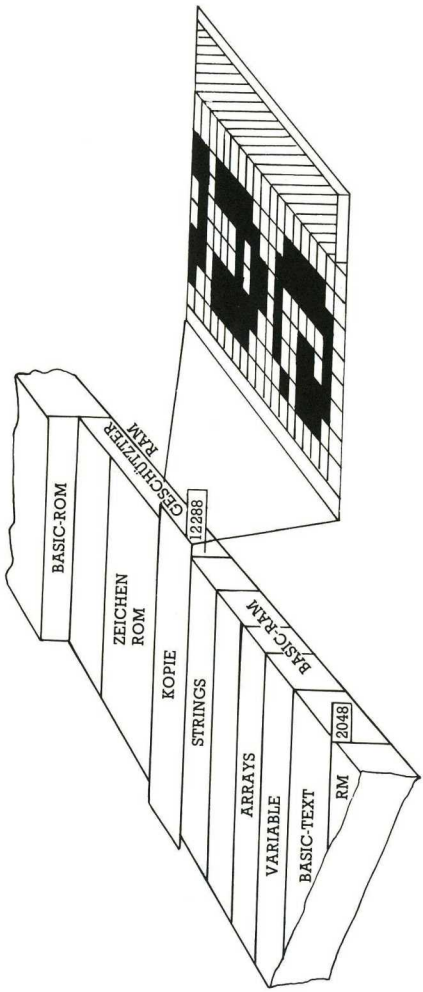


Bild 2.10 Der kopierte Zeichensatz

Nach RUN merken Sie - wenn alles richtig war - noch keinen Unterschied, außer, daß wir eine Menge Speicherplatz verbraucht haben. Aber nun wollen wir ans Zeichenumbauen gehen. Nehmen wir an, daß wir den Buchstaben A zu langweilig finden. Wir werden ihm ein neues Image verleihen. Wenn Sie sich an Kapitel 1 erinnern, dann ist der Buchstabe A nach dem Klammeraffen (@) der zweite Buchstabe. Sein erstes Byte ist an achter Stelle des Zeichen-ROMs, also ab 53256, zu finden. Nachdem wir kopiert haben, finden wir ihn ab 12296, und jetzt verstehen wir auch das Bild 1.8 in Kapitel 1 als gesetzte und gelöschte Bits anzusehen.

Byte	binär		dezimal
12296	00011000	=	24
12297	00111100	=	60
12298	01100110	=	102
12299	01111110	=	126
12300	01100110	=	102
12301	01100110	=	102
12302	01100110	=	102
12303	00000000	=	0

Bild 2.11 Das Zeichen A jetzt im RAM

Wir zeichnen uns ein 8 x 8-Raster und konstruieren darin unser neues "A".

Byte	binär		dezimal
12296	01100110	=	102
12297	00000000	=	0
12298	00011000	=	24
12299	00011000	=	24
12300	10000001	=	129
12301	01100110	=	102
12302	00111100	=	60
12303	00000000	=	0

Bild 2.12 So soll unser neues "A" aussehen

Dann berechnen wir die Dezimalwerte der Bytes und geben schließlich ein:

```
80 POKE 12296,102:POKE 12297,0:POKE 12298,24:POKE 12299,24
:POKE12300,129
90 POKE 12301,102:POKE 12302,60:POKE 12303,0
```

Wenn Sie dieses Programm mit RUN starten, lächelt Sie künftig das A freundlicher an als bisher. Natürlich läßt sich das alles auch viel eleganter lösen. Besonders die Zeilen 80 und 90 können durch eine kleine Schleife, die DATA-Zeilen liest und in den Speicher POKEd, ersetzt werden. Die Adressen werden durch Multiplikation des Commodore-Codes mit 8 berechnet:

Startadresse des Zeichens mit Code C: 12288 + 8*C.

Schließlich noch eine Bemerkung: Obwohl manche Basic-Befehle nach der Änderung etwas vergnügter aussehen als vorher, funktioniert zum Beispiel TAB(5) mit dem 'neuen' A genausogut wie das 'alte' TAB(5). Testen Sie mal:

```
PRINT TAB(5)"ABRAKADABRA".
```

Wenn Sie auf Kleinschreibung oder REVERSE umschalten, sieht alles wieder normal aus.

Hier noch die Lösungen der Aufgaben:

a) 11001, 10000, 101111, 10000000
b) 17, 14, 240
c) 146, 55080, 40961
d) $FFFE, $800, $52F3
e) POKE 770, 243:POKE 771,82

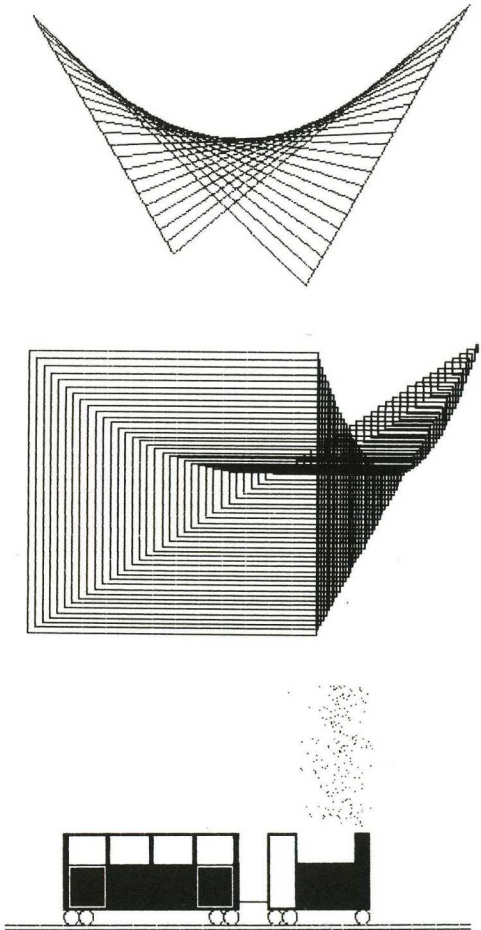


Bild 2.13 Bilder aus dem Testprogramm zur Grafik-Erweiterung HIRES-3

3

Die ersten Schritte

3 Die ersten Schritte

3.1 Für Farbenkünstler: Der Mehrfarben-Zeichen-Modus

Erinnern Sie sich bitte an Bild 2.11 oder an Bild 1.8. Egal, ob es sich um Zeichen aus dem Zeichen-ROM oder um selbstdefinierte Zeichen handelt, eines haben sie gemeinsam: Überall dort, wo sich im Bit-Muster des Zeichens eine 1 befindet, also ein Bit gesetzt ist, taucht bei der Darstellung auf dem Bildschirm ein Punkt in der Zeichenfarbe auf, die im Bildschirmfarbspeicher angegeben ist. Diese Farbe haben wir entweder im Direktmodus durch Drücken der ‹CTRL›- oder ‹COMMODORE›-Taste in Kombination mit einer Zifferntaste oder durch ein "POKE 646,Zeichenfarbe" oder auch durch POKEn eines Farbcodes in die entsprechende Stelle des Bildschirmfarbspeichers festgelegt.

Überall dort, wo im Bitmuster des Zeichens eine 0 ist, taucht bei der Darstellung auf dem Bildschirm nichts auf, das heißt, dort erscheint die Farbe des Bildschirmhintergrunds. Jedes Bit aus dem Bit-Muster wird im Normalbetrieb und auch bei selbstdefinierten Zeichen einzeln interpretiert. Es gibt auch andere Möglichkeiten der Interpretation des Bit-Musters. Eine davon wird im Mehrfarbenmodus angewendet. Hier sind die Bits des Zeichens zu Paaren zusammengefaßt.

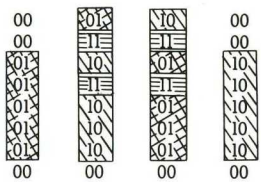


Bild 3.1 Der Buchstabe A, dargestellt im Mehrfarbenmodus

Das ist der Buchstabe "A" mit paarweise zusammengefaßten Bits. Es ergeben sich vier verschiedene Möglichkeiten der Bit-Kombination:

00, 01, 10, 11.

Erinnern Sie sich an die Binärzahlen, erkennen Sie, daß diese vier Möglichkeiten den Zahlen 0, 1, 2, 3 entsprechen. Sie wirken wie Zeiger, die in verschiedene Register des VIC-II-Chip deuten (Tabelle 3.1; siehe auch Registerübersicht in Tabelle 1.1).

		Bit-Paar zeigt auf:	
Bit-Paar	Register #	Adresse	Bedeutung
00	33	53281	Hintergrundfarbe Nr. 0
01	34	53282	Hintergrundfarbe Nr. 1
10	35	53283	Hintergrundfarbe Nr. 2
11 bringt die Farbe, die in Bit 0 bis 2 des Bildschirmfarbspeichers vorhanden ist			

Tabelle 3.1 Bedeutung von Bit-Kombinationen im Mehrfarbenmodus

Dort also, wo ein Bitpaar 00 steht, wird es in der Farbe des Bildschirmhintergrundes angezeigt, wie bisher im Normalmodus mit einem Bit 0 geschehen. In Adresse 53281 haben wir bisher immer die Bildschirmfarbe festgelegt. Hinzu kommen die Register 34 und 35 des VIC-II-Chips entsprechend den Adressen 53282 und 53283. Hier können wir nun weitere Farben eingeben, die an den Stellen auf dem Bildschirm gezeigt werden, an denen die dazugehörende Bit-Kombination (01 oder 10) steht. Erlaubt sind alle Farben (0 bis 15).

Nun sind es noch zwei Voraussetzungen, die uns vom Mehrfarben-Zeichen trennen: Dem Computer muß mitgeteilt werden, daß er das Bit-Muster in Paaren statt einzeln interpretieren soll. Das geschieht dadurch, daß man im Register 22 (Adresse 53270) das Bit 4 auf 1 setzt. Falls Sie sich nicht mehr so genau erinnern, wie wir so etwas bisher gemacht haben - hier ist es nochmal gezeigt.

XXX0XXXX (53270), Bit 4 = 0
OR
00010000 Maske, dez. 16

= **XXX1XXXX** (53270), Bit 4 = 1

Bild 3.2 Das Setzen einzelner Datenbits mittels logischer OR-Verknüpfung

Es muß eingegeben werden:

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

Dieses Byte 53270 im VIC-II-Chip ist ein Beispiel dafür, wie sorgfältig man auf das richtige Setzen oder Löschen von Bits achten sollte. Wenn Sie sich die Registerübersicht in Tabelle 1.1 und dort besonders dieses Byte ansehen, stellen Sie fest, daß fast jedes Bit eine andere Funktion zu erfüllen hat.

Sollten Sie die obige Basic-Zeile im Direktmodus eingegeben haben, verwandeln Sie alle Buchstaben auf dem Bildschirm in mehrfarbige Zeichen. Auf einem Schwarzweiß-Gerät sind sie dann kaum mehr zu erkennen. Hier stellt man auch den Unterschied zwischen einem Farbfernseher und einem Farbmonitor fest. Der letztere trennt die unterschiedlichen Farben deutlich voneinander, während auf dem Fernsehschirm häufig Farbmischungen autreten. Deswegen sollte die Farbgebung in den Register 34 und 35 individuell ausfallen: Probieren, probieren...

Das Ausschalten des Mehrfarb-Zeichen-Modus geschieht durch Löschen des Bit 4 in 53270 mit

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

Falls Sie vorhin beim Anwählen des Mehrfarb-Modus nicht die normale Zeichenfarbe (Code 14), sondern eine zwischen 0 und 7 vorliegen hatten, haben Sie keine Veränderung bemerkt.

Und damit sind wir bei der zweiten Voraussetzung: Es kommt nämlich auch noch auf den Inhalt jeder Bildschirmfarbspeicherstelle an (55296 bis 56295). Erst wenn für eine solche Zelle das Bit 3 gesetzt ist (= 1), findet man Zeichen im dazugehörigen Bildschirmplatz in der Mehrfarbendarstellung. Sehen wir uns dazu noch einige Farbcodes an:

1 = bin. 0001

7 = bin. 0111

Bit 3 ist gelöscht

8 = bin. 1000

15 = bin. 1111

Bit 3 ist gesetzt

Der Farbcode bewirkt zweierlei:

a) Er gibt an, ob ein Zeichen im Mehrfarbmodus dargestellt wird,

   und

b) er verleiht den Bit-Paaren "11" die Farbe.

Zur Demonstration können Sie folgendes Programm eingeben:

```
10 PRINT CHR$(147)
20 POKE 53281,6:POKE 53282,1:POKE 53283,0:REM LADEN DER 3
FARBREGISTER
30 POKE 53270,PEEK(53270)OR 16:REM ANSCHALTEN DES MEHRFARBENMODUS
40 POKE 646,0:REM ZEICHENFARBE AUF 0(BLK)
50 PRINT CHR$(17)"JETZT IST DER MEHRFARBENMODUS":PRINT"EINGESCHALTET"
60 PRINT CHR$(17)"IN DIE BILDSCHIRMFARBZEILEN IST ABER DIE";:PRINT"0
EINGEGEBEN
70 PRINT CHR$(17)"DESWEGEN IST DIE ZEICHENDARSTELLUNG":PRINT"NORMAL"
80 POKE 646,14 :REM ZEICHENFARBE AUF 14(HBLU)
90 PRINT CHR$(17)"DER FARBCODE IST JETZT 14"
100 PRINT CHR$(17)"DER MEHRFARBEN-MODUS IST SICHTBAR"
110 END
120 POKE53270,PEEK(53270) AND 239:REM AUSSCHALTEN DES MEHRFARBEN-
MODUS
130 END
```

Wenn dieses Programm mit Zeile 110 beendet ist, befinden Sie sich weiterhin im Mehrfarbenmodus und können nach Herzenslust durch POKE 646,"Zeichenfarbe", oder mit <CTRL>-, beziehungsweise <COMMODORE>-Taste und einer Zifferntaste die Zeichenfarbe ändern und den Effekt beobachten. Übrigens ist der Cursor unterhalb der READY-Meldung noch da - was Sie durch eine Änderung der Zeichenfarbe sofort feststellen können. Sollten Sie genug von diesem Modus haben, geben Sie CONT <RETURN> ein und das Programm schaltet wieder den Normalmodus ein.

Sollten Sie zufällig mal durch

```
PRINT PEEK(53281),PEEK(53282),PEEK(53283)
```

in die Hintergrundfarbregister sehen, dann werden Sie dort nicht - wie im Programm veranlaßt - die Ziffern 6, 1, 0 sehen, sondern 246, 241, 240. Das liegt daran, daß die Bits 4 bis 7 dieser Register für die Farbgebung nicht benutzt werden und vom Betriebssystem auf 1 gehalten werden. Die Binärzahl 1111 0000 entspricht dem Dezimalwert 240. Deswegen muß man zur Farbcodezahl 240 addieren, um den Speicherinhalt zu erhalten.

Das Abschalten dieses Modus geschieht durch Löschen des gleichen Bits:

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

Die Zeichenfarbe bleibt von der ganzen Umstellung unberührt. Man kann sie auf die nun schon bekannte Weise jederzeit ändern.

Warum kann man eigentlich in diesem Modus nur 64 Zeichen darstellen? Die Antwort darauf lautet, daß nur die Bits 0 bis 5 für den Zeichencode zur Verfügung stehen, also

von 00 0000 = "@"
bis 11 1111 = "?"

mit Hintergrundfarbe aus 53281.

Jede weitere Erhöhung beeinflußt die Bits 7 und 6, wird aber in den unteren 6 Bits nicht mehr wahrgenommen. Dort beginnt einfach die Zählung wieder ab 0:

(1)00 0000 = 64,

das bedeutet Zeichen 0 (@) mit Hintergrundfarbe aus 53282.

Ganz nett wäre es ja eigentlich, wenn man sowohl den Modus mit erweiterten Hintergrundfarben als auch den Mehrfarbzeichenmodus miteinander kombinieren könnte. Jeder Versuch, das zu tun, endet mit einem schwarzen Bildschirm ohne jegliches Zeichen. An diese Möglichkeit haben die Schöpfer des VIC-II-Chip offenbar nicht gedacht.

Betrachten Sie jetzt das nachfolgende Demonstrationsprogramm. Vor der Eingabe schalten Sie bitte durch gleichzeitiges Drücken der <COMMODORE>- und <SHIFT>-Tasten den Kleinschriftmodus ein.

```
5 print chr$(147)
10 poke 53281,6:poke 53282,0:poke 53283,7:poke 53284,10:rem laden der
farbregister
20 poke 53265,peek(53265) or 64:rem anschalten des modus
30 print chr$(17)"der modus ist eingeschaltet aber der"
35 print"zeichencode ist kleiner als 64"
40 print chr$(17)"JETZT WIRD MIT GESHIFTETEN ZEICHEN":print"
GEDRUCKT"
50 print chr$(18)"der druck mit reversed zeichen hat die"
60 print chr$(18)"hintergrundfarbe von register 53283"
70 print chr$(144)chr$(17)chr$(18)"REVERSED UND GESHIFTETE
ZEICHEN"
80 print chr$(154):end
90 poke 53265,peek(53265)and 191
100 end
```

Ebenso wie beim vorigen Programm können Sie nach dem ersten END in Zeile 80 noch mit den verschiedenen Kombinationen von Zeichenfarbe und Zeichencode herumexperimentieren. Wenn Sie genug davon haben, erreichen Sie mit CONT ‹RETURN› die Rückkehr in den Normalzustand.

Auch dieser Modus ist mit selbstdefinierten Zeichen ebenso verwendbar wie mit den ROM-Zeichen.

3.3 Wir betreten Dornröschens Schloß: Das Bit-Mapping-Prinzip

Es ist soweit! Wir stehen an der Schwelle zur hochauflösenden Grafik. Wenn wir bisher mit Zeichen aller Art auf dem Bildschirm gearbeitet haben, waren diese immer durch acht Bytes definiert: entweder durch das Zeichen-ROM oder durch einen von uns erstellten alternativen Zeichensatz im RAM. Jeder Aufruf eines Zeichencodes füllte einen Bildschirmspeicherplatz, zum Beispiel 1024 mit diesem Zeichen, also mit 8 Byte. Zugriff auf einzelne Bytes hatten wir nur beim Selbstdefinieren. Dieser Zugriff legte dann eben in der Anwendung auch wieder 8 Byte auf einmal fest. Wenn es nun eine Möglichkeit gäbe, ständig Zugriff auf einzele Bytes zu haben und die Punktmuster jedes Bytes verändern zu können, hätten wir ganz andere Aussichten.

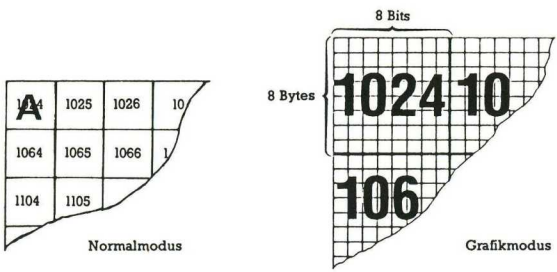


Bild 3.3 Normal- und Grafikmodus

Überlegen wir mal: 1000 Byte (40 x 25) hätten wir dann 8000 Byte (40 x 25 x 8) zu je 8 Bit, also insgesamt 64000 Bildpunkte, die ansprechbar wären! Genau das geschieht beim sogenannten Bit-Mapping. Der Computer legt sich gewissermaßen eine Landkarte dieser 8000 Byte an und bildet ihren Inhalt auf dem Bildschirm ab. Auf dem Bildschirm können wir mit dieser Karte (siehe Bild 3.3) in der Senkrechten (y) 25 x 8 = 200 Positionen und in der Waagerechten (x) 40 x 8 = 320 Positionen ansprechen. Immer dann, wenn ein Bit in der Bit-Map gesetzt (= 1) ist, erzeugt das automatisch das Aufleuchten eines der 64000 Bildpunkte auf dem Bildschirm. Ist ein Bit gelöscht (= 0), ist der Bildpunkt leer.

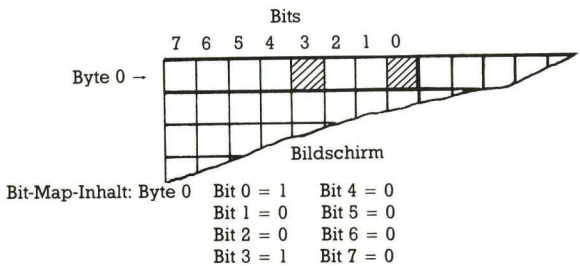


Bild 3.4 Bit-Mapping

Das Einschalten dieses Bit-Map-Modus ist der Kuß, den wir Dornröschen geben. Seien Sie nicht vorzeitig enttäuscht. Dornröschen hat 1000 Jahre geschlafen und wir müssen mit viel Geduld ihre Fähigkeiten nach und nach entwickeln helfen. Wenn Dornröschen jetzt erwacht, ist es hilflos wie ein neugeborenes Kind. Nach dieser Warnung gehen wir so vor: Im VIC-II-Chip Register 53265 setzen wir Bit 5 auf 1, indem wir den Befehl

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

eingeben. Danach sieht der Bildschirm allerdings recht merkwürdig aus. Das ist auch kein Wunder, denn wie gesagt wird jetzt der Inhalt der Bit-Map dargestellt, und wir haben dem Computer noch nicht mitgeteilt, wo die Bit-Map zu finden ist. Das geschieht durch Setzen des Bit 3 des VIC-II-Registers mit der Adresse 53272 auf den Wert 0 oder 1. Dabei gelten die Zusammenhänge nach Tabelle 3.3.

Bit 3 von 53272	Ort der Bit-Map (Adressenbereich)
0	0 - 7999
1	8192 - 16191

Tabelle 3.3 Mögliche Bereiche des Bit-Map

Der normale Inhalt des Bytes 53272 ist 21 (das kennen wir noch aus Kapitel 2). Binär ausgedrückt ist das: 0001 0101.

Bit 3 ist also 0. Alles, was in diesem Bit-Map-Speicherbereich von 0 (Zeropage!) bis 7999 (mitten im Basic-RAM) existiert, wurde beim Einschalten des Bit-Map-Modus abgebildet. Weil die Zeropage laufend vom Betriebssystem für seine ständigen Arbeiten verwendet wird, sind im oberen Bereich unseres seltsamen Bildes auch flimmernde Stellen zu sehen - wo laufend Bits gesetzt und gelöscht werden. Sehen Sie sich Ihren Bildschirm mit diesem merkwürdigen Abbild noch etwas genauer an. Etwa von der Mitte an sehen Sie Zeichen abgebildet. Dort liegt die Speicherzelle 4096, von der an das Geisterbild der Zeichen (siehe Kapitel 1) liegt. Wir sehen Gespenster!

Nun sollten Sie mit ‹RUN/STOP› und ‹RESTORE› wieder den Normalzustand herstellen. Dann schalten Sie mit folgenden Programmen Hochauflösungsmodus und die spezielle Bit-Map ab 8192 an:

```
10 PRINT CHR$(147):POKE53265,PEEK(53265)OR 32
20 POKE 53272,PEEK(53272)OR 8
```

Der letzte Befehl besorgt das Setzen des Bit 3 auf 1. Außerdem fügen wir vorsorglich noch die Rückkehr in den Normalmodus an:

```
110 POKE 53265,PEEK(53265)AND 223          Löschen des Bit 5 in 53265.

120 POKE 53272,PEEK(53272)AND 247          Löschen des Bit 3 in 53272.
```

Dazu bauen wir noch eine GET-Abfrage ein:

```
100 GET A$:IF A$ = "" THEN 100.
```

Starten Sie jetzt mit RUN. Die Bit-Map ab 8192 wird auf dem Bildschirm gezeigt. Das ist leerer Speicher, so wie wir ihn in Form von Zahlen (Blöcken von 0 und 255) schon in Kapitel 1 kennengelernt haben.

Sie erinnern sich vielleicht auch daran, daß manchmal, wenn auch selten, andere Zahlen eingestreut waren. Deswegen sieht man hier auch nicht immer ganz schwarze oder ganz helle Blöcke, sondern auch häufig gesprenkelte. Überall dort, wo ein Byte der Bitmap den Wert 255 hat, steht binär 1111 1111, das heißt 8 Bildpunkte auf dem Bildschirm sind eingeschaltet. Wir sehen schon, wir müssen den Speicher noch löschen, also überall 0 hinschreiben:

```
30 B = 8192:FOR I = 0 TO 7999:POKE I+B,0:NEXT I
```

Wenn Sie diese Programmzeile noch einfügen und dann mit RUN starten, können Sie dem Computer bei der Arbeit zusehen.

3.4 Jetzt kommt Farbe ins Bild

Vermutlich sind Sie jetzt enttäuscht. Ein leerer schwarzer Bildschirm ist nicht gerade eindrucksvoll. Deswegen wollen wir nun noch zwei Dinge lernen: Wie kommt Farbe ins Bild und wie bekomme ich Punkte auf den Bildschirm?

Wenn Sie vor der Zeile 100 im Programm einen Stop-Befehl einfügen, tauchen nach der Abarbeitung der ersten Zeilen plötzlich farbige Quadrate auf dem Bildschirm auf. Die Antwort auf die Frage nach der Herkunft dieser Quadrate birgt für uns zugleich die Möglichkeit der Farbgebung. Geben Sie nun CONT ‹RETURN› ein. Auch das erscheint als Folge farbiger Quadrate. Jetzt wartet der Computer auf einen Tastendruck, der auf dem Bildschirm folgendes erscheinen läßt:

```
BREAK IN 40
READY
CONT

READY
```

Genau an derselben Stelle waren im HIRES-Modus die farbigen Quadrate zu sehen. Die Farbe wird also anscheinend vom Inhalt der Bildschirmspeicherzellen gesteuert. Das B von BREAK hat einen Bildschirmcode von 2. Zwei ist auch der Farbcode für Rot. Falls Sie einen Farbmonitor Ihr eigen nennen, sehen Sie im Hochauflösungsmodus an der Stelle des B ein rotes Quadrat. Des Rätsels Lösung sieht so aus: Die Farbe des Hochauflösungsbildes wird von 8x8-Bit-Feldern gesteuert, die den Bildschirmspeicherzellen entsprechen. Die Steuerung ist vom Inhalt des Bildschirmspeichers abhängig, nicht vom Inhalt des Farbspeichers. Schreiben wir zum Beispiel in Zelle 1024 eine 1,

erhalten wir im Hochauflösungsmodus ein weißes Feld in der linken oberen Bildschirmecke. Nun wollen wir in diesen Bereich auch noch eine kleine Linie zeichnen: Löschen Sie bitte zunächst mal den STOP-Befehl und fügen Sie ein:

```
40 POKE 1024,1:POKE 8193,255
```

Nach dem Starten erhalten wir - wie erwartet - das weiße Feld links oben und darin einen schwarzen Strich. Um offen zu sein - ganz so erwartet war es ja nicht. Denn woher hätten wir wissen können, daß der Strich schwarz ist? Sehen wir uns den Inhalt der Speicherzelle 1024 genauer an. Im binären System geschrieben steht dort jetzt:

0000	0001
MSN	LSN
= 0	= 1

mit N = Nibble (4 Bit), also MSN = Most Significant Nibble, LSN = Least Significant Nibble.

Das legt die Vermutung nahe, daß die Hintergrundfarbe durch das LSN (hier 1 = weiß) und die Punktfarbe durch das MSN (hier 0 = schwarz) gesteuert wird. Probieren wir es einfach mal aus. Wir tauschen das ganze um und schreiben in 1024 den folgenden Wert:

0001	0000
MSN	LSN
= 1	= 0
(dezimal 16)	

Außerdem ändern wir in der Zeile 40 den ersten POKE-Befehl in POKE 1024,16. Es klappt! Nach dem Starten (wie immer in diesem Programm nach einigem geduldigen Warten) erscheint ein weißer Strich auf schwarzem Grund. Nun können wir dieses triste Schwarz gezielt entfernen und von vornherein bestimmen, welche Farbe unsere Punkte bekommen sollen und welche der Hintergrund, indem wir in die Bildschirmzellen den Dezimalwert unserer Binärzahl eingeben. Damit Sie nicht jedesmal umrechnen müssen, hier eine nützliche Formel:

Farbkennziffer (F) = Zeichenfarbe (ZF) x 16 + Hintergrundfarbe (HF).

Zum Ausprobieren ändern wir unser Programm:

```
5 INPUT"ZF,HF =";ZF,HF
```

3.5 Dornröschen lernt zeichnen: Punkte setzen im Hochauflösungs-Modus

Wir sind es jetzt leid, immer einen fast leeren Bildschirm anzusehen. Ob er nun schwarz oder blau ist: Wir wollen nun auch mal etwas darauf sehen! Die Schwierigkeit liegt gar nicht darin, daß wir nicht wüßten, wie wir Punkte auf den Bildschirm kriegen. Wir müssen nur ein Byte im Bereich von 8192 bis 16191 mit ein paar gesetzten Bits versehen, zum Beispiel durch POKE 12000,1, und schon sehen wir einen Punkt. Das Problem liegt vielmehr darin, daß wir einen Weg finden, gezielt Punkte an bestimmte Orte des Bildschirms zu setzen. Dazu müssen wir den Aufbau der Bit-Map kennen und ihren Zusammenhang mit dem Bildschirm. Das Prinzip ist in Bild 3.5 gezeigt.

Die Speicherzellen der Bit-Map sind angeordnet wie ein vollgeschriebener Bildschirm. Dort hatte jeder Buchstabe seine 8 Byte im Punktemuster. Aus der Anordnung ergibt sich außerdem, daß wir in der Y-Richtung 200 mögliche Positionen (0 bis 199) und in der X-Richtung 320 (0 bis 319) finden. Wir haben somit Bildschirmkoordinaten (Bild 3.6) und können Punkte definieren (zum Beispiel im Bild den Punkt P mit den Koordinaten X = 110 und Y =100).

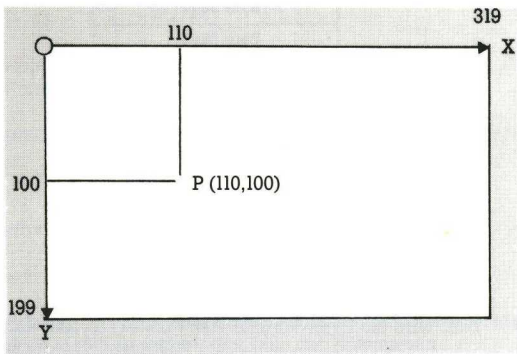


Bild 3.6 Bildschirmkoordinaten bei hochauflösender Grafik

Wir wissen nur noch nicht, in welchem Byte wir welches Bit setzen müssen, um diesen Punkt zu sehen. Betrachten wir dazu Bild 3.5 genauer. Die 25 Bildschirmzeilen und die 40 Spalten sind noch erhalten. Jeder solche Bildschirmplatz besteht aus acht Byte mit je acht Bit. Stellen wir zunächst einmal fest, um welche Zeile es sich handelt:

```
Z = INT(Y/8)
```

Beispiel: Z = INT(100/8)=12 (Zeile 12)

Dann ermitteln wir die Spalte, in der unser Punkt liegt: S=INT(X/8)

Beispiel: S = INT(110/8)=13 (Spalte 13)

In unserem Beispiel muß der Punkt in Zeile 12 an Spaltenplatz 13 stehen. Weil in jeder Zeile 320 Byte (8x40) und an jedem Platz 8 Byte vorhanden sind, fängt unser Bildschirmfeld bei Byte Nummer

```
Z * 320 + S * 8 = 12 * 320 + 13 * 8 = 3944
```

an (siehe Bild 3.7).

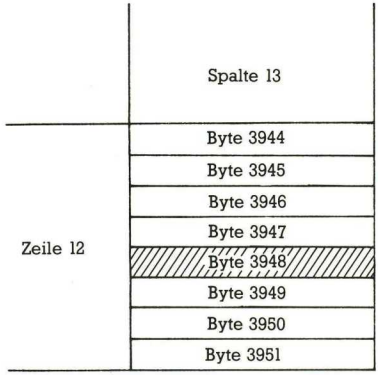


Bild 3.7 Hier wird die Byte-Adresse zum Punkt (110,100) ermittelt

Nun müssen wir feststellen, um welches Byte in diesem Feld es sich handelt. Wir haben vorhin beim Feststellen der Zeile einfach gleich INT(Y/8) berechnet. Wenn wir nun nur Y/8 berechnen, kommen wir in unserem Beispiel auf Y/8=100/8=12.5. Ziehen wir davon INT(Y/8) ab: Y/8-INT(Y/8)=12.5-12=0.5 und multiplizieren das Ergebnis mit 8:

```
80 * (Y/8 - INT(Y/8)) = 8 * 0.5 = 4
```

Genau das ist unser gesuchtes Byte: Nr. 4 von oben. Das sieht komplizierter aus, als es ist. Aus alledem ergibt sich jetzt, in welchem Byte der Bit-Map ein Bit zu setzen ist:

```
BYTE = Z * 320 + S * 8 + 8 * (Y/8 - INT(Y/8)).
```

In unserem Beispiel ist es das Byte 3948 der Bit-Map. Nun müssen wir noch herauskriegen, welches Bit in diesem Byte zu setzen ist.

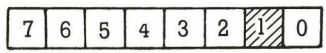


Bild 3.8 Die Bit-Adresse

Auch hier haben wir vorhin bei der Berechnung der Spalte einfach

```
INT(X/8) = 110 - 8 * INT(110/8)
```

gerechnet. Das ergibt 6. Wenn wir das von 7 abziehen (Nummer des höchstwertigen Bit), erhalten wir:

```
BIT = 7 - (X - 8 * INT(X/8))
```

In unserem Beispiel ist BIT = 1. Es muß also Bit 1 gesetzt werden. Damit haben wir jetzt folgende Formeln zur Verfügung:

```
BYTE = INT(Y/8) * 320 + INT(X/8) * 8 + 8 * (Y/8 - INT(Y/8))
```

und

```
BIT = 7 - (X - 8 * INT(X/8))
```

Den ersten Ausdruck für Byte kann man noch etwas vereinfachen:

```
BYTE = 8 * (39 * INT(Y/8) + INT(X/8) + Y/8)
```

Die genaue Speicherzelle ergibt sich durch Addieren der Bit-Map-Startadresse (8192) zu BYTE. Um einen Punkt zu setzen, gibt man das Kommando:

```
POKE 8192+BYTE,PEEK(8192+BYTE) OR(2^BIT)
```

Wir ergänzen jetzt unser Programm. Die Zeile 50 wird neu:

```
50 BY=8*(39*INT(Y/8)+INT(X/8)+Y/8):BI=7-(X-8*INT(X/8))
```

Dann folgt: `60 POKE B + BY,PEEK(B+BY) OR (2^BI)`

Außerdem müssen wir natürlich Punktkoordinaten eingeben: `6 INPUT"X,Y=";X,Y`

Dabei ist darauf zu achten, daß X zwischen 0 und 319, Y zwischen 0 und 199 liegt. Starten Sie mit RUN, geben Sie die Koordinaten ein und nach einer Weile wird Ihr Punkt auf dem Bildschirm zu sehen sein.

Auf eines sollten Sie noch achten, wenn Sie die Bit-Map bei 8192 beginnen lassen (im nächsten Kapitel werden wir andere Möglichkeiten kennenlernen): Schützen Sie die Bit-Map vor dem Überschreiben durch Basic! Der verbliebene Speicherplatz für Basic-Text, Variable, Arrays und Strings ist so knapp, daß Sie ganz schnell die Bit-Map überschreiben. Deswegen sollten Sie (siehe Kapitel 2) eingeben:

```
POKE 51,255:POKE 52,31:POKE 55,255:POKE 56,31.
```

Alles, was wir bis jetzt über die Hochauflösungsgrafik gelernt haben, soll hier nochmal als kleines Demonstrationsprogramm zusammengefaßt werden. Also NEW eingeben und dann:

```
10 POKE 51,255:POKE 52,31:POKE 55,255:POKE 56,31
20 ZF=0:HF=6:F=16*ZF+HF
30 DEFFNA(X)=50*SIN(X/30)+100
40 PRINT CHR$(147)
50 POKE 53265,PEEK(53265)OR 32
60 POKE 53272,PEEK(53272)OR 8
70 B=8192:FOR I=0 TO 7999:POKE B+I,0:NEXT I
80 FOR I=1024 TO 2023:POKE I,F:NEXT I
90 FOR X=0TO 319:Y=FNA(X)
100 BY=(XAND 504)+40*(Y AND 248)+(Y AND 7):BI=7-(X AND 7)
110 POKE B+BY,PEEK(B+BY)OR(2^BI):NEXT X
120 GET A$:IF A$=""THEN 120
130 POKE 53272,PEEK(53272)AND(255-8)
140 POKE 53265,PEEK(53265)AND(255-32)
150 PRINT CHR$(147)
```

Dieses Programm setzt 320 Punkte nach der Form einer Sinus-Funktion. Zur Erläuterung sei noch bemerkt, daß in Zeile 100 eine etwas elegantere Lösung der Berechnung von BYTE und BIT angeführt ist. Sie können aber genauso gut die bisher verwendete benutzen. In Zeile 150 wird der Bildschirm nochmal gelöscht, denn bei der Eingabe der Farben haben wir ja die Bildschirmzellen mit einem Zeichencode belegt, der im Normalmodus immer unter dem Cursor sichtbar wäre.

Dornröschen ist also erwacht und hat die ersten zaghaften Schritte getan. Alles geht noch etwas langsam vor sich. Besonders das Löschen der Bit-Map (Programmzeile 70) erfordert geduldiges Abwarten - jedenfalls solange wir in Basic arbeiten.

4

In Riesenschritten zum Ziel: Die hochauflösende Grafik in BASIC-Programmen

4 In Riesenschritten zum Ziel: Die hochauflösende Grafik in BASIC-Programmen

4.1 Wir krempeln den Commodore 64 um: Speicherveränderungen für hochauflösende Grafik

Wenn Sie als stolzer Besitzer eines C 64 früher ein ebenso stolzer Besitzer eines VC 20 waren, ist Ihnen sicherlich in wehmütiger Erinnerung, was Sie sehen, wenn Sie durch die POKE-Kommandos

```
POKE 51,255:POKE 52,31:POKE 55,255:POKE 56,31
```

im letzten Kapitel die Bit-Map vor dem Überschreiben durch Basic geschützt haben und dann mit PRINT FRE(1) nach dem freien Basic-Speicher fragten: Da zeigt sich: 6144 (ohne Programm)!

Geht das Ringen um jedes Byte nun wieder los? Wie soll denn in diese 6 KByte ein besseres Spiel mit hochauflösender Grafik - ganz zu schweigen von anspruchsvolleren Programmen, zum Beispiel einer Kurvendiskussion - hineinpassen? Nun, keine Sorge: Wozu haben wir im C 64 denn 64 KByte RAM? Wir müßten nur wissen, wie wir sie nutzen können.

Dazu sehen wir uns nochmal den VIC-II-Chip an. Im Gegensatz zur CPU (unserem Prozessor 6510), die über 16 Adreßleitungen verfügt, stehen beim VIC-II-Chip lediglich 14 zur Disposition. Während man mit 16 Leitungen alle Adressen

von 0 bis 1111 1111 1111 1111 = 65535 Adressen

ansteuern kann, ist bei 14 Leitungen ein Maximum

von 11 1111 1111 1111 = 16383 Adressen

möglich (16 KByte).

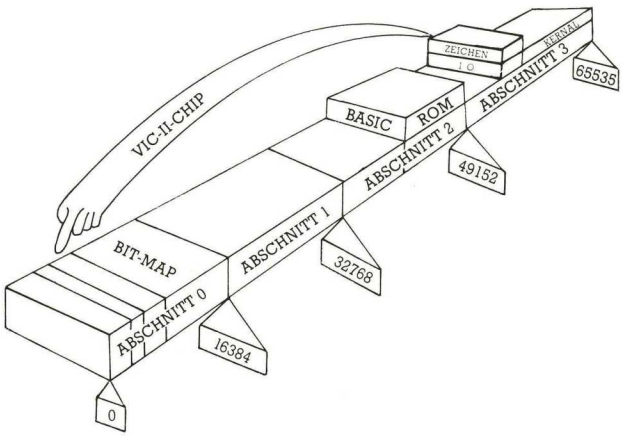


Bild 4.1 Der VIC-II-Chip hat im Normalfall Zugriff auf Abschnitt 0

Der gesamte Speicherraum des C 64 ist in vier solche 16 KByte-Blöcke aufteilbar und wie wir wissen, blickt der VIC-II-Chip im Normalfall auf den ersten 16 KByte-Block (siehe Bild 4.1). Nun kann man dem VIC-II-Chip mitteilen, daß er seine Aufmerksamkeit auf die anderen Speicherviertel richten möge. Das erfordert die Mitarbeit des "Portiers" CIA 2 (siehe Kapitel 1). Er hat im Gebäude 65576 zwei Zimmer (Bits 0 und 1), aus denen dem VIC-II-Chip die Anweisungen gegeben werden, um welches Viertel unseres Speichers er sich kümmern soll. Auf welche Weise diese Bits den VIC-II-Chip-Zugriff regeln, sehen Sie aus der nächsten Tabelle.

56576 BITS 10	I BITWERT DEZIMAL	AB-SCHNITT	SPEICHERBEREICH von — bis	BEMERKUNGEN
11	3	0	0 — 16383	EINSCHALT-ZUSTAND VON 4096 — 8191 ZEICHEN-SPIEGELBILDER
10	2	1	16384—32767	KEINE ZEICHEN VERFÜGBAR
01	1	2	32768—49151	VON 36864 — 40959 ZEICHEN-SPIEGELBILDER VON 40960—49151 BASIC-ROM
00	0	3	49152 — 65535	KEINE ZEICHEN VERFÜGBAR VON 53258-65535 I/O, ZEICHEN-ROM BETRIEBSSYSTEM

Tabelle 4.1 Die Bits 1 und 0 von Speicherstelle 56576 regeln den Zugriff des VIC-II-Chips auf den Speicher

Commodore empfiehlt nun noch sicherzustellen, daß die zu dieser Abschnittsauswahl gehörenden Bits des Datenrichtungsregisters Port A im CIA 2 (Speicherstelle 56578) auf 1, also auf Ausgabe, gestellt werden. In allen Programmen, die ich bisher verwendet habe, wäre das eigentlich nicht nötig gewesen, jedenfalls habe ich nichts bemerkt, als ich das nicht getan habe. Trotzdem gebe ich hier diese Empfehlung weiter, falls in Ihren Programmen diese Maßnahme notwendig wird. Falls Sie vergessen haben sollten, wie man Bits setzt oder löscht, hier die nötigen Programmzeilen dazu:

```
20 POKE 56576,(PEEK(56576) AND 252) OR I
30 POKE 56578,PEEK(56578) OR 3
```

I ist dabei der in Tabelle 4.1 gezeigte Dezimalwert der Bits 0 und 1.

Der VIC-II-Chip managt auch den Bildschirm. Im Einschaltzustand packt er den Bildschirmspeicher - wie wir schon wissen - in den Bereich 1024 und 2023. Wenn wir nun einen anderen 16 KByte-Abschnitt wählen, legt er den Bildschirm an die entsprechende Stelle dieses Abschnittes:

In Abschnitt 0: Bildschirm von 1024 bis 2023

in Abschnitt 1: Bildschirm von 16384 + 1024 = 17408
bis 16384 + 2023 = 18407

in Abschnitt 2: Bildschirm von 32768 + 1024 = 33792
bis 32768 + 2023 = 34791

in Abschnitt 3: Bildschirm von 49152 + 1024 = 50175.
bis 49152 + 2023 = 51175

Damit brauchen wir uns aber nicht zufrieden geben.

Im 16 KByte-Speicherabschnitt hat der Bildschirmspeicher 16mal Platz und wir können ihn ohne weiteres an eine andere Stelle schieben. Damit kehren wir nochmal zur schon oft besungenen Speicherstelle 53272 zurück. Die Bits 4 bis 7 geben dem VIC-II-Chip den Ort des Bildschirmspeichers an. Durch Verändern dieser 4 Bits können wir tatsächlich 16 Bildschirme pro 16 KByte-Abschnitt einrichten. Der Zusammenhang zwischen den Bits 4 bis 7 von Adresse 53272 und dem Ort des Bildschirmspeichers im Abschnitt (und entsprechend parallelverschoben in den anderen Abschnitten) ist in Tabelle 4.2 gezeigt.

SPEICHERSTELLE 53272	DEZIMALWERT (BITS 0-3 als 0 angenommen)	BILDSCHIRMSTARTADRESSE (IM ABSCHNITT 0)	
BITS: 7654	W	DEZIMAL	HEX
0000	0	0	0
0001	16	1024	400
0010	32	2048	800
0011	48	3072	C00
0100	64	4096	1000
0101	80	5120	1400
0110	96	6144	1800
0111	112	7168	1C00
1000	128	8192	2000
1001	144	9216	2400
1010	160	10240	2800
1011	176	11264	2C00
1100	192	12288	3000
1101	208	13312	3400
1110	224	14336	3800
1111	240	15360	3C00

Tabelle 4.2 Zusammenhang zwischen den Bits 4 bis 7 von Speicher 53272 und dem Ort des Bildschirms in Abschnitt 0

Um die entsprechende Bitanordnung zu erreichen, müssen wir also eingeben:

```
40 POKE 53272,(PEEK(53272) AND 15) OR W
```

Dabei ist W der Dezimalwert der Bits 4 bis 7 aus Tabelle 4.2.

Das Betriebssystem muß auch noch erfahren, daß der Bildschirmspeicher verlegt worden ist. Man kann es ihm mitteilen, indem man die Pagenummer der Bildschirmstartadresse in Speicherstelle 648 einPOKEd. Also ist zum Beispiel der normale Inhalt von Speicher 648: 1024/256 = 4. Auf Page 4 beginnt der Bildschirm im Einschaltzustand.

Die Pagenummer ergibt sich aus I und W (siehe Tabellen 4.1 und 4.2) durch folgende Rechnung:

```
50 P = (W/16*1024 + 16348*(3-I))/256
```

und wird dann eingePOKEd:

```
55 POKE 648,P
```

So! Jetzt können wir theoretisch 64 Bildschirme erzeugen. Um uns das in der Praxis mal anzusehen, ergänzen wir die bisher verwendeten Programmzeilen (die Sie hoffentlich noch nicht mit RUN gestartet haben, das wäre nämlich mit etwas Pech eine gute Methode, den Rechner abstürzen zu lassen!) noch um folgende:

```
10 INPUT"I,W";I,W:PRINT CHR$(147)
60 PRINT CHR$(147)I,W:END
65 PRINT CHR$(147)
70 POKE 56576,151:POKE 56578,63:POKE 53272,21:POKE 648,4
80 I=3:W=16:PRINT CHR$(147)I,W
90 END
```

Bevor Sie das starten, sollten Sie bitte daran denken, daß einige I,W-Kombinationen den Computer zum totalen Black-Out führen: Zum Beispiel I = 3, W = 0 legt den Bildschirmstart direkt in die Zeropage und ist deshalb nicht empfehlenswert. I = 3, W = 32 zerstört unser Programm, das genau bei 2048 anfängt. Am besten speichern Sie das Programm vor dem Starten ab.

Jeweils nach RUN und Eingabe der Werte I und W, zum Beispiel I = 3, W = 48, wird der Bildschirm gelöscht und in der obersten Zeile wird der I- und der W-Wert angegeben. Danach meldet sich READY und ein Cursor. Jetzt befinden wir uns im neuen Bildschirm (in unserem Beispiel 3072 bis 4071) und können damit herumexperimentieren.

4.2 Bildschirm-Experimente

Wenn wir jetzt (falls keine Programmänderung in der Zwischenzeit durchgeführt wurde) CONT eingeben, wird der Ursprungszustand (Bildschirm bei 1024) wieder hergestellt und dies durch die Angabe der I- und W-Werte 3 und 16 angezeigt.

Wenn Sie mit diesem Programm in den Bereich 4096 bis 8191 vorstoßen, werden Sie feststellen, daß hier kein normales Bild erscheint. Hier stören die mehrfach schon beschworenen Geisterbilder des Zeichen-ROM, die in diesem Bereich liegen. Es kann so-

gar passieren, daß der Rechner nach der Eingabe von CONT nur noch SYNTAX ERRORs meldet und nicht mehr in den Normalzustand zurückzuführen ist. Ab 8192 bis 15360 (jeweils Start des Bildschirmes) kann man wieder ohne Störung Bildschirme einrichten. Wenn Sie jetzt I = 2 und verschiedene W-Werte versuchen, sehen Sie nur Nonsens oder gar nichts auf dem Bildschirm; dasselbe geschieht bei I = 0.

Das ist wieder eine Besonderheit des VIC-II-Chip. Er ist so strukturiert, daß er (im Normalfall) in diesen beiden Abschnitten keinen Zugang zu den normalen Zeichenspeichern hat. Dafür gibt es in Abschnitt 1 (I = 2) keine Störung durch die Zeichen-Geisterbilder, ebenso in Abschnitt 3 (I = 0). In Abschnitt (I = 1) begegnen wir zwischen 36864 und 40959 wieder den hier ein zweites Mal vorhandenen Zeichen-Gespenstern. Unterhalb von 36864 läßt sich der neue Bildschirm gut verwenden.

Ein Problem stellt sich hier und auch im obersten Abschnitt aber noch auf andere Weise: Wenn wir den Bildschirm zum Beispiel mit I = 1 und W = 128 nach 40960 legen (tun Sie es bitte nicht!), erhalten wir bei jeder Eingabe nur noch "SYNTAX ERRORs" und können den Computer nur durch Aus- und Einschalten wieder zu normaler Tätigkeit bewegen. Was ist da los? Die Erklärung ist, daß von 40960 bis 49151 das Basic-ROM und von 57344 bis 65535 das Betriebssystem dem RAM überlagert sind. Wenn man in diese Regionen hineinPOKEd, landet die Information natürlich im darunterliegenden RAM. Das was auf dem Bildschirm erscheint, wenn wir aus dem Programm-Modus aussteigen, ist allerdings - leider - der Inhalt des darüberliegenden ROM. Ersetzen Sie das "END" in Zeile 60 durch die folgenden Zeilen:

```
60 PRINT CHR$(147)I,W:PRINT"BILDSCHIRM LIEGT UNTER DEM ROM"
65 GET A$:IF A$=""THEN 65
```

Siehe da! Es funktioniert im Programm-Modus (zum Beispiel mit I = 1 und W = 128). Wir könen daher unter das Basic-ROM auf diese Weise acht Bildschirme legen. Unter das Betriebssystem lassen sich so auch Bildschirme legen, nur können wir hier den Text nicht lesen, weil der VIC-II-Chip wie gesagt - hier keinen Zugriff zum Zeichen-ROM hat. So legt zum Beispiel I = 0 und W = 24 den Bildschirm nach 64512, was leicht durch die Zeile nachprüfbar ist :

```
62 POKE 65000,1:POKE 55784,1.
```

Damit wenden wir uns nun dem kritischen Bereich zwischen 53248 und 57343 zu. Hier liegen das Zeichen-ROM und die Ein- und Ausgabebausteine. Normalerweise - wie man auch durch unsere POKE's in diesen Bereich erkennen kann - sind hier die Ein- und Ausgabebausteine eingeschaltet. Wenn wir hierher Bildschirme legen, kann alles mögliche passieren, weil wir Register des VIC-II-Chip, des SID und CIAs beeinflussen. Hier sollte man mit viel Vorsicht und gegebenenfalls nur in Maschinensprache operieren.

Was wir durch die Programmzeile 62 noch erkennen können: Das Bildschirmfarben-RAM verschiebt sich nicht, egal, welches Speicherviertel wir wählen und wohin wir den Bildschirm auch legen: Das Farb-RAM liegt immer von 55296 bis 56295.

Nun aber zum großen Speicherfresser: Zur Bit-Map. Mit ihren 8000 Byte paßt sie im Prinzip achtmal in unseren Computer. Im ersten Viertel (0 bis 16383) haben wir sie schon gehabt und das als unbefriedigend empfunden. Nun wollen wir uns andere Möglichkeiten ansehen und dabei noch bedenken, daß wir auf den normalen Zeichensatz verzichten können (wir stellen ja hochauflösende Grafik dar!). Zu diesem Zweck werden wir das bisher verwendete Bildschirmtestprogramm um einen Hochauflösungsteil erweitern (Programm 1). Um die leidige Eintipperei minimal zu halten, wurde auf Schönheit und erläuternde REM-Zeilen verzichtet. Der Hochauflösungsteil stimmt weitgehend mit dem Programm aus dem letzten Kapitel überein. Laden Sie jetzt das Programm 1 von der Diskette, und probieren Sie es aus.

```
1 REM **************************
2 REM *    PROGRAMM  1          *
3 REM **************************
5 F=6:DEFFNA(X)=50*SIN(X/30)+100
10 PRINTCHR$(147)CHR$(17)"EINGABE DER WERTE"CHR$(17)
20 PRINTCHR$(17)"I(SIEHE TAB.1) ABSCHNITT-KENNZIFFER"CHR$(17)
30 PRINT"W(SIEHE TAB.2) BILDSCHIRM-KENNZIFFER"CHR$(17)
40 PRINT"B=0,BIT-MAP IM UNTEREN ABSCHNITT-BEREICH"
50 PRINT" =1,BIT-MAP IM OBEREN ABSCHNITT-BEREICH"CHR$(17)
60 PRINTCHR$(17):INPUT"I,W,B=";I,W,B:A1=3-I:A2=W/16*1024+A1*16384:A3=A1*16384+B*
8192
70 P=A2/256:PRINTCHR$(17)CHR$(17)"MIT DIESEN EINGABEN HABEN SIE"
80 PRINT"DEN ABSCHNITT "A1" GEWAEHLT."
90 PRINT"IHR BILDSCHIRM STARTET BEI "A2
100 PRINT"UND IHRE BIT-MAP BEI "A3
110 PRINTCHR$(17)"IST DAS SO IN ORDNUNG?(J/N)"
120 GETA$:IFA$=""THEN120
130 IFA$="N"THEN10
140 PRINTCHR$(147)
145 REM **** NEUER SPEICHER ****
150 POKE56576,(PEEK(56576)AND252)ORI
160 POKE56578,PEEK(56578)OR3
170 POKE53272,(PEEK(53272)AND15)ORW
180 POKE648,P
190 POKE53265,PEEK(53265)OR32
200 POKE53272,PEEK(53272)OR(8*B)
210 FORJ=0TO7999:POKEA3+J,0:NEXTJ
220 FORJ=0TO999:POKEA2+J,F:NEXTJ
230 FORX=0TO319:Y=FNA(X)
240 BY=(XAND504)+40*(YAND248)+(YAND7):BI=7-(XAND7)
250 POKEA3+BY,PEEK(A3+BY)OR(2↑BI):NEXTX
260 GETA$:IFA$=""THEN260
265 REM **** ALTER SPEICHER ****
270 POKE53272,21:POKE53265,27:POKE648,4:POKE56578,63:POKE56576,151
280 PRINTCHR$(147):END

READY.
```

Listing 4.1 Testprogramm zur Bildschirmlokalisierung (Programm 1)

In Bild 4.2 sind alle möglichen Positionen der Bit-Map und des Bildschirmes angegeben.

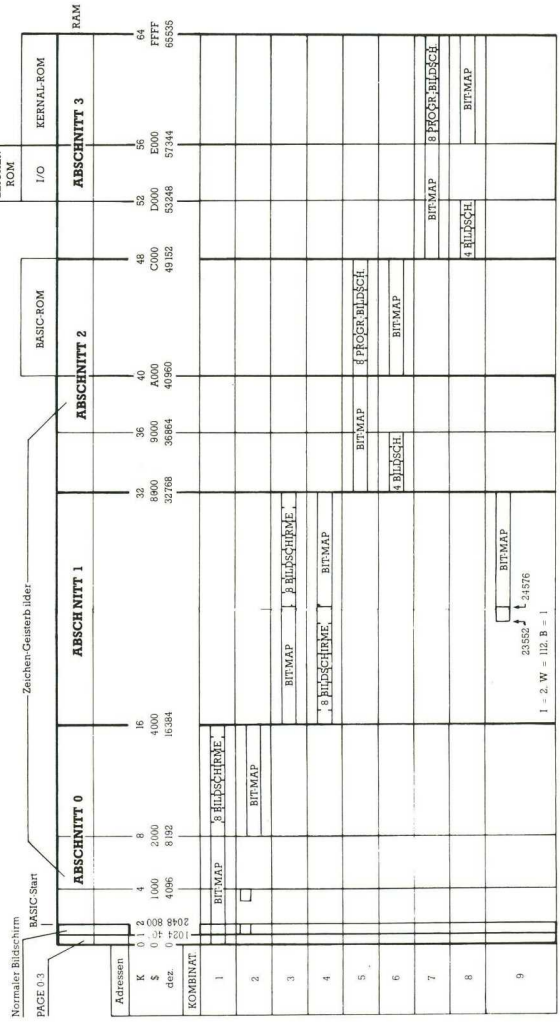


Bild 4.2 Die Speicherbelegung des Commodore 64 und die möglichen Positionen von Bildschirm und Bit-Map

Wie man sehen kann, scheiden die Kombinationen Nummer 1 und Nummer 7 von vornherein aus, weil wir mit dem Löschen der Bit-Map auch das Lebenslicht unseres Rechners ausblasen. Die Kombination Nummer 2 kennen wir schon: So haben wir in der letzten Folge hochauflösende Grafik betrieben und waren enttäuscht über den geringen verbliebenen Basic-Speicher. Bei Nummer 5 funken uns die Zeichen-Spiegelbilder in die Bit-Map, diese Kombination scheidet also auch aus. Nett sieht es aus, wenn wir die Kombinationen Nummer 6 und Nummer 8 testen. Hier machen sich die ROM-Inhalte grafisch zwar ganz interessant aus, aber mit dem Sinn unserer hochauflösenden Grafik hat das nichts mehr zu tun. Für uns brauchbar sind die Positionen im Abschnitt 1: Die Kombinationen aus Nummer 3 und Nummer 4. Ein Maximum an Basic-Speicher findet man bei der letzten gezeigten Kombination Nummer 9, wo ab 23552 der Bildschirm und ab 24576 die Bit-Map liegen. Wenn Sie den Basic-Speicher für diese Anordnung schützen - mit POKE 55,0:POKE 56,92:POKE 52,92 -, und den Computer dann mit PRINT FRE(1) nach dem freien Speicherplatz fragen, erhalten Sie als Antwort immerhin satte 21501 freie Bytes.

Der Idealfall wäre es, wenn man die Bit-Map unter das ROM legen könnte (Kombinationen 6 oder 8). Das geht natürlich auch! Von Basic aus wird ein Programm dann allerdings noch langsamer, weil man für jeden Punkt ähnliche Operationen vornehmen müßte, wie wir sie in Kapitel 2 beim Kopieren des Zeichen-ROM ins RAM verwendet haben. Auch so ist die ganze Hochauflösungsgrafik schon ziemlich langsam. Wir werden bald einige Routinen in Maschinensprache kennenlernen, die uns mehr Möglichkeiten eröffnen.

4.3 Die Grafik bekennt Farbe: Der Bit-Map-Mehrfarben-Modus

Hochauflösende Grafik in Farbe: Kann das der C 64 überhaupt? Die Antwort lautet Ja und Nein. Ja, weil der bislang von uns verwendete Bit-Map-Modus anstelle der zwei bisher benutzten auch mit vier Farben laufen kann. Nein, weil die Punkteauflösung dann eigentlich nicht mehr die Bezeichnung "hochauflösend" verdient. Die horizontale Auflösung ist hier anstelle der bisher ansprechbaren 320 nur noch 160 Positionen groß. Meine persönliche Meinung dazu ist, ernsthafte hochauflösende Grafik sollte sich im bisher besprochenen Bit-Map-Modus abspielen, denn ein Bildschirm mit 64000 Bildpunkten ist schon eine Minimalanforderung. 32000 Bildschirmpositionen sind allenfalls für Spielereien ganz nett, wenn auch etwas teuer, denn ohne Farbmonitor haben Sie von der Farbe nicht viel und über den Unterschied zwischen Farbfernseher und Farbmonitor haben wir im Verlauf dieses Buchs schon einiges erfahren. Nach dieser etwas desillusionierenden Vorrede widmen wir uns der mehrfarbigen Bit-Map-Grafik. Es handelt sich um eine Kombination aus der bisher bekannten Bit-Map-Grafik und dem Mehrfarben-Modus, den wir schon bei den mehrfarbigen Zeichen kennengelernt haben. Auf dem Bildschirm wird der Inhalt der Bit-Map wiedergegeben, aber die Farben der

Zeichnungen sind abhängig von Bit-Paaren. Welche Farben Sie bei welcher Paarung sehen, darüber gibt Tabelle 4.3 Aufschluß.

BIT-PAAR	FARBQUELLE
00	SPEICHER 53281, HINTERGRUNDREG. Nr.0
01	BITS 4-7 } DES VIDEO-RAM
10	BITS 0-3 } DES VIDEO-RAM
11	BILDSCHIRMFARBSPEICHER

Tabelle 4.3 Herkunft der Farben im Bit-Map-Mehrfarbenmodus in Abhängigkeit von den Bit-Paaren

Es existiert also eine Hintergrundfarbe für den gesamten Bildschirm, die überall dort auftaucht, wo in der Bit-Map ein Bit-Paar 00 vorhanden ist. Diese Hintergrundfarbe ist durch den Zahlenwert in Register 53281 bestimmt. Die anderen drei Farben treten jeweils in den 8x8-Bit-Bildschirmfeldern auf, in die das Video-RAM, beziehungsweise der Bildschirmfarbspeicher aufgeteilt sind.

Wir ergänzen unser Programm 1 für den Mehrfarben-Modus: Außer dem Bit-Map-Modus muß hier noch der Mehrfarben-Modus eingeschaltet werden:

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

Weiterhin ändern wir noch die Zeile 5 und schreiben anstelle von F = 6 jetzt GOSUB 300. Folgende Zeilen kommen neu hinzu:

```
300 PRINT CHR$(147)CHR$(17)"FARBENWAHL:"
310 PRINTCHR$(17)"HINTERGRUNDFARBE"TAB (30);:INPUT F0
320 PRINT"BITS 4-7 VIDEOMATRIX "TAB(30);:INPUT F1
330 PRINT"BITS 0-3 VIDEOMATRIX"TAB(30);:INPUT F2
340 PRINT "BILDSCHIRMFARBSPEICHER"TAB(30);:INPUT F3
350 POKE53281,F0:F=16*F1 + F2:FOR I = 55296 TO 56295:POKE I,F:NEXT I
360 PRINT CHR$(147):RETURN
```

Um des Effektes willen ändern wir noch die Zeile 220, in der die Videomatrix mit den Farbzahlen belegt wird:

```
220 FOR J = 0 TO 998 STEP 2:POKE J,F:NEXT J:FOR J= 1TO 999 STEP 2:
POKE J,F+1:NEXT J.
```

Starten Sie dieses Programm nach dem Abspeichern mit RUN, probieren Sie alle möglichen Farbkennzahlen aus. Sie werden fast immer bemerken, daß die Hoch- und Tiefpunkte der Kurve nicht mitgezeichnet werden. Das liegt daran, daß unser Verfahren der Berechnung, welches Bit in welchem Byte gesetzt werden soll, noch auf einzelne Bits und nicht die paarweise Verwendung abgestimmt ist. Es gibt zwei Möglichkeiten: Entweder ändert man das Berechnungsverfahren in Zeile 240, um an die richtige Stelle die passenden Bit-Paare zu bekommen, oder wir ändern die Programmzeile 230 zu

```
230 FQR I = 0 TO 319 STEP 2:Y=FNA(X)
```

Das ist zwar mal wieder etwas primitiv, aber - wie bereits gesagt - sind meine Ambitionen zur Mehrfarben-"Hochauflösung" sowieso nicht so stark. Wer Lust hat, kann sich mit der korrekten Weise der neuen Berechnung herumschlagen. Das Wissen, diese Aufgabe zu bewältigen, haben Sie jetzt. Auf dem Schwarzweiß-Monitor sieht zum Beispiel folgende Zahlenkombination ganz gut aus:

Hintergrundfarbe:	7
Bits 4 bis 7:	5
Bits 0 bis 3:	1
Bildschirmspeicher:	0
Außerdem natürlich noch:	I=2, W=112, B=1

4.4 Langsam - aber zuverlässig: Eine kleine Grafik-Unterprogramm-Bibliothek in BASIC

Mit dem hier folgenden Abschnitt soll zunächst einmal die hochauflösende Grafik beiseite gelegt werden. Dornröschen ist erwacht und genesen, allerdings noch nicht voll bei Kräften. Das wird später noch anders werden. Davor wollen wir aber noch weitere Grafik-Besonderheiten des C 64 behandeln. An dieser Stelle werden wir einen Zwischenhalt einlegen und eine kleine Sammlung von Basic-Unterprogrammen zur Grafik-Programmierung vorstellen. In Programm 2 sind diese Grafik-Unterprogramme, in Programm 3 ein Beispiel-Aufrufprogramm abgedruckt. Beide Programme finden Sie auf der mitgelieferten Diskette

```
49990 REM ------------------
49991 REM --UNTERPROGRAMME-
49992 REM ------------------
49993 REM
49996 REM ------------------
49997 REM --SPRUNGTABELLE--
49998 REM ------------------
49999 REM
50000 GOTO50500:REM HIRES AN
50010 GOTO50600:REM BIT-MAP-LOESCHEN
50020 GOTO50700:REM FARBGEBUNG
50030 GOTO50800:REM HIRES AUS
50040 GOTO50900:REM PUNKT SETZEN
50050 GOTO51000:REM PUNKT LOESCHEN
50060 GOTO51100:REM STRECKE ZEICHNEN
50070 GOTO51200:REM STRECKE LOESCHEN
50080 GOTO51300:REM ELLIPSE ZEICHNEN
50090 GOTO51400:REM ELLIPSE LOESCHEN
50100 GOTO51500:REM KOMBINIERTES HIRES AN
50490 REM
50491 REM --------------------
50492 REM ----HIRES AN------
50493 REM --------------------
50494 REM
50500 POKE56576,(PEEK(56576)AND252)OR2
50510 POKE56578,PEEK(56578)OR3
50520 POKE53272,120:POKE648,92
50530 POKE53265,PEEK(53265)OR32
50540 RETURN
50590 REM
50591 REM --------------------
50592 REM -BIT-MAP-LOESCHEN--
50593 REM --------------------
50594 REM
50600 FORI=24576TO32575:POKEI,0:NEXTI
50610 RETURN
50690 REM
50691 REM ----------------------
50692 REM -----FARBGEBUNG------
50693 REM ----------------------
50694 REM
50700 F=16*F1+F2
50710 FORI=23552TO24551:POKEI,F:NEXTI
50720 RETURN
50790 REM
50791 REM ------------------------
50792 REM ------HIRES AUS-------
50793 REM ------------------------
50794 REM
50800 POKE53265,27:POKE53272,21:POKE648,4
50810 POKE56578,63:POKE56576,151
50820 RETURN
50890 REM
50891 REM ------------------------
50892 REM -----PUNKT SETZEN-----
50893 REM ------------------------
50894 REM
50900 L=0
```

```
50905 IFX<0ORX>319ORY<0ORY>199THEN50940
50910 BY=(XAND504)+40*(YAND248)+(YAND7):BI=7-(XAND7)
50920 IFL=1THENPOKE24576+BY,PEEK(24576+BY)ANDNOT(2↑BI):GOTO50940
50930 POKE24576+BY,PEEK(24576+BY)OR(2↑BI)
50940 RETURN
50990 REM
50991 REM ------------------------
50992 REM -----PUNKT LOESCHEN----
50993 REM ------------------------
50994 REM
51000 L=1:GOTO50905
51090 REM
51091 REM -------------------------
51092 REM ----STRECKE ZEICHNEN----
51093 REM -------------------------
51094 REM
51100 L=0
51110 IFABS(X2-X1)<5THEN51150
51115 FORX=X1TOX2STEP(X2-X1)/319
51120 Y=(Y2-Y1)/(X2-X1)*(X-X1)+Y1
51130 GOSUB50905:NEXTX
51140 RETURN
51150 FORY=Y1TOY2STEP(Y2-Y1)/199
51160 X=(X2-X1)/(Y2-Y1)*(Y-Y1)+X1
51170 GOSUB50905:NEXTY
51180 RETURN
51190 REM
51191 REM -------------------------
51192 REM ----STRECKE LOESCHEN-----
51193 REM -------------------------
51194 REM
51200 L=1:GOTO51110
51290 REM
51291 REM ---------------------------
51292 REM -----ELLIPSE ZEICHNEN-----
51293 REM ---------------------------
51294 REM
51300 L=0
51310 FORW=WUTOWO:WB=W*π/180
51320 X=XM+HX*COS(WB):Y=YM+HY*SIN(WB)
51330 GOSUB50905
51340 NEXTW:RETURN
51390 REM
51391 REM ---------------------------
51392 REM -----ELLIPSE LOESCHEN-----
51393 REM ---------------------------
51394 REM
51400 L=1:GOTO51310
51490 REM
51491 REM ---------------------------
51492 REM --KOMBINIERTES HIRES AN---
51493 REM ---------------------------
51494 REM
51500 GOSUB50000:GOSUB50010:GOTO50020
```

Listing 4.2 Unterprogramm zur hochauflösenden Grafik (Programm 2)

4.4.1 Erläuterungen zu Programm 2 (Zeilenbereich 49990 bis 51500)

Sprungtabelle:

Diese Art der Programmierung ist in Basic im allgemeinen nicht üblich, sondern wird häufiger bei Maschinensprache-Programmen verwendet. Trotzdem hat es auch hier seine Vorteile. Es kann ja sein, daß Sie einige Änderungen oder Ergänzungen in den Unterprogrammen vornehmen wollen. Sie müßten dann auch immer die Adressen in den jeweils aufzurufenden Hauptprogrammen umschreiben. Mit der Sprungtabelle ist das nicht mehr nötig, denn die GOSUB-Adressen im Hauptprogramm bleiben unverändert, nur die neuen GOTO-Adressen im Unterprogramm sind einzusetzen.

HIRES an

Hier machen wir uns die Erkenntnisse aus diesem Kapitel zunutze und legen den Bildschirm nach 23552 und die Bit-Map nach 24576. Beides müssen wir - wie gehabt - vor dem Überschreiben durch Basic schützen:

```
POKE 52,92:POKE 56,92.
```

Am besten packt man diese POKE-Befehle gleich in die ersten Zeilen des aufrufenden Hauptprogramms.

Bit-Map-Löschen

Hierzu gibt es nichts mehr zu sagen, außer, daß I als Laufvariable dient.

Farbgebung

Bevor dieses Unterprogramm aufgerufen wird, müssen im Hauptprogramm

F1 = Zeichenfarbe und
F2 = Hintergrundfarbe

definiert sein. An Variablen treten noch auf:

F = Farbcodezahlen auf F1 und F2
I = Laufvariable

HIRES aus

Dieses Programm stellt die ursprüngliche Speicherorganisation wieder her (Bildschirm- und Zeichenspeicher) und schaltet in den Normalmodus zurück.

Punkt setzen

Auch hier müssen vor dem Aufruf des Unterprogramms im Hauptspeicher die Punktkoordinaten X,Y definiert sein (siehe Bild 4.3) sowie die L-Löschmarke. Wenn L = 0 ist, wird der Punkt gesetzt (Zeile 50930), so, wie wir das schon kennen. Ist L = 1, wird ein vorhandener Punkt gelöscht (Zeile 50920).

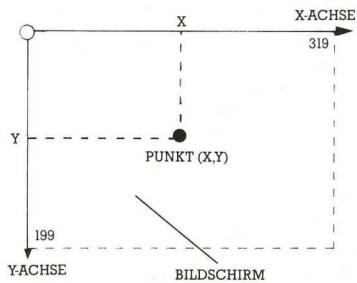


Bild 4.3 Ein Punkt (X,Y) im Bildschirmkoordinatensystem (0<=X<=319,0<=Y<=199)

Die Zeile 50905 achtet darauf, daß keine Punktkoordinate außerhalb des Bildschirms liegt, was unter Umständen ein Aussteigen mit einer Fehlermeldung im hochauflösenden Modus zur Folge hätte. Das ist hier zwar nicht so tragisch, weil man durch Eingeben von GOTO 50030 <RETURN> schnell wieder in den Normalmodus gelangen kann (eventuell muß vorher noch <"SHIFT" + "CLR/HOME"> gedrückt werden). Trotzdem ist es dumm, wenn inmitten all dieser zeitraubenden Grafiktätigkeiten auch noch das Programm aussteigt. Eine Grenzüberschreitung der Koordinaten ist um so leichter möglich, als die Punkt-Routine von allen folgenden Unterprogrammen aufgerufen wird. Außer X,Y und L tauchen noch die Variablen BY und BI auf, die wir schon kennengelernt haben als das Byte, in dem ein Bit zu setzen oder zu löschen ist.

Punkt löschen

Hier geschieht nichts anderes, als die Löschmarke L auf 1 zu setzen und dann in die Punkt-Setz-Routine zu springen. Deswegen gilt für dieses Unterprogramm dasselbe wie für das vorangegangene.

Strecke zeichnen

Vor dem Aufruf müssen dem Rechner schon der Startpunkt (X1,Y1) und der Endpunkt (X2,Y2) der Strecke bekannt gemacht sein.

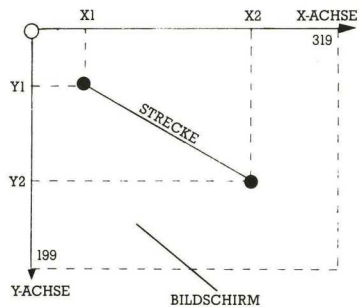


Bild 4.4 Eine Strecke (X1,Y1) bis (X2,Y2) im Bildschirmkoordinatensystem

Den mathematisch Versierten wird bei der Betrachtung der Zeilen 51120 beziehungsweise 51160 schon aufgefallen sein, daß zur Berechnung der Punkte, aus denen sich die Strecke zusammensetzt, die sogenannte 2-Punkte-Form der Geradengleichung verwendet wurde:

$$\frac{Y-Y1}{X-X1} = \frac{Y2-Y1}{X2-X1}$$

Den mit Mathematik nicht so vertrauten Lesern sei gesagt, daß es sich um eine Formel aus der sogenannten analytischen Geometrie handelt. Das ist ein Gebiet der Mathematik, das für die Grafik auf Computern eine nicht unerhebliche Rolle spielt.

Die Punkte (X1,Y1) und (X2,Y2) dürfen auch außerhalb des Bildschirmsystems liegen. Im ersten Teil des Unterprogramms dient die Übergabevariable X auch gleichzeitig als

Laufvariable,während Y berechnet wird. Wenn allerdings der Absolutbetrag von X2-X1 kleiner als 5 wird, kehren sich die Verhältnisse um: Y wird Laufvariable und X berechnet. Das beschleunigt das Zeichnen von Senkrechten und verhindert außerdem eine Division durch Null. Der Wert von 5 ist dabei ziemlich willkürlich gewählt. L ist wieder die Löschmarke.

Strecke löschen

Es gilt dasselbe wie für das Unterprogramm Strecke zeichnen, nur daß hier wieder die Löschmarke gesetzt wird.

Ellipse zeichnen

Vor dem Aufruf müssen folgende Werte schon definiert sein (siehe auch Bild 4.5):

(XM,YM)	= Mittelpunktkoordinaten
HX	= Halbmesser in X-Richtung
HY	= Halbmesser in Y-Richtung
WU,WO	= Der zu zeichnende Ellipsenbogen beginnt beim Winkel WU und endet beim Winkel WO (Gradmaß)

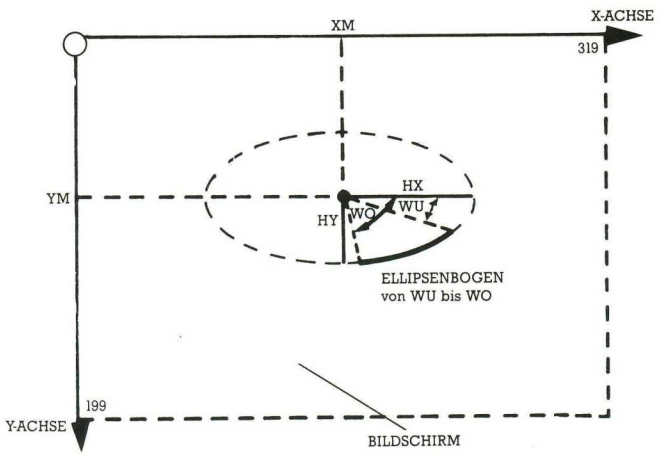


Bild 4.5 Elipsenbogen der Ellipse mit Mittelpunkt (XM,YM), Halbmessern HX und HY, von Winkel WU bis WO im System der Bildschirmkoordinaten

Eine volle Ellipse wird gezeichnet, wenn WU = 0 und WO = 360 ist. Der Kreis ist ein Sonderfall der Ellipse. Dann muß nur HX = HY sein.

Für mathematisch Interessierte: Es werden die Parametergleichungen der Ellipse verwendet:

X = XM + HX*COS(BW) und
Y = YM + HX*SIN(WB)

Auch hier gibt es keine Einschränkung wie beim Strecken-Zeichnen in der Größe von XM, YM, HX, WU, WO.

W ist eine Laufvariable (ein Winkel) der in WB (gleicher Winkel im Bogenmaß umgerechnet wird. L ist wieder die Löschmarke.

Ellipse löschen

Bis auf das Setzen der Löschmarke gilt dasselbe wie für das Zeichnen der Ellipse.

Kombination

Erfordert schon definierte Farbkennzahlen F1 und F2 (siehe Farbgebung) und schaltet dann in den hochauflösenden Modus um, löscht die Bit-Map und sorgt für die Farbe.

Soweit die Unterprogramme in Listing 2.

4.4.2 Ein Beispiel für die Möglichkeiten der Grafik-Programmbibliothek

Als ein Beispiel für die Möglichkeiten der Unterprogramm-Sammlung habe ich (ohne nun besonders auf Schönheit zu achten - das sind Sie ja von mir schon gewohnt), noch ein Hauptprogramm angefügt, mit dem Sie etwas herumprobieren können (Programm 3). Das Listing ist ausführlich kommentiert, so daß hier nur wenige Erläuterungen folgen müssen.

Beim Eintippen habe ich für einige Zeilen die Abkürzungen (siehe Handbuch Seite 130ff) der Basic-Befehle eingeben, da die Zeilen sonst länger als 80 Zeichen geworden wären. Die Kombination der beiden Programme finden Sie unter dem Namen "Prog.3 + Prog.2" auf der Diskette.

```
1 REM ********************************
2 REM *  GRAFIK TEST PROGRAMM VON    *
3 REM *  H.PONNATH 1984 VERBROCHEN   *
4 REM ********************************
5 POKE52,92:POKE56,92:DEFFNA(X)=50*SIN(X/30)+100
7 REM ********************************
8 REM *      MENUE-GUTEN APPETIT     *
9 REM ********************************
10 PRINTCHR$(147)CHR$(17)"UNTERPROGRAMME GRAFIK TEST"CHR$(17)
20 PRINTTAB(2)"NUR HIRES AN"TAB(25)"1"
30 PRINTTAB(2)"NUR HIRES AUS"TAB(25)"2"
40 PRINTTAB(2)"BIT MAP LOESCHEN"TAB(25)"3"
50 PRINTTAB(2)"FARBGEBUNG"TAB(25)"4"
60 PRINTTAB(2)"KOMBINATION"TAB(25)"5"
62 PRINTTAB(2)"PUNKTE SETZEN"TAB(25)"6"
64 PRINTTAB(2)"PUNKTE LOESCHEN"TAB(25)"7"
66 PRINTTAB(2)"STRECKE ZEICHNEN"TAB(25)"8"
68 PRINTTAB(2)"STRECKE LOESCHEN"TAB(25)"9"
70 PRINTTAB(2)"ELLIPSE ZEICHNEN"TAB(25)"A"
72 PRINTTAB(2)"ELLIPSE LOESCHEN"TAB(25)"B"
74 PRINTTAB(2)"DEMONSTRATION"TAB(25)"C"
76 PRINTTAB(1)"MENUE"TAB(25)"M"
78 PRINTTAB(1)"AUSSTEIGEN"TAB(25)"←"
80 GETA$:IFA$=""THEN80
90 IFA$="←"THENEND
92 IFA$="A"THENA$="10"
94 IFA$="B"THENA$="11"
96 IFA$="C"THENA$="12"
98 IFA$="M"THEN10
100 ONVAL(A$)GOSUB50000,50030,50010,200,300,400,500,600,700,800,900,1000
110 GOTO80
190 REM ****************************
191 REM *      FARBGEBUNG          *
192 REM ****************************
200 INPUT"ZEICHENFARBE,HINTERGRUNDFARBE=";F1,F2:GOTO50020
290 REM ****************************
291 REM * FARBE FUER KOMBINATION  *
292 REM ****************************
300 INPUT"ZEICHENFARBE,HINTERGRUNDFARBE=";F1,F2:GOTO50100
390 REM ****************************
391 REM * BEISPIEL PUNKTE SETZEN  *
392 REM ****************************
400 GOSUB50000:FORX=0TO319:Y=FNA(X):GOSUB50040:NEXTX
410 RETURN
490 REM ****************************
491 REM * BEISPIEL PUNKTE LOESCHEN*
492 REM ****************************
500 GOSUB50000:FORX=20TO250:Y=FNA(X):GOSUB50050:NEXTX
510 RETURN
590 REM ****************************
591 REM * AUFRUFPROGRAMM FUER     *
592 REM * STRECKE ZEICHNEN        *
593 REM ****************************
600 PRINT"■STRECKE■ VON (X1,Y1) BIS (X2,Y2)":INPUT"X1,Y1,X2,Y2=";X1,Y1,X2,Y2
610 GOSUB50000:GOTO50060
690 REM ****************************
691 REM * AUFRUFPROGRAMM FUER     *
692 REM * STRECKE LOESCHEN        *
693 REM ****************************
700 PRINT"■STRECKE■ VON (X1,Y1) BIS (X2,Y2)":INPUT"X1,Y1,X2,Y2=";X1,Y1,X2,Y2
710 GOSUB50000:GOTO50070
790 REM ****************************
791 REM * AUFRUFPROGRAMM FUER     *
792 REM * ELLIPSE ZEICHNEN        *
793 REM ****************************
800 PRINT"■ELLIPSE■ MIT MITTELPUNKT (XM,YM),":PRINTTAB(1)"HALBMESSERN HX UND HY"
810 PRINTTAB(1)"ZEICHNEN VON WINKEL WU":PRINTTAB(1)"BIS WINKEL WO (GRADMASS)"
820 INPUT"XM,YM,HX,HY,WU,WO=";XM,YM,HX,HY,WU,WO:GOSUB50000:GOTO50080
```

```
890 REM ****************************
891 REM * AUFRUFPROGRAMM FUER      *
892 REM * ELLIPSE LOESCHEN         *
893 REM ****************************
900 PRINT"■ELLIPSE■ MIT MITTELPUNKT (XM,YM),":PRINTTAB(1)"HALBMESSERN HX UND HY"
910 PRINTTAB(1)"LOESCHEN VON WINKEL WU":PRINTTAB(1)"BIS WINKEL WO (GRADMASS)"
920 INPUT"XM,YM,HX,HY,WU,WO=";XM,YM,HX,HY,WU,WO:GOSUB50000:GOTO50090
990 REM ****************************
991 REM * AUFRUFPROGRAMM FUER      *
992 REM * DEMONSTRATION            *
993 REM ****************************
1000 GOSUB50030:PRINTCHR$(147):FORI=1TO10:PRINTCHR$(17);:NEXTI
1010 PRINTTAB(8)"BITTE ETWAS GEDULD":PRINTTAB(5)"DIE BIT-MAP WIRD GELOESCHT"
1020 GOSUB50010:PRINTTAB(5)"UND MIT FARBE VERSEHEN":FORI=1TO1000:NEXTI
1030 F1=7:F2=6:GOSUB50000:GOSUB50020:FORI=1TO500:NEXT:GOSUB50030
1040 PRINTCHR$(147):FORI=1TO10:PRINTCHR$(17);:NEXTI
1050 PRINTTAB(5)"ZEICHNEN VON STRECKEN":FORI=1TO500:NEXTI
1060 GOSUB50000:FORI=0TO12
1070 X1=30+I*10:Y1=180-I*14.17:X2=150+I*13.3:Y2=10+I*14.583
1080 GOSUB50060:NEXTI:FORK=0TO9:F1=K:F2=K+1:GOSUB50020:FORJ=1TO500:NEXTJ:NEXTK
1090 GOSUB50030:PRINT:PRINTTAB(5)"STRECKEN LOESCHEN"
1100 FORI=1TO500:NEXTI:GOSUB50000:X1=30:Y1=180:X2=150:Y2=10:GOSUB50070
1110 X1=310:Y1=185:GOSUB50070:FORI=1TO500:NEXTI:GOSUB50030
1120 PRINT:PRINTTAB(5)"ELLIPSEN ZEICHNEN":FORI=1TO500:NEXTI:GOSUB50000
1130 XM=170:YM=150
1140 FORI=0TO16:WU=I*90:WO=WU+90
1150 HX=20+8*INT((3+I)/4):HY=10+8*INT((2+I)/4)
1160 GOSUB50080:NEXTI:X1=0:Y1=0:X2=319:Y2=0:GOSUB50060
1170 X2=0:Y2=199:GOSUB50060:X1=319:Y1=199:GOSUB50060:X2=319:Y2=0:GOSUB50060
1180 X1=100:Y1=0:X2=100:Y2=80:GOSUB50060:X1=0:Y1=80:GOSUB50060
1190 FORX=0TO100:Y=40+25*SIN(X/15):GOSUB50040:NEXTX
1200 X1=219:Y1=0:X2=219:Y2=80:GOSUB50060:X1=319:Y1=80:GOSUB50060
1210 XM=249:YM=40:HX=20:HY=30:WU=0:WO=360:GOSUB50080
1220 X1=279:Y1=10:X2=279:Y2=70:GOSUB50060:Y1=40:X2=309:Y2=10:GOSUB50060
1230 Y2=70:GOSUB50060
2000 RETURN
```

Listing 4.3 Grafik-Test und Demonstration (Programm 3)

Nach "RUN" sehen Sie ein Menü, das alle Möglichkeiten der Grafik-Unterprogramme ansteuert. Die Optionen 8 (Strecke zeichnen) bis B (Ellipse löschen) sowie 4 (Farbgebung) und 5 (Kombinationen) erfordern Eingaben. Es ist daher sinnvoll, diese Optionen nur im Normalmodus anzuwählen. Der Normalmodus ist immer dann zu erreichen, wenn Zeichenoperationen im hochauflösenden Modus abgeschlossen sind. Drücken Sie dann "2", sind Sie wieder im normalen Rechnerbetrieb.

Sollten Sie durch irgendeinen Umstand (zum Beispiel durch Drücken der <RUN/STOP>-Taste) im hochauflösenden Modus aus dem Programm fallen, hilft der folgende Weg:

1. <SHIFT> + <CLEAR/HOME>
2. <RUN> <RETURN>
3. dann "2" eingeben.

Die Option "C" zeigt eine kleine Demonstration von Möglichkeiten der Grafik-Unterprogramme. Allerdings sollten Sie ein bißchen Zeit mitbringen, wenn Sie C anwählen: Das ganze dauert zirka 25 Minuten.

Option 6 (Punkte zeichnen) ist so eingerichtet, daß 320 Punkte in Form einer Sinus-Funktion gezeichnet und mit Option 7 (Punkte löschen) teilweise wieder gelöscht werden.

Dieses Kapitel soll nicht ohne einen kleinen tröstlichen Ausblick beendet werden. Wie Sie - besonders im letzten Programm - feststellen konnten, braucht man schon einiges Sitzfleisch für hochauflösende Grafik in Basic. Wenn Sie aber ein kommerzielles Grafik-Programm laufen sehen, geht das alles erheblich schneller. Wo liegt der Unterschied? Da wäre zunächst einmal die Programmiersprache: Unser C 64 kann eigentlich gar nicht Basic. Er braucht den Basic-Interpreter, der zunächst jeden Befehl liest und dann in Maschinensprache übersetzt. Die versteht unser Rechner zwar, die Übersetzung und das Lesen dauern jedoch sehr lange. Eine starke Beschleunigung der Grafik ist durch Programmieren in Maschinensprache möglich. Einige solche Maschinenprogramme zur beschleunigten Grafik finden Sie als HIRES-3 in Kapitel 8 und auf der Diskette. Allerdings stoßen wir da bald an die Grenzen unseres Commodore. Ein 8-Bit-Computer mit zirka 1 Megahertz Taktfrequenz wie unser C 64 ist beispielsweise in der Fließkomma-Arithmetik (wie sie für das Zeichnen von Ellipsen nötig ist) zeitlich gehandicapt, und deswegen sind der Geschwindigkeit bei komplexerer Grafik doch Grenzen gesetzt.

5

Sprites

5 Die Sprites

Sprite heißt auf deutsch soviel wie "Kobold", "Gespenst". Wie richtige Kobolde können sie sowohl in Dornröschens Schloß (also im Bit-Map-Modus) als auch außerhalb (nämlich im Normalmodus) über den Bildschirm geistern - anscheinend dabei allen bisher gelernten Regeln über die Grafik widersprechend. Wir werden in diesem Kapitel zwar lernen, mit ihnen umzugehen, und sie zu beherrschen - aber ihre genaue Funktion und Herkunft wird weiterhin im Dunkeln bleiben: Meines Wissens gibt es noch kein Listing des Sprite-unterstützenden Maschinenprogramms, das wohl im tiefsten Dunkel des VIC-II-Chips verborgen liegt: Denn der VIC-II-Chip belegt die Speicherplätze 53248 bis 54271. Im erreichbaren Teil dieses Zauberschlosses (53248 bis 53294) liegen die bisher viel von uns begangenen 47 Register (siehe Tabelle 1.1), aber wo ist die Geheimtür zu den anderen 978 Bytes? Alles recht romantisch, werden Sie sagen. Nun - ganz so mysteriös ist die Sache nun auch wieder nicht, wie uns der Name Sprite = Kobold, Gespenst einreden will. Gehörig entschleiert wird das Geheimnis schon durch den anderen englischen Ausdruck für diese Dinger: MOBs. Das bedeutet: Movable Object Blocks, also bewegliche Blöcke von Objekten (Bildern, Darstellungen). Sie werden sehen, wenn wir mit den MOBs umgehen können, ist das verbleibende Geheimnis eigentlich keines mehr, sondern verwandelt sich nur noch in eine Herausforderung für einen Maschinensprache-Programmierer.

Wir werden zunächst einmal schöpferisch tätig und denken uns ein Sprite aus. Der erste Schritt dazu ist kreativ: Wie soll das Ding aussehen? Da bietet sich zum Beispiel der aus der Magie bekannte Drudenfuß an, auch Pentagramm genannt, weil wir ja schließlich diese Geister bannen wollen.

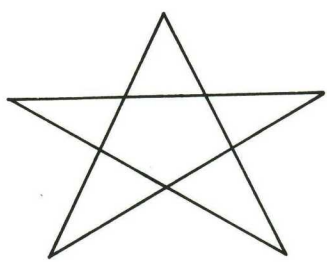


Bild 5.1 Ein Drudenfuß oder Pentagramm

Jetzt müssen wir diese Zeichnung in eine Form bringen, die unser Computer versteht, also in Bytes. Wie auch schon bei den Buchstaben und der hochauflösenden Grafik sind hier wieder gesetzte und gelöschte Bits in den Bytes die Anzeige für "Punkt sichtbar" oder "Punkt nicht sichtbar". Das im VIC-II-Chip organisierte Sprite-Programm nimmt die Bytes in der im Bild 5.2 gezeigten Anordnung wahr.

	Abschnitt 1	Abschnitt 2	Abschnitt 3
Zeile 1	Byte 0	Byte 1	Byte 2
Zeile 2	Byte 3	Byte 4	Byte 5
.	.	.	.
.	.	.	.
.	.	.	.
Zeile 20	Byte 57	Byte 58	Byte 59
Zeile 21	Byte 60	Byte 61	Byte 62

Bild 5.2 Die Sprite-Organisation im VIC-II-Chip

Den so gebildeten Block verwaltet es in genau dieser Anordnung als ein Sprite. Deswegen müssen wir uns nun die Mühe machen, unser Pentagramm in so ein Bit-Raster einzufügen.

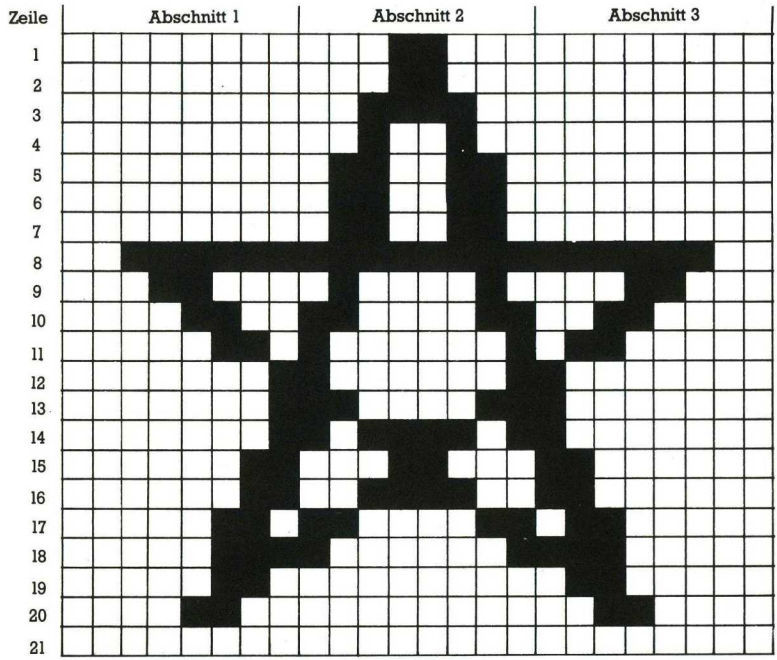


Bild 5.3 Das Pentagramm im Sprite-Bitraster

Nun muß dieses Bild in einen Zahlencode übersetzt werden. Überall dort, wo ein Bitfeld ausgefüllt ist, steht bei der Binärzahl eine 1, sonst eine Null. Demnach ergeben sich die Kennzahlen in der Tabelle 5.1.

Zeile	Byte	binär	dez.	Zeile	Byte	binär	dez.
	0	00000000	4		33	00000001	1
1	1	00011000	24	12	34	10000001	129
	2	00000000	0		35	10000000	128
	3	00000000	0		36	00000001	1
2	4	00011000	24	13	37	11000011	195
	5	00000000	0		38	10000000	128
	6	00000000	0		39	00000001	1
3	7	00111100	60	14	40	10111101	189
	8	00000000	0		41	10000000	128
	9	00000000	0		42	00000011	3
4	10	00100100	36	15	43	00011000	24
	11	00000000	0		44	11000000	192
	12	00000000	0		45	00000011	3
5	13	01100110	102	16	46	00111100	60
	14	00000000	0		47	11000000	192
	15	00000000	0		48	00000110	6
6	16	01100110	102	17	49	11000011	195
	17	00000000	0		50	01100000	96
	18	00000000	0		51	00000111	7
7	19	01100110	102	18	52	10000001	129
	20	00000000	0		53	11100000	224
	21	00111111	63		54	00000110	6
8	22	11111111	255	19	55	00000000	0
	23	11111100	252		56	01100000	96
	24	00011000	24		57	00001100	12
9	25	01000010	66	20	58	00000000	0
	26	00011000	24		59	00110000	48
	27	00001100	12		60	00000000	0
10	28	11000011	195	21	61	00000000	0
	29	00110000	48		62	00000000	0
	30	00000110	6				
11'	31	10000001	129				
	32	01100000	96				

Tabelle 5.1 Kennzahlen für die Berechnung von Sprites

Dieser Wust an Zahlen legt unser Sprite fest. Diese doch recht aufwendige Rechnerei und Planerei ist, wenn Sie aufmerksamer Leser von Computerliteratur sind - meist nicht mehr nötig. Obwohl es sicher gut ist, das Planen eines MOBs auch von Hand zu beherrschen (wie wir jetzt!), kann man sich die Arbeit durch Eintippen eines der vielen Sprite-Editor-Programme sehr erleichtern , die es in fast allen Fachzeitschriften wie Sand am Meer gibt. Weil hier nicht der 1001. Sprite-Editor abgedruckt werden soll, dient das Programm, das wir hier erarbeiten, zu Lehrzwecken. (Ein nettes kurzes Listing von H. Kunz finden Sie zum Beispiel in der Zeitschrift Computer persönlich

Nr. 21, Jahrgang 1983 auf Seite 120). Jetzt müssen wir noch dafür sorgen, daß der C 64 diese Zahlen irgendwo zugreifbar hat, mit anderen Worten: Sie müssen in den Speicher gePOKEd werden. Im allgemeinen verwendet man dazu eine kleine FOR-NEXT-Schleife, in der die in DATA-Zeilen abgelegten Zahlen gelesen und eingePOKEd werden. Wohin packt man die Kennzahlen? Im Prinzip kann man sie überall - wo sie nicht gerade lebenswichtige Computerfunktionen oder Basicprogramme stören - unterbringen. Es gibt lediglich zwei Dinge, die zu beachten sind:

a) Die Startadresse muß ohne Rest durch 64 teilbar sein. (Zum Beispiel 896 = 14 mal 64 etc).

b) Aus Gründen, auf die wir noch zu sprechen kommen werden, sollten die Sprite-Daten im gleichen 16 Kbyte-Speicherabschnitt abgelegt werden, in dem sich das Video-RAM befindet, welches bei der Sprite-Nutzung eingeschaltet ist.

Damit gibt es im Prinzip 256 Orte pro Speicherabschnitt, in denen die Sprite-Daten gespeichert werden können. Allerdings sind einige Stellen zum Beispiel im Abschnitt 0 (mit dem normalen Bildschirm) bei der Verwendung von nur wenigen Sprites besonders bevorzugt, weil man keinen Basicspeicher wegnimmt und deshalb auch keine Schutz-POKEs nötig sind:

1) 704 - 767 ungenutzte Adressen
2) 832 - 895 Kassettenpuffer
3) 896 - 959 Kassettenpuffer
4) 960 - 1023 restlicher Kassettenpuffer und freier Platz.

Für weitere MOBs muß dann Basicspeicher verwendet und dieser vor dem Überschreiben durch Programmtext, Variablen oder Strings - wie in Kapitel 2 gezeigt - geschützt werden. Dem Erfindungsreichtum sind allerdings keine Grenzen gesetzt. So könnte man beispielsweise das Bild an den oberen Rand des Abschnitts 2 verschieben und darunter oder darüber Sprite-Daten ablegen. Man muß dann zwar auch den Speicherschutz einPOKEn, hat aber trotzdem meistens noch mehr als genug Basicspeicherplatz. Wenn man nur vier Sprites gleichzeitig verwendet, insgesamt aber mehrere definiert hat, kann man sie alle im 4 KByte-Bereich ab $C000 abspeichern und bei Bedarf den Sprite, der dran ist, mit einer kleinen FOR-NEXT-Schleife in einen der vier Speicherbereiche (704... und so weiter) umladen. Sicher fallen Ihnen noch mehr Möglichkeiten ein. Wir werden uns im nachfolgenden einfach mit drei Sprites begnügen und uns nur im normalen Abschnitt 0 bewegen.

Legen wir also nun zunächst unser Pentagramm in den Bereich 704 bis 767 und nach 832 bis 895:

```
10 FOR I = 0 TO 62:READ A:POKE 704+I,A:POKE832+I,A:NEXT I
30 DATA hier werden jetzt unsere 63 Kennzahlen eingegeben.
.
60 DATA.
```

5.1 Na, wo ist er denn? Sprite-Zeiger

Wo die Sprite-Daten liegen, wird dem Computer durch 8 Sprite-Zeiger mitgeteilt. Sie befinden sich immer im gleichen 1 KByte-Bereich, in dem auch der Bildschirmspeicher liegt. Weil dieses Video-RAM nur 1000 Bytes benötigt, sind oberhalb desselben noch 24 Bytes frei, von denen die obersten 8 als Sprite-Zeiger dienen. Wenn also im Normalfall der Bildschirmspeicher von 1024 bis 2023 geht, liegen die Sprite-Zeiger von 2040 bis 2047. Verschiebt man den Bildschirm, werden die Sprite-Zeiger mit verschoben. In diese Sprite-Zeiger-Bytes POKEn wir die Zahlen ein, die mit 64 multipliziert die Startadresse der Sprite-Daten ergeben. In unserem Beispiel:

704/64 = 11 und 832/64 = 13.

Wenn wir diese Sprite-Zeiger eingeben, nehmen wir automatisch gleichzeitig auch die Numerierung vor. Dabei gehört zu

Sprite Nr. 0 der Sprite-Zeiger 2040
Sprite Nr. 1 der Sprite-Zeiger 2041
Sprite Nr. 2 der Sprite-Zeiger 2042

und so weiter. Nennen wir den Sprite, dessen Daten von 704 bis 767 liegen, Nummer 0 und den anderen Nummer 1, müssen wir

```
70 POKE 2040,11:POKE 2041,13
```

eingeben. Damit sind die Sprite-Zeiger gesetzt.

5.2 Einschalten der Sprites

Wenn Sie jetzt freudig RUN ‹RETURN› eingetippt haben, um unsere Drudenfüsse zu sehen, werden Sie ein langes Gesicht gemacht haben: Nix zu sehen! Das ist wie bei einer Lampe: Sie haben den Lampenschirm, die Glühbirne, den Stecker eingesteckt und sie leuchtet nicht! Deswegen schalten wir sie jetzt ein, die Sprites. Der Schalter dafür sitzt im Sprite-Kontrollregister 53269 (siehe Tabelle der VIC-II-Chip-Register). Dort gibt es für jedes MOB ein Bit. Sprite Nummer 0 entspricht Bit Nummer 0 und so weiter. Ein Sprite ist eingeschaltet, wenn sein Bit auf 1 gesetzt ist. Wie man einzelne Bits setzt oder löscht, kennen wir noch aus Kapitel 2. Spielen wir das hier nochmal an unserem Beispiel durch: Wir wollen Sprite Nummer 0 und Sprite Nummer 1 einschalten, müssen also die Bits 0 und 1 auf den Wert 1 setzen. Da gab es doch die OR-Funktion und eine sogenannte Maske:

```
      XXXXXXXX      ein beliebiger binärer Inhalt von 53269
OR    00000011      Maske (= dezimal 3)
      XXXXXX11      Ergebnis (Bits 0 und 1 sind = 1)
```

Das ergäbe die Programmzeile:

```
80 POKE 53269,PEEK(53269)OR 3
```

Will man einzelne MOBs mit der Nummer N einschalten, empfiehlt sich folgender Befehl:

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

Das Ausschalten geschieht mit der AND-Funktion. Jedes Bit, das mit 0 AND-verknüpft wird, wird dadurch gelöscht. Wenn wir Sprite Nummer 1 ausschalten wollen, verwenden wir wieder eine Maske:

```
      XXXXXX11 PEEK(53269)    Sprite 0 und 1 eingeschaltet
AND   11111101                Maske (dezimal = 253)
      XXXXXX01                Ergebnis: Bit 1=0, Sprite 1 ausgeschaltet,
                              Bit 0 = 1, Sprite 0 eingeschaltet.
```

Allgemein kann man einzelne Sprites mit

```
POKE 53269, PEEK(53269) AND (255-2 N)
```

abschalten, wobei wieder N die Sprite-Nummer ist.

Ach, Ihre Geduld wird schon auf eine harte Probe gestellt. Wenn Sie nämlich bis jetzt alle Programmzeilen brav eingegeben und gestartet haben, sehen Sie immer noch kein Sprite! Aber Sie müssen dem MOB noch sagen, wo er erscheinen soll!

5.3 Schon wieder ein Koordinatensystem: Ort der Sprites

Vor den Erfolg haben auch die Commodore-Softwareplaner den Schweiß gesetzt! Denn nicht genug damit, daß wir den Bildschirm schon im Normalmodus in X-Richtung, in Y-Richtung in 25 Positionen und im Bit-Map-Modus in X-Richtung in 320, in Y-Richtung in 200 Positionen aufgeteilt finden, jetzt kommt noch eine Einteilung, die sogar über den sichtbaren Bildschirm hinausreicht! In Bild 5.4 sehen wir diese Aufteilung in 512 horizontale und 256 vertikale Koordinaten.

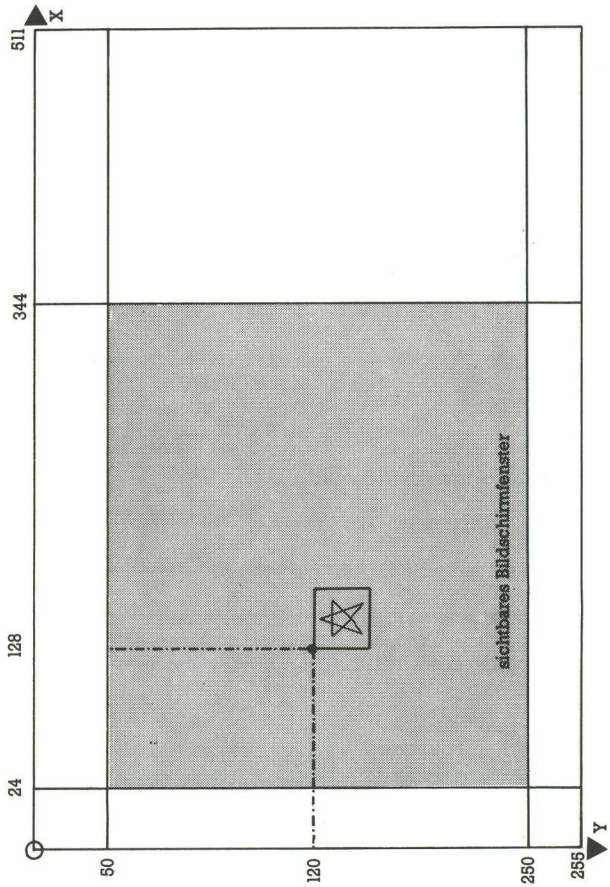


Bild 5.4 Sprite-Koordinatensystem

Für den Ort eines Sprites ist - wie im Bild angedeutet - die linke obere Ecke des Spritedefinitionsfeldes entscheidend. Also auch dann, wenn diese Ecke (wie in unserem Pentagramm) unsichtbar ist. So hat das im Bild gezeigte MOB die X-Koordinate 18 und die Y-Koordinate 120. Diese X- und Y-Werte muß man nun in die zur Sprite-Nummer gehörenden Register einPOKEn. Dabei handelt es sich um folgende Speicherstellen:

X-Position von Sprite 0 : 53248
Y-Position von Sprite 0 : 53249

X-Position von Sprite 1 : 53250
Y-Position von Sprite 1 : 53251

und so weiter bis

X-Position von Sprite 7 : 53262
Y-Position von Sprite 7 : 53263

Um nun endlich unser Sprite Nummer 0 sichtbar zu machen, geben wir ein:

```
100 POKE 53248,128:POKE53249,120
```

So! Jetzt tippen Sie RUN ‹RETURN› ein und endlich erscheint unser Pentagramm. Gefällt es Ihnen? Sprite 1, das zweite Pentagramm, soll am rechten Bildschirmrand auftauchen, also ungefähr bei X = 266 und bei Y = 130. Deswegen müßten wir aber in die entsprechende X-Positions-Speicherstelle für Sprite 1 eine Zahl größer als 255 einPOKEn! Wenn Sie's versuchen, meldet der Computer natürlich einen ILLEGAL QUANTITY ERROR. Geht also nicht! Anscheinend war das bisher noch nicht verzwickt genug. Jetzt müssen wir nämlich mal wieder das Hexadezimalsystem bemühen (die Sechzehnfingerlinge, erinnern Sie sich: Kapitel 2).

Dezimal 266 ist hexadezimal $010A. Jetzt zerteilen wir diese Zahl wieder in das LSB und das MSB:

$ 01	0A
MSB	LSB
= 1	=0A

und rechnen wieder zurück ins Dezimalsystem:

MSB = 1
LSB = 10

Rechnen Sie bitte nach: Das MSB kann auch bei der höchsten X- Position nie größer als 1 werden. Es gibt also nur zwei Fälle: Ist die X-Position größer als 256, dann ist das MSB 1, sonst ist es 0.

Deswegen speichert man die MSBs aller 8 Sprites in nur einem Byte. Ebenso wie beim Kontrollregister gehört auch hier zu jedem Sprite ein Bit für das Ein-und Ausschalten : Bit 1 gehört zu Sprite 1 und so weiter. Wenn nun die X-Koordinate von Sprite 1 gleich 266 ist, ergibt die Aufspaltung, wie oben gezeigt, MSB = 1 UND LSB = 10. Das Register für die MSBs ist Speicherstelle 53264. Die 10 (das LSB) wird in die normale Speicherstelle für die X- Position gePOKEt. Das MSB (= 1) ist Bit 1 (weil Sprite 1) von Speicherstelle 53264. Hier ist wieder eine OR-Operation nötig. Die Y-Position wird ganz normal eingegeben. Es ergibt sich die Programmzeile:

```
110 POKE 53250,10:POKE 53264, PEEK(53264)OR(2^1):POKE 53251,130
```

Wenn Sie einen Schwarzweiß-Monitor haben, sehen Sie nach RUN ‹RETURN› trotzdem nur unser Sprite Nummer 0. Farbmonitor-Eigner bewundern schon jetzt Sprite Nummer 1. Warum, das wird uns gleich noch beschäftigen.

Welche Koordinaten eines Sprites sind eigentlich möglich, und was sieht man dann? Um das zu erforschen, bauen wir uns ein Primitiv-Sprite:

```
20 FOR I=0TO 62:POKE I+896,255:NEXT I:POKE 2042,14:REM SPRITE NR. 2
```

Dann schalten wir ein:

```
90 POKE 53269,PEEK(53269)OR(2^2)90POKE 53269,PEEK(53269)OR(2^2)
```

und bauen eine Abfrage ein für die Position:

```
130 INPUT"SPRITE 2:X,Y=";X,Y:IFX=-1 THEN END
```

Für die X-Position soll der Computer noch die Umrechnung in LSB und MSB für uns durchführen:

```
140 X1=INT(X/256):X2=X-256*X1
```

Dann POKEn wir diese Koordinaten ein:

```
150 POKE 53252,X2:POKE 53253,Y:IF X1=1 THEN POKE 53264, PEEK(53264)
OR(2^2)
```

Wenn die X-Koordinate kleiner als 256 ist, also MSB 0=0, muß sie natürlich gelöscht werden:

```
160 IF X1=0 THEN POKE 53264,PEEK(53264)AND(255-2^2)
```

Schließlich lassen wir uns außer dem Sprite Nummer 2 auch noch die eingegebenen Koordinaten zeigen und kehren zur Abfrage zurück:

```
170 PRINTCHR$(147),"X="X,"Y="Y:GOTO 130
```

Wenn Sie jetzt starten, können Sie alle möglichen Positionen für X und Y eingeben. Versuchen Sie doch mal X=265, Y=130. Da ist auch für die Schwarzweiß-Seher (ich bin auch einer) unser Sprite Nummer 1 zu erkennen. Warum, dazu kommen wir noch. Wenn Sie genug ausprobiert haben, geben Sie für X jetzt -1 und irgendeinen Y-Wert ein und das Programm ist beendet. Vermutlich haben auch Sie festgestellt, daß man unseren Testsprite ganz allmählich über den sichtbaren Bildschirmrand hinauswandern lassen kann. Es ergeben sich so die Grenzkoordinaten in Tabelle 5.2:

Sprite gerade noch voll sichtbar		Sprite gerade nicht mehr sichtbar	
links oben	rechts oben	links oben	rechts oben
X = 24 Y = 50	X = 320 Y = 50	X = 0 oder/und Y = 29	X = 344 oder/und Y = 29
links unten	rechts unten	links unten	rechts unten
X = 24 Y = 229	X = 320 Y = 229	X = 0 oder/und Y = 250	X = 344 oder/und Y = 250

Tabelle 5.2 Grenzposition normaler Sprites

Die in der Tabelle angegebenen Werte gelten allerdings nur für die normalen Sprites. Also gibt es auch noch anormale Sprites? Ja, die gibt's auch. Aber wir wollen der Reihe nach vorgehen. Wir können jetzt Sprites entwerfen, Sprite-Zeiger setzen, MOBs an- und wieder abschalten und sie an die richtige Stelle setzen.

5.4 Mal wieder Farbe: Diesmal die von Sprites.

Eines haben wir bisher total vergessen: Welche Farbe soll unser Sprite haben und wie geben wir sie ihm? Auch hierfür gibt es natürlich wieder Register, die Sprite-Farben-Register und zwar die in der Tabelle.

Sprite No.	Register
0	53287
1	53288
2	53289
.	.
.	.
.	.
7	53294

Tabelle 5.3 Zuordnung der Sprite-Farben-Register

In diese Register POKEt man die Farbe (Kennzahlen 0 bis 15) ein, in der der MOB erscheinen soll. Testen Sie mal: Ändern wir Zeile 130. Dort soll in der IF-THEN-Anweisung nach THEN anstelle von END jetzt 180 stehen. Diese Zeile 180 schaltet das vorerst ausgediente Sprite 2 aus:

```
180 POKE 53269,PEEK(53269) AND(255-2^2)
```

Damit eröffnen wir uns die Farbeingabemöglichkeit mit:

```
190 PRINT CHR$(147):INPUT"SPRITE 0,SPRITE 1 FARBEN";F1,F2:IF F1=-1
THEN 210
```

Schließlich POKEn wir die Farben in die zu den Sprites gehörigen Register:

```
200 POKE 53287,F1:POKE 53288,F2:GOTO190
210 END
```

Jetzt können Sie, bis Sie schließlich für F1 -1 eingeben, allerlei Farben durchprobieren. Wir erkennen nun auch, daß bislang Sprite 1 für Schwarzweiß-Seher nicht erkennbar war, weil seine Farbe keinen Kontrast zur Hintergrundfarbe gebildet hat.

Wem's bisher noch nicht bunt genug war, der hat auch hier bei den Sprites die Möglichkeit, den Mehrfarben-Modus zu verwenden. Während im bisher gebrauchten Modus jedes Sprite-Definitions-Bit entweder 0 (Hintergrundfarbe) oder 1 (Farbe des zum Sprite gehörenden Sprite-Farb-Registers) sein konnte, zählen - wie auch sonst im Mehrfarbenmodus - wieder Bit-Paare. Dabei stammt dann die jeweilige Farbe aus den in Tabelle 5.4 gegebenen Registern.

Bit-Paar	Farbherkunft
00	durchsichtig
01	Sprite Mehrfarbregister 53285
10	normale Sprite-Farben-Register (53287 bis 53294)
11	Sprite Merhfarbregister 52286

Tabelle 5.4 Herkunft der Bitpaar-Farben im Mehrfarben-Modus

Unser Drudenfuß sieht somit dann so aus:

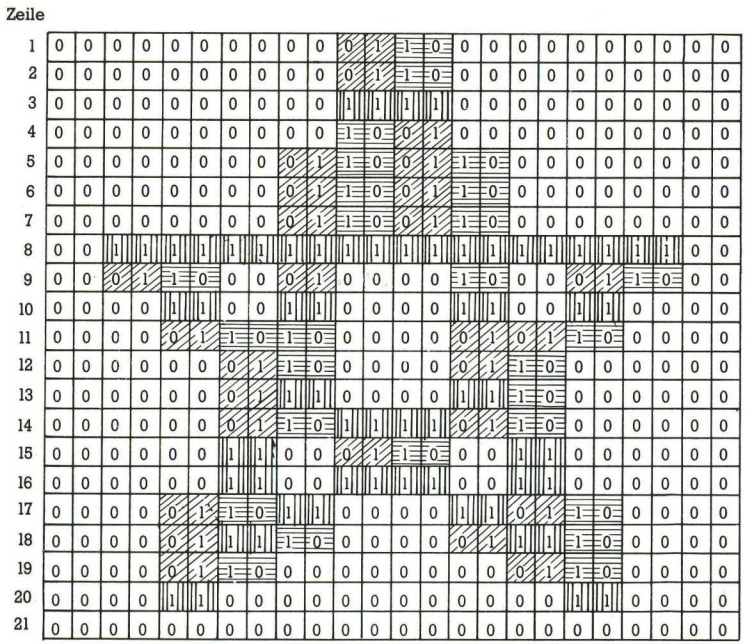


Bild 5.5 Das Pentagramm im Mehrfarben-Modus

Das wollen wir noch einmal auf dem Bildschirm ansehen. Wir schreiben ab Zeile 190 neu:

```
190 PRINTCHR$(147):INPUT"MOB1,MOB2,MULTCOL1,MULTCOL2";F1,F2,F3,F4:IF
F1=-1THEN220
200 POKE 53287,F1:POKE 53288,F2:POKE 53285,F3:POKE 53286,F4
220 END
```

So läuft natürlich noch nichts Neues, denn der Mehrfarben-Modus muß noch angeschaltet werden. Auch dazu gibt es wieder ein Register: 53276. Wie bei einigen anderen Sprite-Registern gehört auch wieder zu jedem MOB das entsprechende Bit, also zu Sprite 1 das Bit 1 und so weiter. Wenn dieses Bit auf 1 gesetzt ist, ist für das zugehörige Sprite der Mehrfarben-Modus eingeschaltet. Man kann sie also einzeln oder zusammen - ganz wie's beliebt - im Normal-Modus oder im Multicolor-Modus betrachten - Wir schalten mit der Zeile

```
210 POKE 53276,PEEK(53276)OR3:GOTO 190
```

Sprite 0 und Sprite 1 in den Mehrfarb-Modus. Will man nur Sprite Nummer N umschalten, dann verwendet man wie gehabt:

```
POKE 53276,PEEK(53276)OR(2^N).
```

Das Zurückschalten in den Normalmodus geschieht dann durch Löschen der entsprechenden Bits:

```
POKE 53276,PEEK(53276)AND(255-2^N).
```

Auch hier habe ich das Programm so gebaut, daß man durch Eingabe von -1 und drei beliebigen anderen Zahlen aussteigen kann. Im Verlauf des Probierens werden Sie bestimmt gemerkt haben, daß man Sprites, die für den Mehrfarben-Modus gedacht sind, speziell unter Berücksichtigung der Bit-Paar-Zusammenstellungen konstruieren sollte. Unser Drudenfuß gefällt mir im Normal-Modus jedenfalls besser.

Deswegen schalten wir in Zeile 220 lieber den Mehrfarben-Modus aus:

```
220 POKE 53276,PEEK(53276)AND 252
```

5.5 Anormale Sprites? Gequetschte und gezerrte MOBs

Die normalen Sprites bestehen aus 24 x 21 Bildpunkten und haben auf dem Bildschirm eine Ausdehung von zirka drei Zeilen mal drei Spalten. Für einige Effekte ist es ganz nett, sie in ihrer Größe verändern zu können. Genau das ist möglich, und zwar in X-Richtung und in Y-Richtung und das sowohl gleichzeitig, als auch unabhängig voneinander. Das Merkwürdige an dieser Sache ist - außer dem manchmal recht verzerrten Aussehen -, daß die gleichen 24 mal 21 Pixel abgebildet werden, nur daß jedes Pixel vergrößert erscheint. Wenn sowohl in X- als auch in Y-Richtung vergrößert wurde, ist einfach jeder Bildpunkt viermal so groß wie vorher. Auch hier geschieht das natürlich wieder über Kontrollregister, von denen jedes Bit wieder zu einem Sprite gleicher Nummer gehört. Die Verdoppelung in X-Richtung wird durch eine 1 im zum Sprite gehörenden Bit des Registers 53277 geschaltet, die in Y-Richtung auf die gleiche Weise im Register 53271 erreicht. Wenn wir unser Sprite Nummer 0 in X-Richtung verdoppeln wollen, müssen wir noch

```
230 POKE 53277,PEEK(53277)OR1:FOR I=0 TO 2000:NEXT I
```

eingeben. Dann sehen wir uns das nach einer kleinen Pause noch in Y-Richtung an:

```
240 POKE 53277,PEEK(53277)AND254:POKE 53271,PEEK(53271)OR1:FOR I=0 TO
2000:NEXT I
```

Jetzt vergrößern wir noch in beide Richtungen:

```
250 POKE 53277,PEEK(53277)OR 1
```

Wie Sie sich denken können, stimmt jetzt die Positionierung dieser vergrößerten Sprites auf dem sichtbaren Bildschirmteil nicht mehr. Um das zu testen, bemühen wir wieder unser Testsprite Nummer 2 und vergrößern es in beiden Richtungen:

```
120 POKE 53277,PEEK(53277)OR4:POKE53271,PEEK(53271)OR4
```

Wenn wir nun das Programm mit RUN ‹RETURN› starten, taucht unser Test-Sprite in vergrößerter Form auf und wir können es wieder an verschiedene Orte auf den Bildschirm packen. Für dieses in beide Richtungen verdoppelte MOB findet man dann die Grenzwerte in Tabelle 5.5.

Sprite gerade noch voll sichtbar		Sprite gerade nicht mehr sichtbar	
links oben	rechts oben	links oben	rechts oben
X = 24 Y = 50	X = 296 Y = 50	X nicht möglich nur: Y = 8	X = 344 oder/und Y = 8
links unten	rechts unten	links unten	rechts unten
X = 24 Y = 208	X = 926 Y = 208	X nicht möglich nur: Y = 250	X = 344 oder/und Y = 250

Tabelle 5.5 Grenzposition doppelt großer Sprites

Will man ein solchermaßen vergrößertes Sprite langsam über den Bildschirm ziehen lassen, ist das nicht nach links möglich, weil selbst bei X=0 der Sprite noch teilweise sichtbar ist.

Jetzt wissen wir eigentlich fast alles, was mit dem einzelnen Sprite zusammenhängt. Wenn Sie ein sich veränderndes Sprite darstellen wollen, ist das zum Beispiel möglich, indem alle zu zeigenden Bewegungsphasen als Spritemuster im Speicher abgelegt werden und dann per Programm der Sprite-Zeiger auf den jeweils aktuellen Bewegungszustand umgeschaltet wird. So könnte man wie in Bild 5.6 die Abläufe A bis G im Speicher ablegen und zum Beispiel den Zeiger für Sprite 3 zuerst auf das Muster A, dann nach entsprechender Verzögerung auf Muster B, dann C, D, E, F, G und schließlich wieder A richten. Wenn die einzelnen Spritemuster gut gemacht und die Verzögerungsschleifen richtig abgestimmt sind, kann so eine Art Zeichentrickfilm ablaufen, der nun auch noch durch die anderen bisher gelernten Sprite-Eigenheiten (Vergrößern, Position, Farbe etc.) veränderbar ist. Wie Sie unschwer erkennen, sind Ihrer Phantasie keine Grenzen gesetzt, und ich würde mich freuen, von Ihnen mal einen witzigen Trickablauf sehen zu können. Auf einem ähnlichen Prinzip basiert auch ein Programm von Hans Grigat in der Zeitschrift Happy-Computer, Ausgabe Nummer 11 (Jahrgang 1983), Seite 99 ff. Es lohnt sich, dieses Programm genau anzusehen, weil hier die anfangs erwähnte Möglichkeit der Sprite-Daten-Verschiebung genutzt wurde.

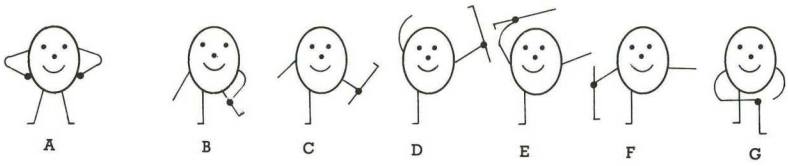


Bild 5.6 Ein Trickfilm-Bewegungsablauf

5.6 Wer hat Vorfahrt? Prioritäten

Wir wollen uns jetzt noch um die Beziehung von Sprites zu ihrer Umwelt (also zu anderen Sprites und/oder zu Zeichen auf dem Bildschirm) kümmern. Sie erinnern sich vielleicht an unseren Versuch, unser kleines Test-Sprite über den Bildschirm zu bewegen und an ein Ergebnis davon: nämlich, daß auch für Schwarzweiß-Seher plötzlich bei der Eingabe der abgedruckten Bildschirmposition das Sprite Nummer 1 sichtbar war. Wenn sich - wie in diesem Fall - zwei Sprites überdecken, welches von beiden wird dann gezeigt?

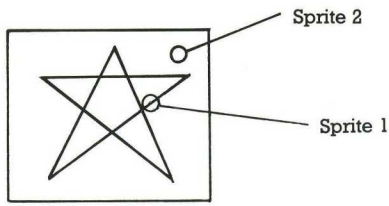


Bild 5.7 Sprite 1 überdeckt Sprite 2

Wir sehen folgendes auf diesem Bild: an den Stellen, wo Sprite 1 Bits mit dem Wert 0 vorliegen hat, ist Sprite 2 zu sehen. Wo aber der Bitwert des Sprite 1 gleich 1 ist, wird Sprite 2 durch Sprite 1 verdeckt. Das MOB 1 hat eine höhere Priorität als MOB 2. Der VIC-II-Chip organisiert die Prioritäten von Sprites untereinander also in folgender

Weise: Höchste Priorität hat Sprite 0, dann folgt Sprite 1, Sprite 2 und so weiter bis zum Schlußlicht Sprite 7.

Wenn Sie also Programme planen, in denen Sprites aneinander vorbeiziehen sollen, denken Sie daran, daß an dieser Vorfahrtsregelung nichts zu ändern ist. Unter Umständen muß man dann die Sprite-Zeiger wechseln, um die Prioritäten umzukehren: Man macht dann beispielsweise für den Augenblick der Überschneidung aus Sprite 7 zum Beispiel Sprite 1 und umgekehrt.

Eine andere Vorfahrtsregelung gilt, wenn Sprites und Bildschirm-Zeilen (oder Bit-Map-Darstellungen) aufeinandertreffen. Hier haben wir ein Kontrollregister zur Hand (mal wieder eines!), Speicherstelle 53275, wo wieder jedem Sprite ein Bit entspricht (also Bit 0 entspricht Sprite 0 und so weiter). Wenn nun dieses Bit den Wert 0 hat, steht das zugehörige MOB vor den Bildschirmdarstellungen, andernfalls verkrümelt es sich dahinter.

Das wollen wir uns mal ansehen! Starten Sie das Programm nochmal und wenn Sie an die Farbabfrage kommen, geben Sie dem Sprite 0 die Farbe 0 (schwarz). Beenden Sie das Programm in der angegebenen Weise und wenn sich READY gemeldet hat, LISTen Sie das ganze. Jetzt steht unser schwarzes Pentagramm vor dem Listing. Geben Sie nun im Direktmodus ein:

```
POKE 53276,1  <RETURN>
```

Siehe da: Sprite 0 versteckt sich hinter dem Listing. Wir können zusammenfassen: Überall dort, wo auf dem Bildschirm (durch Buchstaben oder andere Zeichen) ein Bit gesetzt ist, verschwindet dahinter ein gegebenenfalls vorhandener Sprite-Bildpunkt (wenn natürlich das zum Sprite gehörende Bit im Register 53275 gesetzt ist). Nur dort, wo auf dem Bildschirm ein Bit nicht gesetzt ist (also 0 ist), sieht man einen dort vorhandenen Sprite-Bildpunkt.

Etwas komplizierter liegen die Verhältnisse, wenn wir das MOB im Mehrfarben-Modus vorliegen haben. Hier werden nämlich auch Bildschirmbitpaare 01 (von Zeichen oder Bit-Map-Darstellungen) genauso behandelt wie Bitpaare 00. Das heißt, auch an solchen Stellen wird ein eventuell vorhandenes Sprite-Bitpaar dargestellt.

5.7 Zusammenstöße: Kollisionsregister

Erfreulicherweise haben die Software-Planer des C 64 auch zwei Möglichkeiten vorgesehen, Zusammenstöße in Registern abzufragen. Weil es zwei Sorten von Zusammen-

stößen gibt (Sprite kollidiert mit Sprite und Sprite kollidiert mit Zeichen), kann man zwei Register abfragen.

Da hätten wir zunächst das Sprite-Sprite-Kollisions-Register 53278. Auch hier gehört zu jedem Bit ein Sprite, wie bei den anderen Kontroll-Registern. Von einem Zusammenstoß spricht man in Sprite-Kreisen immer dann, wenn undurchsichtige Teile der MOBs aufeinandertreffen. Überlappen sich nur die Teile, deren Bits bei der Sprite-Definition auf 0 gesetzt wurden, zählt das nicht als Kollision. Die Bits der Sprites, die in den Zusammenstoß verwickelt sind, werden auf 1 gesetzt, zum Beispiel erzeugt ein Zusammenstoß von Sprite 0 mit Sprite 1 folgenden Inhalt des Registers 53278:

00000011 = dezimal 3.

Wenn in einem Supercrash acht Sprites kollidieren, steht im Sprite-Sprite-Kollisionsregister 255. Übrigens werden auch Zusammenstöße registriert, die außerhalb des sichtbaren Bildbereichs liegen (siehe Bild 5.4).

Wenn ein Sprite mit Bildschirmdarstellungen zusammenstößt, wird in Register 53279 das zu ihm gehörende Bit gesetzt. Stößt Sprite Nummer 1 zum Beispiel mit einem A zusammen, liest man aus dem Register 53279: 00000010 = dezimal 2. Apropos lesen: Die Register 53278 und 53279 sind so gestaltet, daß sie nach dem Herauslesen (PEEKen) gelöscht sind! Deswegen empfiehlt es sich, wenn man diese Inhalte danach noch braucht, sie in Variablen abzulegen. Auch bei dieser Sorte von Zusammenstößen zählen nur diejenigen, bei denen undurchsichtige Sprite-Teile kollidieren. Wenn übrigens Bildschirminhalte horizontal aus dem sichtbaren Bereich "herausgescrollt" worden sind (zum "Scrollen" kommen wir noch), und dabei ein Zusammenstoß mit einem Sprite geschieht, wird ebenfalls ein Bit in 53279 gesetzt.

Obwohl es zu den MOBs noch einiges zu sagen gäbe (wie kommen sie auf den Bildschirm, schnelles Steuern und so weiter), soll das Thema hiermit abgeschlossen sein. Ich denke, daß wir mit Hilfe unserer Pentagramme die kleinen Kobolde schon weitgehend entzaubern konnten. Zu den Sprites gibt es mehr Literatur als zu vielen anderen C 64-Themen. Hier eine kleine Auswahl:

- Zwei Artikel wurden schon genannt (von H. Grigat und H. Kunz)
- Herbert Kunz hat auch schon etwas über schnelles Bewegen von MOBs geschrieben (im 64'er, Ausgabe 4/83 auf Seite 70).
- Einen Überblick vor allem über die Anwendung von Sprites in Spielen geben Schneider und Eberl in den Bänden 1 und 3 des Commodore 64-Buches. Erschienen 1984 in Haar bei München im Markt & Technik Verlag. Diese beiden Bände sind auch von den sehr gut überschaubaren Programmbeispielen her zu empfehlen.

6

Zwei- und dreidimensionale Darstellung

6 Zwei- und dreidimensionale Darstellung

6.1 Wir schaffen es, nichts zu sehen: Abschalten des Bildschirms

Wenn Sie an Ihrem Computer eine Kassettenstation verwenden, kennen Sie das zur Genüge: Sie geben irgendeinen Kassettenbefehl ein, der Computer fordert zum Beispiel PRESS RECORD & PLAY. Sobald Sie dieser Aufforderung nachgekommen sind, ist der Bildschirm total leer. Dieses üble Verhalten - wir sehen ja nun einige Meldungen nicht mehr - hat seinen Grund. Die Kassettenoperationen können etwas schneller laufen. Überhaupt beschleunigt ein abgeschalteter Bildschirm den Programmablauf. Im allgemeinen nützt uns das nicht viel, denn wir reden ja mit unserem Computer per Bildschirm und wenn der Gesprächspartner einen "black-out" hat, kann nicht miteinander verkehrt werden. Manchmal gibt es aber Situationen, wo dieses Abschalten für kurze Zeit ganz angenehm sein kann. Wenn wir zum Beispiel ein kompliziertes Titelbild aufbauen, mögen viele nicht zuschauen. Sondern nach einem leeren Bildschirm soll sofort das fertige Bild auftauchen.

Was passiert beim Abschalten des Bildschirms? Das ist optisch eigentlich gar nicht so aufregend, denn der ganze Bildschirm nimmt die Farbe des Rahmens ein. Er wird genau genommen nicht abgeschaltet. Geschaltet wird etwas, das wir nicht sehen können. Nämlich eine ganz bestimmte Fähigkeit unseres Prozessors, auf den Systembus anders als im Normalbetrieb zuzugreifen. Diese Art des Zugriffs birgt eine versteckte, mögliche Fehlerquelle in sich: Wenn während Kassettenoperationen Sprites angeschaltet bleiben, können Störungen beim Zugriff auf Daten auftreten. Betrachten Sie dazu Tabelle 1.1. In Register 17 (Adresse 53265) dient das Bit 4 als Schaltbild für Bildschirm "aus" oder "ein". Wenn wir dieses Bit auf 0 setzen, erblindet der Monitor. Dieses Register 17 ist ein ziemlich komplexes Ding. Wir müssen darauf achten, daß nur Bit 4 gesetzt oder gelöscht wird. Ich hoffe, daß Sie noch wissen, wie das ging. Zur Erinnerung:

Normaler Inhalt von 53265: XXX1 XXXX

AND 239: 1110 1111

Jetzt ist Bit 4 gelöscht: XXX0 XXXX

Das entspricht dem Befehl: POKE 53265,PEEK (53265)AND239

Nun noch zum Wiedereinschalten des Bildschirmes:

Mit gelöschtem Bit 4: XXX0 XXXX

OR 16: 0001 0000

Normaler Inhalt wieder hergestellt: XXX1 XXXX

In Basic ist das dann: POKE53265,PEEK(53265)OR16.

Zum Ausprobieren ein kleiner Dreizeiler:

```
10 POKE 198,0:WAIT 198,1:POKE 53265,PEEK(53265)AND 239
15 PRINT CHR$(147):POKE 211,15:POKE214,10:SYS58640:PRINT"HALLO"
20 POKE 198,0:WAIT 198,1:POKE 53265,PEEK(53265)OR16
```

Nach RUN wartet das Programm auf einen Tastendruck, auf den hin dann der Bildschirm "ausgeschaltet" wird. Ein weiterer Tastendruck macht das Bild wieder sichtbar.

Nebenbei bmerkt: Falls Sie einige POKEs und das SYS-Kommando nicht kennen, hier die Erklärung: 198 ist das Byte, in dem die Anzahl gedrückter Tasten gespeichert wird. Deswegen kann man mit WAIT 198,1 und vorher erfolgtem Nullsetzen dieses Bytes dasselbe erreichen, wie mit der häufig benutzten GET-Abfrage. SYS 58640 ist ein Einsprung in die Betriebssystemroutine, die den Cursor setzt. Die gewünschte Spalte wird in Byte 211, die Zeile in 214 eingePOKEt. Damit erspart man sich den Wust an Cursorsteuerzeichen, die dem Drucker Kopfschmerzen bereiten können.

6.2 Das sogenannte Scrollen: Bildverschiebung

Wir kennen jetzt fast alle Register des VIC-II-Chips. Zwei Bytes, nämlich 53265 und 53270, bergen noch Geheimnisse, die wir nun lüften wollen. Zunächst die Bits 3. Sie enthalten normalerweise den Wert 1. Jetzt ändern wir das mal in 53265:

```
POKE 53265,PEEK(53265)AND247 <RETURN>
```

Haben Sie's bemerkt? Wenn nicht, zählen Sie mal die Zeilen auf dem Bildschirm nach: Es sind nur noch 24 zu sehen. Rückgängig können Sie das durch Setzen des Bits 3 wieder machen:

```
POKE53265,PEEK(53265)OR8 <RETURN>
```

Probieren wir dasselbe mit der Speicherstelle 53270:

```
POKE53270,PEEK(53270)AND247 <RETURN>
```

Unsere Kommandos und auch der Cursor sehen jetzt etwas merkwürdig aus. Wir haben nicht mehr 40 Spalten, sondern nur noch 38 zur Verfügung. Zurück in den Normalzustand:

```
POKE53270,PEEK(53270)OR8 <RETURN>
```

Beim Eingeben dieser Zeile werden Sie bemerkt haben, daß trotzdem noch der ganze 40-Zeichen-Bereich zur Verfügung steht, nur sind die erste und die letzte Spalte verdeckt. Der Rahmen ist breiter geworden und versteckt gewissermaßen den Rand des Bildschirms.

Nun kommen wir zum Scrollen. Das englische Wort "scroll" heißt auf Deutsch etwa Rolle oder Schriftrolle. Wenn wir ein längeres Programm listen oder mit dem Cursor an den unteren Bildrand fahren, rollt der C 64 den Bildschirminhalt nach oben. Das ist ein zeilenweises Scrollen. Der VIC-II-Chip gestattet aber auch eine andere Art des Scrollens, die im englischen als "smooth scrolling", etwa "fließendes" Scrollen bezeichnet wird. Wie Sie sich sicherlich erinnern, ist jedes Zeichen aus 8 Bytes zusammengesetzt. Während beim normalen Scrollen alle 8 Bytes (also ein Zeichen) auf einmal nach oben springen, geht das beim fließenden Scrollen Byte für Byte. Dazu dienen die Bits 0 bis 2 der Register 53265 und 53270. Sehen wir uns das mal in Schritt für Schritt an. Der Maximalwert in den 3 Bits (0,1,2) kann 7 sein (binär 111). Alle anderen Bits müssen geschützt sein. Normalerweise steht in 53265 in den Bits 0 bis 2 der Wert 3 und an derselben Stelle in 53270 der Wert 0. Folgendes Kurzprogramm soll diese Werte mit jedem Tastendruck verändern:

```
5 PRINT CHR$(147)CHR$(166)
10 FOR I = 0 TO 7
20 POKE 53265,(PEEK(53265)AND248)OR I
30 GET A$:IF A$=""THEN 30
40 NEXT I
50 POKE 53265,(PEEK(53265)AND248)OR3
```

Wenn Sie dieses Programm starten, wandert das Grafikzeichen bei jedem Tastendruck Byte für Byte herunter. Wenn Sie jetzt noch einfügen:

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

Das war das Einführen des 24-Zeilen-Modus, dann ist das Grafikzeichen zu Anfang fast versteckt. Wir sollten am Programmende noch den Normalzustand wiederherstellen :

```
60 POKE 53265,PEEK(53265)OR8
```

Auf diese Weise geschieht das fließende Scrollen nach unten. Verändern wir Zeile 10:

```
10 FOR I = 7 TO 0 STEP-1
```

erfolgt Scrollen nach oben. Das gleiche Programm können wir mit kleinen Änderungen für horizontales Scrollen verwenden. Wir müssen nur überall dort, wo 53265 steht, jetzt 53270 und in Zeile 50 anstelle von OR 3 jetzt OR 0 einsetzen. Wenn Sie Zeile 10 mit der Rückwärtsschleife belassen haben, scrollt der Bildschirminhalt nach links. Ändern Sie dagegen Zeile 10 wieder zur ursprünglichen Vorwärtsschleife, erfolgt ein Scrollen nach rechts. Sie werden jetzt sagen, daß das alles ja ganz nett ist, aber was kann man damit tun? Damit kann man Laufschriften erzeugen, die in beliebiger Richtung über den Bildschirm sausen oder schleichen - je nachdem. Leider, leider kommen wir auch hier langsam aber sicher an die Grenzen von Basic. Denn jetzt taucht ein Problem auf, dessen Lösung - in Basic als auch in Maschinensprache - gleich ein zweites nach sich zieht.

In dem Moment, wo man das Zeichen geduldig Byte für Byte aus dem Versteck herausgeholt hat, also 7 in die Bits 0 bis 2 von 53265 oder 53270 eingegeben hat, muß man dafür sorgen, daß

1) der Bildschirminhalt um ein Zeichen weitergeschoben wird,
2) im Versteck (am Rand) ein nächstes Zeichen parat liegt.

Dazu können wir uns wie im nachfolgenden Programm zum Beispiel des ohnehin schon eingebauten Zeilen-Scrolls bedienen:

```
10 POKE 53265,PEEK(53265)AND 247
20 PRINTCHR$(147):FOR I=1 TO24:PRINTCHR$(17);:NEXT I
30 POKE 53265,(PEEK(53265)AND248)OR7
40 PRINT"SCROLLING NACH OBEN"
50 FOR I=6 TO 0 STEP-1
60 POKE 53265,(PEEK(53265)AND248)OR I:FOR J=1TO 50:NEXT J
70 NEXT I:GOTO30
```

Wenn Sie dieses Programm gestartet haben, können Sie mit ‹RUN/STOP› und ‹RESTORE› den Normalzustand wiederherstellen. Haben Sie etwas bemerkt? Ein Flackern des Bildes trat auf. Das ist das zweite Problem, auf das wir gleich kommen werden.

Ein Beispiel für das horizontale Scrollen gibt das Programm von H. Gehrmann in der Zeitschrift Happy-Computer, Ausgabe 4/84, Seite 108f. H. Gehrmann hat es geschafft, in Basic tatsächlich fließendes Scrollen ohne Bildschirmflackern zu erzeugen. In Zeile 36 seines Programms ist ein Teil seiner Lösung für Problem Nummer 2 zu finden. Dort steht:

```
36 IF PEEK(53265) <>165 THEN 36
```

Erst danach druckt er an den oberen Bildschirmrand die zu scrollende Textzeile und scrollt dann, wie wir es bisher kennengelernt haben. Was ist denn nun dieses zweite Problem? Sehen wir dazu zunächst einmal in die Speicherstelle 53265 hinein. Geben Sie (nach NEW) die folgende Textzeile ein:

```
10 PRINT PEEK(53265):GOTO10
```

Nach RUN sehen Sie einige Male den Wert 27 und ab und zu taucht auch der Wert 155 auf. Wenn Sie jetzt mit ‹RUN/STOP› und ‹RESTORE› dieses Testprogramm angehalten haben, sehen wir uns diese Zahlen mal binär an:

```
27   = binär 0001 1011
155  = binär 1001 1011
```

Alle Bits außer Bit 7, welches sich verändert hat, kennen wir schon . Es handelt sich um das msb des Rasterzeilenregisters. Das lsb davon findet sich in Speicherzelle 53266. Wir haben es hier mit einer 9stelligen Binärzahl zu tun, von der 8 Stellen (die Bits 0 bis 7) in 53266 und die 9. Stelle als Bit 7 in 53265 zu finden sind. Gezählt werden ständig in diesen neun Bits die Rasterzeilen. Sie werden sich fragen, was das soll, hier ist die Antwort:

An der Entstehung eines Fernsehbilds ist ein Elektronenstrahl beteiligt. Er schreibt Punktzeile für Punktzeile von oben nach unten ein Raster auf den Bildschirm und erzeugt dabei das Bild. 50mal in der Sekunde baut er ein neues Bild auf, und in welcher Punktzeile (Rasterzeile) er gerade ist, das zählt der Computer mit in diesem Rasterzeilenregister. Er muß deshalb mitzählen, weil er dem Elektronenstrahl einige Informationen übergeben muß, nämlich ob ein Punkt sichtbar sein soll oder nicht. Der Zeilenstrahl überstreicht den gesamten Bildschirm, also auch Bereiche, die außerhalb des lesbaren Textfensters liegen. Deswegen müssen wir auch weiter als bis 255 Rasterzeilen zählen. Weil aber in ein Byte (hier nun 53266) nur bis 255 gezählt werden kann, kommt noch das Bit 7 von 53265 als msb dazu.

Jedesmal, wenn der Zeilenstrahl über die 255. Rasterzeile hinaushuscht, wird dieses Bit gleich 1. Das Problem Nummer zwei läßt sich also jetzt so ausdrücken: Wenn eine Bildänderung stattfindet, während der Zeilenstrahl über die Änderungsstelle hinwegstreicht, kommt es zu Zuckungen des Bildes. Man muß dafür sorgen, daß man diesem Flitzer nicht ins Gehege kommt. H. Gehrmann hat das in seinem Programm so gemanagt, daß er den Bildinhalt verändert, während der Rasterstrahl außerhalb des sichtbaren Bereiches arbeitet. Trotzdem gibt es hier aber noch Schwierigkeiten. Die Bildverarbeitung muß abgeschlossen sein, bis der Elektronenstrahl wieder im sichtbaren Bereich ist.

Normalerweise ist das eine Aufgabe für Maschinenprogramme, von denen Sie einige in der nachfolgend genannten Literatur finden:

- R. Babel, M. Krause, A. Dripke: Das Interface Age Systemhandbuch zum Commodore 64, München, 1983, S. 63-64.
- A. Plenge: Das Grafikbuch zum Commodore 64, Düsseldorf 1984, S. 272ff.

6.3 Dornröschen ist wieder dran: 2D-Grafik besser

Von nun an kümmern wir uns wieder um Dornröschen, die hochauflösende Grafik. Wie Sie wohl spätestens in Kapitel 4 bemerkt haben, ist das Koordinatensystem, das unser Commodore 64 auf dem Bildschirm einrichtet, reichlich unüblich. Zur Erinnerung sehen Sie sich mal das Bild an:

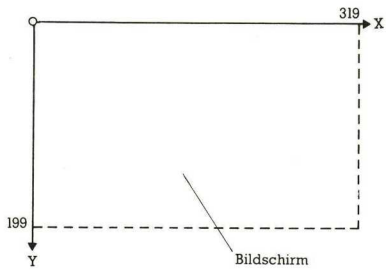


Bild 6.1 Das Bildschirmkoordinatensystem

Wie Sie aus der Schule vielleicht noch wissen, ist die normale Darstellung eines Koordinatensystems aber so wie in Bild 6.2 gezeigt.

Im Programm 3 mußte deshalb eine einfache Sinus-Funktion so verändert werden, daß man sie von der Formel her kaum mehr erkennen kann (Zeile 5). Bleiben wir beim Beispiel der Sinus-Kurve. Die sieht so aus wie in Bild 6.3 gezeigt.

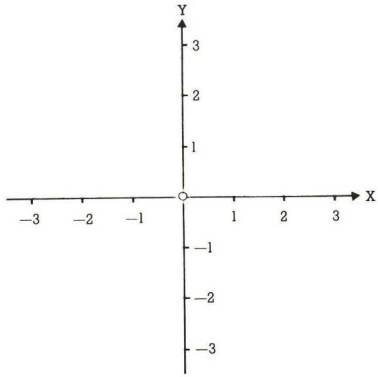


Bild 6.2 Das gewohnte Koordinatensystem

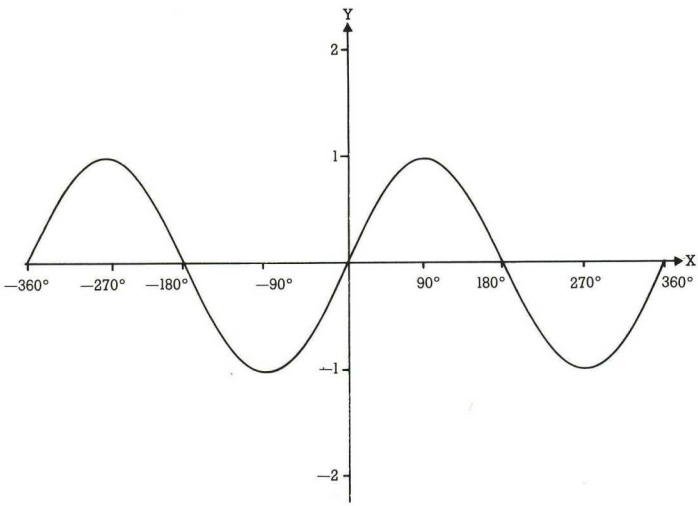


Bild 6.3 Eine normale Sinuskurve

Wie groß oder klein X also auch immer wird, Y bewegt sich nur zwischen +1 und -1 hin und her. Wenn wir die normale Sinus-Gleichung (Y=SIN(X)) auf unserem Bildschirmsystem zeichnen lassen, werden wir gar nichts oder kaum etwas sehen.

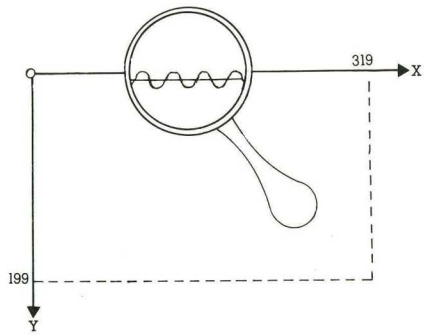


Bild 6.4 Die normale Sinuskurve im Bildschirmkoordinatensystem

Wir müssen also einen Weg finden, ein uns genehmes Koordinatensystem anstelle des Bildschirmsystems zu verwenden. Dazu soll uns die Mathematik verhelfen. Man nennt das, was wir jetzt tun werden, eine Transformation! Keine Angst, es wird nicht zu schwierig! Bild 6.5 zeigt zwei Koordinatensysteme.

1) Das Bildschirmsystem X,Y mit Nullpunkt 0
2) Das von uns gewünschte System X',Y', mit Nullpunkt 0'.

Außerdem ist der Punkt P1 zu sehen. Je nach dem, auf welches Koordinatensystem er bezogen wird, hat er die Koordinaten

1) X1,Y1 oder
2) X1',Y1'.

Unser eigenes System soll vom unteren Wert XU bis zum oberen X-Wert XO reichen und vom unteren Y-Wert YU bis zum oberen YO. Am Beispiel unserer Sinuskurve wäre es sinnvoll, die X-Werte von XU = -1 bis XO = 10 und die Y-Werte von YU = -2 bis YO = +2 abzubilden. Die Entfernung von XU bis XO (also XO-XU) erstreckt sich auf 319 Bildpunkte. Man kann nun ein Verhältnis bilden, denn X1 muß sich zu (X1'-XU) (das ist die Länge der Strecke von XU bis X1') genauso verhalten wie 319 zur Gesamtentfernung (XO-XU).

Als Formel sieht das so aus:

$$\frac{X1}{X1' - XU} = \frac{319}{XO - XU}$$

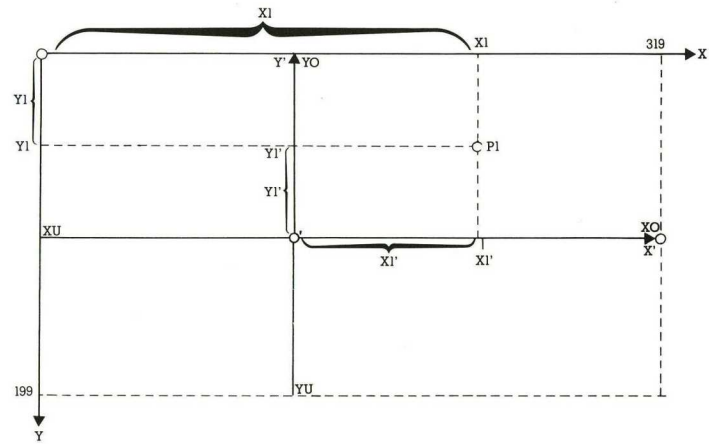


Bild 6.5 Transformation

Das einzige in dieser Formel, was wir nicht kennen (X1' ist ja unsere Eingabe, die auf unser eigenes Koordinatensystem bezogen ist) ist nun X1, also die X-Koordinate unseres Punktes P1 im Bildschirmsystem. Man kann diese Verhältnisgleichung nach X1 auflösen:

```
X1=(X1' - XU)/(XO - XU)*319
```

So ähnlich geht das auch bei der Y-Koordinate. Es ist nur zu beachten, daß im Bildschirmsystem die Y-Achse nach unten, in unserem System aber nach oben zeigt. Deswegen ist die Entfernung von YO nach Y1':YO-Y1'. Das Verhältnis hat hier die Formel:

$$\frac{Y1}{YO - Y1'} = \frac{199}{YO - YU}$$

Nach Y1 aufgelöst ergibt sich:

```
Y1=(YO-Y1')/(YO-YU)*199
```

Es empfiehlt sich, diese beiden sogenannten Transformationsgleichungen als Basicfunktionen zu definieren:

```
DEFFNX(X)=(X-XU)/(XP-XU)*319
DEFFNY(Y)=(YO-Y)/(YO-YU)*199
```

Nun können wir ein kleines Aufrufprogramm schreiben, welches sich wieder der Unterprogrammsammlung aus Programm 2 bedient. Laden Sie zuerst die Unterprogramme und tippen Sie dann ein:

```
5 POKE 52,92:POKE 56,92
100 DEFFNX(X)=(X-XU)/(XO-XU)*319
110 DEFFNY(Y)=(YO-Y)/(YO-YU)*199
120 REM + + + + + BEISPIELFUNKTION + + + + +
140 DEFFNA(X)=SIN(X)
150 PRINTCHR$(147)CHR$(17):INPUT"XU,XO,YU,YO=";XU,XO,YU,YO
160 INPUT"ZEICHEN- UND HINTERGRUNDFARBE=";F1,F2
170 GOSUB50100:FOR I=XU TO XO STEP(XO-XU)/319
180 X=FNX(I):Y=FNY(FNA(I)):GOSUB 50040
190 NEXT I:X1=FNX(XU):Y1=FNY(0):X2=FNX(XO):Y2=Y1: GOSUB 50060
195 X1=FNX(0):Y1=FNY(YU):X2=X1:Y2=FNY(YO):GOSUB 50060
200 GET A$:IF A$=""THEN 200
210 GOSUB 50030
220 END
```

Nach RUN und mit der erforderlichen Geduld (das kennen Sie ja schon) zeichnet der Computer eine ganz normale Sinuskurve in das von Ihnen durch XU,XO,YU und YO angegebene Koordinatensystem. In die Zeile 140 können Sie jetzt jede beliebige Funktion eingeben. Allerdngs ist es unglücklich, daß man jedesmal das Programm ändern muß, nämlich Zeile 140, wenn man eine neue Funktion sehen will. Es gibt da einen Trick, der mit dem Tastaturpuffer arbeitet (Speicherstellen 631-640) und den ich Ihnen nun zeigen möchte. Zunächst fügen wir noch eine Zeile 130 ein:

```
130 F$="SIN(X)"
```

Wenn das Programm startet, soll man zunächst einmal erfahren, welche Funktion überhaupt in Zeile 140 steht. Deswegen also:

```
10 PRINTCHR$(147):POKE211,5:POKE214,10:SYS58640:PRINT"FUNKTION IM
PROGRAMM:"
20 PRINT:PRINTTAB(3)"Y="F$
```

Wenn wir das so laufen lassen, wird keine Funktion erscheinen, weil diese dem Computer erst in Zeile 130 mitgeteilt wird. Der Computer muß erst vorher nachsehen, als was F$ definiert ist. Deswegen führen wir ein:

```
15 K=1:GOSUB 130:K=0
```

und

```
145 IF K=1 THEN RETURN
```

Jetzt druckt das Programm uns die Funktion aus. Als nächstes soll der Computer erfahren, ob die Funktion zu verändern ist:

```
30 PRINT:PRINTTAB(5)CHR$(18)"A"CHR$(146)"LTE ODER"
CHR$(18)"N"CHR$(146)"EUE FUINKTION?
35 GET A$:IFA$<>"A"AND A$<>"N"THEN 35
```

Wenn die alte Funktion gewählt wurde, dann kann alles wie bisher ablaufen:

```
40 IF A$="A"THEN 100
```

Ansonsten wollen wir jetzt unseren Trick anwenden. Zunächst das Eingeben der neuen Funktion:

```
45 PRINT:PRINTTAB(3);:INPUT"Y=";F$
```

Dann

```
50 PRINTCHR$(147)CHR$(17)CHR$(17)"130F$="CHR$(34)F$CHR$(34)
55 PRINT"140DEFFNA(X)="F$
60 PRINT"RUN 100":PRINTCHR$(19);
65 POKE 631,13:POKE 632,13:POKE 633,13:POKE 198,3:END
```

Tippen Sie's ein und fügen Sie in Zeile 100 STOP ein, damit Sie sehen können, was mit diesen Zeilen passiert. Sie sehen, daß der Bildschirm zunächst frei gemacht wird (CHR$(147) in Zeile 50) und dann zwei Programmzeilen 130 und 140 mit der neuen Funktion gedruckt werden , danach noch die RUN-Anweisung. Der Cursor springt dann wieder in die Ecke links oben (CHR$(19) in Zeile 60). Jetzt arbeitet unser Computer den Inhalt des Tastaturpuffers ab, der aus drei RETURNs (die 13) besteht. Die Anzahl 3 muß in speicherstelle 198 festgehalten sein. Die Tatsache, daß der Tastaturpuffer abgearbeitet wird, haben wir dem END zu verdanken. Weil aber ein RETURN nach RUN 100 folgt, startet das Programm sofort wieder selbtändig ab Zeile 100. Wenn Sie jetzt mal Zeile 130 und 140 listen lassen, steht dort die neue Funktion. Weil sie sich im Programmablauf störend auswirken, sollen die zu druckenden Zeilen unsichtbar werden.

Deswegen fügen wir in Zeile 45 noch an:

```
:POKE 646,6
```

und ab Zeile 100 soll wieder sichtbar werden, was zu drucken ist, weshalb wir dort anfügen:

```
:PRINT CHR$(147):POKE 646,14
```

Dieser Trick ist sehr vielseitig anwendbar. Man muß nur beachten, daß durch das RUN alle Variablen, die bis dahin definiert wurden, gelöscht werden. Das ist ja auch sinnvoll, weil wir das Basicprogramm verändert haben und so unter Umständen Variable überschrieben wurden. Sollte es nicht möglich sein, alle Variablen nach dem Neustart zu definieren, läßt sich ein ähnlicher Kniff anwenden, wie wir ihn in Zeile 15 verwendet haben. Probieren Sie das Programm mal mit folgenden Eingaben:

```
Y=SIN(X)                    mit XU=-1,XO=10,YU=-2,YO=2

oder

Y=EXP(-X/10)*COS(X)         mit XU=-18,XO=10,YU=-5,YO=5

oder

Y=2*SIN(X)+SIN(2*X+2)       mit XU=-6,XO=15,YU=-3,YO=3
```

Ihrer Phantasie sind keine Grenzen gesetzt. Das komplette 2D-Programm liegt als Programm 4 auf der Diskette vor.

6.4 Flacher Raum: 3D-Grafik, wie funktioniert das?

Nun sind wir bei dreidimensionalen, also räumlichen Abbildungen angekommen. Zunächst einmal: Es ist klar, weil der Fernsehschirm eben nur eine Fläche wie ein Blatt Zeichenpapier ist, daß wir etwas Räumliches auf einer Fläche darstellen müssen. Alles andere ist Science fiction, auch für die größten Computer! Bei 3D-Darstellungen unterscheidet man hauptsächlich zwei Arten:

a) Zeichnungen, die mit einer Spezialbrille räumlich wahrgenommen werden können,

b) Zeichnungen, die perspektivisch gestaltet sind und deswegen als räumlich empfunden werden.

Im Prinzip ist der Weg a) für uns begehbar. Wir könnten die dazu nötigen zwei verschiedenfarbigen Zeichnungen (meist rot und blau) mit Hilfe des Mehrfarben-Bitmap-Modus erzeugen. Allerdings ist der rechnerische Aufwand nicht unerheblich. Ab und zu werden in Programmzeitschriften entsprechende Entwürfe veröffentlicht.

Der übliche Weg allerdings geht über eine perspektivische Darstellung. Zwei Arten wiederum finden vor allem Anwendung:

a) Die sogenannte Kavaliers-Perspektive
b) Die Perspektive mit mindestens einem Fluchtpunkt.

Den Unterschied soll Bild 6.6 erläutern.

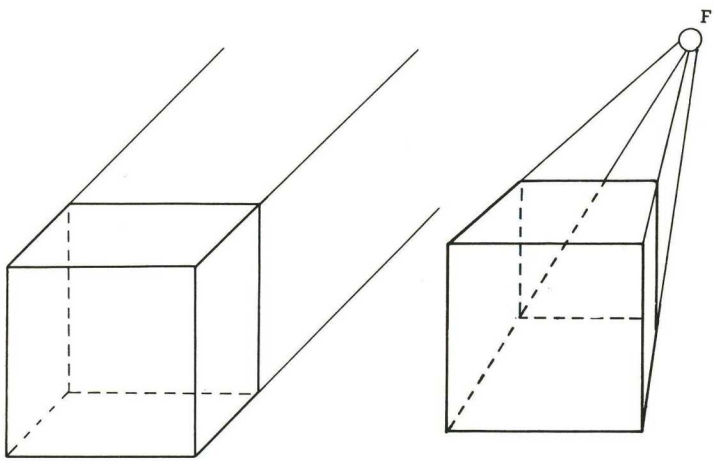


Bild 6.6 Zwei verschiedene Perspektiven

Bei der Kavaliers-Perspektive bleiben alle Linien, die auch in Wirklichkeit parallel verlaufen, parallel, während sie im anderen Fall in einem Fluchtpunkt zusammenlaufen, wenn sie in die räumliche Tiefe weisen. Gemeinsam ist beiden Verfahren, daß die Linien, die in den Raum verlaufen (also nach hinten oder vorne) verkürzt werden, weil sich unsere Augen sonst über die Größenverhältnisse täuschen lassen. Zwar sieht die Fluchtpunktperspektive sehr viel natürlicher aus, aber die Kavaliers-Perspektive ist mathematisch bei weitem nicht so kompliziert. Deswegen wollen wir uns mit ihr befassen. Weiterhin schränken wir uns noch ein: Der Winkel, unter dem die Linien in den Raum laufen, soll 45° betragen, also ein halber rechter Winkel sein. In diesem Fall verkürzen sich die nach vorne oder hinten verlaufenden Kanten auf die Hälfte. Genauso, wie man einen Punkt durch seine Koordinaten im ebenen System festlegen kann, ist er auch in einem räumlichen Koordinatensystem definierbar. Man baut einfach noch eine dritte Achse (eine in den Raum weisende Z-Achse) zu den vorhandenen dazu und erhält so Zusammenhänge wie in Bild 6.7.

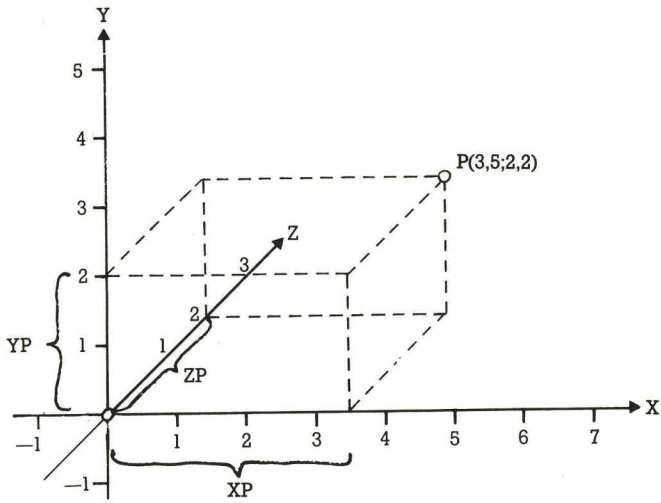


Bild 6.7 Ein Punkt im räumlichen Koordinatensystem

Das Verfahren soll durch diese Erklärung noch etwas verdeutlicht werden. Man kann sich den Raum als eine große Anzahl dicht an dicht hintereinander gestapelter X-Y-Ebenen vorstellen, etwa wie in Bild 6.8.

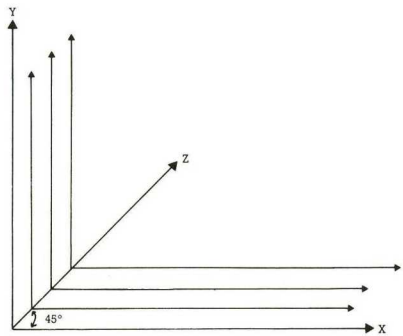


Bild 6.8 Hintereinander gestapelte X-Y-Ebenen

Im Bild sind sie, damit man das erkennen kann, mit Lücken gezeichnet. Auf einer Fläche zeichenbar wird das Verfahren dann, wenn man jede X-Y-Ebene um den Betrag ihrer Raumtiefe (das entspricht der halben Z-Koordinate) entlang der Achse Z nach rechts oben verschiebt wie in Bild 6.9.

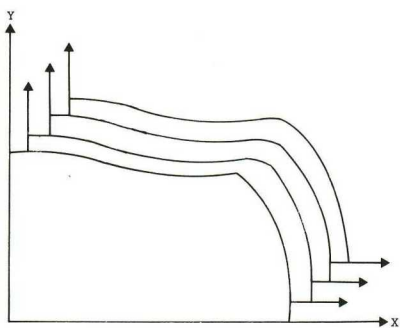


Bild 6.9 Parallelverschobene X-Y-Systeme

Nehmen wir nun noch an, daß auf jeder Ebene eine Kurve gezeichnet sei, die sich von Ebene zu Ebene etwas verändert, dann bekommen wir auf diese Weise aus allen Kurven zusammen den Eindruck einer räumlichen Fläche wie in Bild 6.10.

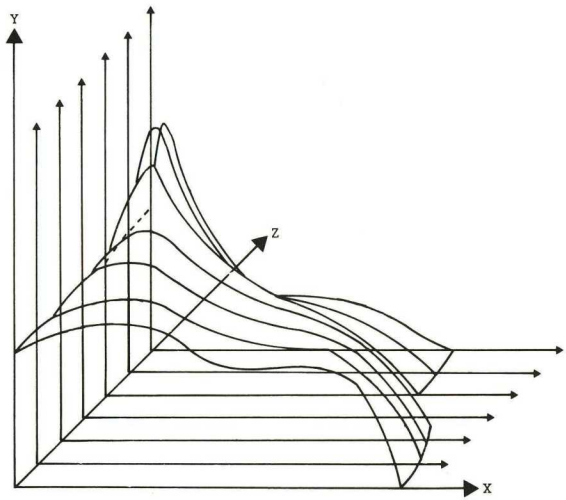


Bild 6.10 Zusammensetzung von Raumflächen aus ebenen Kurven

Das Prinzip ist jetzt wohl klar. Wenn wir nun bedenken, daß die Raumfläche aus lauter ebenen Kurven und die Kurven aus lauter Punkten zusammengesetzt sind, dann stellt sich für uns die Frage, wie man einen Punkt P(X1',Y1',Z1') an der richtigen Stelle unseres Bildschirmes darstellen kann. Weil wir schon wissen (von den 2D-Zeichnungen her), wie wir unser gewünschtes Koordinatensystem auf den Bildschirm bringen können, stellt sich die Frage für uns einfacher. Wir müssen uns nur noch überlegen, wie man mit zwei Koordinaten X1,Y1 den räumlichen Punkt in unserem selbstgewählten System zeichnen kann. Sehen wir uns dazu Bild 6.11 an:

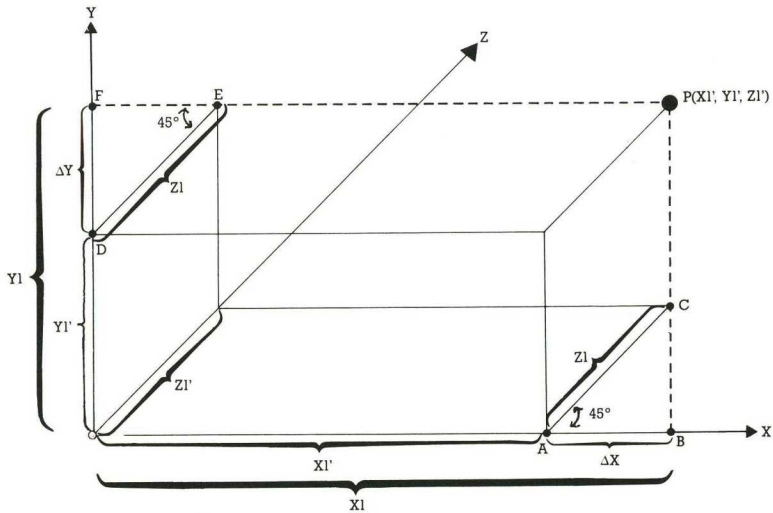


Bild 6.11 Transformation von 3D nach 2D

Wir haben hier einfach so getan, als gäbe es die Z-Achse gar nicht, sondern der Punkt P wäre einfach um einen Wert Δ X nach rechts und einen weiteren Wert Δ Y nach oben geschoben worden. Die Koordinaten von P im ebenen System sind dann

$$X1 = X1' + \Delta X$$
$$Y1 = Y1' + \Delta Y$$

wie man auch aus der Zeichnung sehen kann. Wie lang ΔX ist, kann man aus dem Dreieck ABC berechnen. Mathematisch Versierte werden mir zustimmen, daß

$$COS(\alpha) = \Delta X / Z1$$

ist und im Dreieck DEF gilt

$$SIN(\alpha) = \Delta Y / Z1$$

Daraus ergibt sich dann: $\Delta X = Z1*COS(\alpha)$
$\Delta Y = Z1*SIN(\alpha)$

Weil außerdem noch nach unserer anfänglichen Vereinbarung der Winkel $\alpha = 45°$ beträgt und die Z-Achse auf die Hälfte verkürzt ist und weil schließlich

$$SIN(45°) = COS(45°) = 1/\sqrt{2}$$

ist, ergibt sich: $\Delta X = Z1'/(2*SQR(2))$
$\Delta Y = Z1'/(2*SQR(2))$

Beide sind also gleich und für unser eigenes Koordinatensystem hat der Punkt P die Koordinaten:

$X1 = X1' + Z1'/(2*SQR(2))$
$Y1 = Y1' + Z1'/(2*SQR(2))$

Damit ist unser Problem gelöst. Denn X1',Y1' und Z1' sind die vorgegebenen Koordinaten und X1,Y1 können jetzt mit unseren Transformationen X=FNX(X1),Y=FNY(Y1) in Bildschirmkoordinaten umgerechnet werden. Wir verschachteln nun zwei Schleifen ineinander wie in Bild 6.12 und lassen uns damit Kurve für Kurve zeichnen.

Nun wollen wir auch für die 3D-Grafik ein Programm entwerfen. Tippen Sie NEW ein, laden Sie die Grafik-Unterprogramme und erwecken Sie unser 3D-Dornröschen mit dem nachfolgenden Aufrufprogramm.

```
5 POKE 52,92:POKE 56,92
100 DEFFNX(X)=(X-XU)/(XO-XU)*319
105 DEFFNY(Y)=(YO-Y)/(YO-YU)*199
110 DEFFNZ(Z)=Z/(2*SQR(2))
140 DEFFNA(X)=SIN(Z)*COS(X)
150 PRINTCHR$(147)CHR$(17)CHR$(18)"UNSER SYSTEM:"CHR$(146):PRINT:
PRINT
160 INPUT"XU,XO=";XU,XO:INPUT"YU,YO=";YU,YO
```

Jetzt wird es ein bißchen kompliziert, weil wir darauf achten müssen, daß die Z-Achse noch auf den Bildschirm paßt. Wir lassen uns berechnen und anzeigen, wie groß ZO höchstens sein darf und wie klein ZU:

```
162 Z3=2*XO*SQR(2):Z4=2*YO*SQR(2):IF Z3<Z4 THEN PRINT "ZO MAXIMAL = "
Z/3: GOTO 166
164 PRINT"ZO MAXIMAL ="Z4
166 Z5=2*XU*SQR(2):Z6=2*YU*SQR(2):IF Z5>Z6 THEN PRINT "ZU MINIMAL ="
Z5: GOTO 170
168 PRINT"ZU MINIMAL=";Z6
170 PRINT:INPUT"ZU,ZO=";ZU,ZO:Z1=FNZ(ZO):Z2=FNZ(ZU)
```

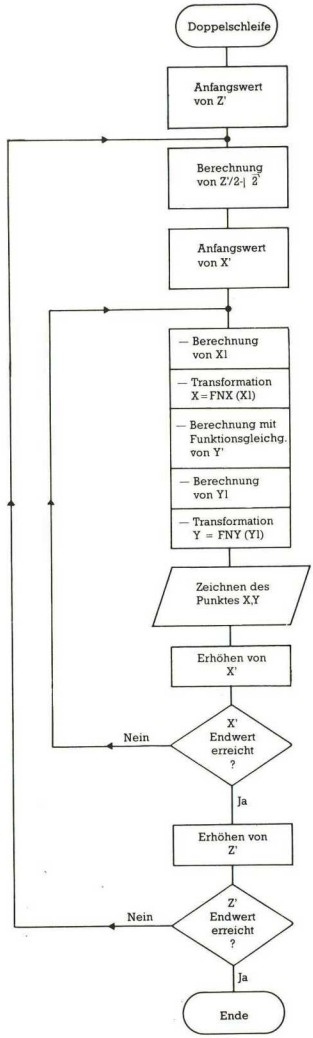


Bild 6.12 Das Flußdiagramm für die verschachtelten Schleifen

Im nachfolgenden geht es nach dem (nicht immer angebrachten) Motto: Vertrauen ist gut, Kontrolle besser:

```
172 IF Z1>YO OR Z1>YO THEN 162
174 IF Z2<XU OR Z2<YU THEN 162
```

Dann lassen wir uns die Freiheit der Farbenwahl:

```
180 PRINT:INPUT"ZEICHEN- UND HINTERGRUNDFARBE=";F1,F2
```

Nach dem Anschalten des hochauflösenden Modus zeichnen wir die Achsen:

```
190 GOSUB 50100:X1=FNX(XU):Y1=FNY(0):X2=FNX(XO):Y2=Y1:GOSUB 50060
192 X1=FNX(0):Y1=FNY(YU):X2=X1:Y2=FNY(YO:GOSUB 50060
194 X1=FNX(Z2):Y1=FNY(Z2):X2=FNX(Z1):Y2=FNY(Z1):GOSUB 50060
```

Wir könnten jetzt mit den beiden ineinandergeschachtelten Schleifen beginnen, wenn es da nicht noch das Problem der Schrittweite in den Schleifen gäbe. Die Schrittweite von X haben wir schon im 2D-Programm verwendet:

```
200 DX=(XO-XU)/319
```

Die Schrittweite von Z ist nicht so einfach zu fassen, weil man ja, je nach Kurvenart, mal weitere und mal engere Schritte machen muß. Deswegen halten wir diese Schrittweite veränderlich und lassen uns vorher nochmal danach fragen:

```
202 DZ=A*(Z1-Z2)/(FNX(Z1)-FNX(Z2))
182 INPUT"SCHRITTWEITE VON Z(CA.8)=";A:PRINT
```

Mit Zeile 202 hätten wir bei A=1 die selbe Punktauflösung wie in der X-Schleife. Das ist meistens zu eng und A sollte deshalb Werte um etwa 8 haben.

Es ist ganz gut, sich auch die Freiheit offenzulassen, von wo bis wo unsere Raumfläche zu zeichnen ist. Es sieht einfach besser aus, wenn sie nicht bis zum Rand des Bildschirms reicht. Die Schleifen sollten also nicht von ZU bis ZO und von XU bis XO reichen, sondern von den Anfangswerten ZA beziehungsweise XA bis zu den Endwerten ZE beziehungsweise XE, Auch diese Werte müssen abgefragt werden:

```
184 PRINT CHR$(18)"UNSERE ZEICHNUNG:"CHR$(146):PRINT
185 INPUT"X-BEREICH XA,XE=";XA,XE
186 INPUT"Z-BEREICH,ZA,ZE=";ZA,ZE
```

Jetzt kommen die Schleifen:

```
210 FOR Z=ZA TO ZE STEP DZ:ZZ=FNZ(Z)
220 FOR XX=XA TO XE STEP DX:X1=XX + ZZ
230 X=FNX(X1):YY=FNA(XX):Y1=YY + ZZ:Y=FNY(Y1):GOSUB 50040
240 NEXT XX
250 NEXT Z
```

Abschließend muß wieder auf den Normalbildschirm zurückgeschaltet werden :

```
300 GET A$:IF A$=""THEN 300
310 GOSUB 50030
320 END
```

So! Nun können Sie das ganze vorsichtshalber abspeichern und dann mit RUN starten. Aber fassen Sie sich in Geduld. Bis das Bild komplett ist, vergehen zirka 21 Minuten. Probieren Sie mal die Eingaben:

XU=-1,XO=6,YU=-2,YO=6,ZU=-1
ZO=6,Schrittweite von Z=8,XA=0,XE=4,ZA=0,ZE=4.

Es wird Ihnen sicherlich aufgefallen sein, daß ich zu Beginn des Programms den gleichen Zeilennummernabstand wie beim 2D-Programm verwendet habe. Sie können, genauso wie dort, nun noch unseren Trick zur Funktionseingabe einsetzen. Wenn Sie das getan haben, ist unser 3D-Programm komplett und Sie können nach Herzenslust Funktionen ausprobieren. Allerdings kommt es zum Erreichen einer besonders schön aussehenden Raumfläche nur dann, wenn Sie die ganzen Eingaben zu Beginn des Programms wohlüberlegt machen oder aber mehrere Male ausprobieren.

Diese Menge an Eingaben hat zwar den Vorteil, daß man sehr frei in der Gestaltung des Ergebnisses ist, aber eine gewisse Automatik wäre schon ganz angenehm. Das überlasse ich Ihnen, denn der Programmaufwand dazu würde den Rahmen dieses Buches sprengen.

Auf ein anderes Problem werden wir im nächsten Kapitel eingehen: Wie verhindert man, daß auch Linien gezeichnet werden, die eigentlich beim realen Körper (oder bei der Fläche) nicht sichtbar sind, weil sie hinter anderen Teilen der Zeichnung verschwinden?

7

Das Ziel ist erreicht

7 Das Ziel ist erreicht

7.1 Das Hinterschneidungsproblem: Wie verhindert man, daß verdeckte Kanten gezeichnet werden?

Es stellte sich die Frage, wie man es wohl erreichen könnte, daß nur sichtbare Teile einer Raumfläche gezeichnet werden. H.W. Franke hat in einem Artikel über Computergrafik geschrieben: "Bis heute kann man das Problem noch nicht als völlig gelöst ansehen". Damit meinte er das Hinterschneidungsproblem. Zumindest für unsere Aufgabenstellung und den C 64 kann man es lösen, wenn man folgenden Gedankengang verfolgt. Stellen Sie sich den Bildschirm des C 64 in 320 senkrechte Linien unterteilt vor, für jeden möglichen X-Wert also eine Senkrechte. Nun nehmen wir mal eine davon (bei X=X1) und ordnen ihr in unserem Koordinatensystem willkürlich einen Y-Wert 0 zu.

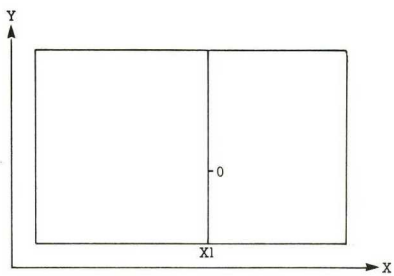


Bild 7.1 Eine der 320 Senkrechten (X=X1)

In der Doppelschleife wird nun irgendwann X=X1 und der zugehörige Y-Wert wird berechnet. Dieser soll zum Beispiel gleich 1 sein. Der Punkt wird gesetzt, und die Schleife läuft weiter.

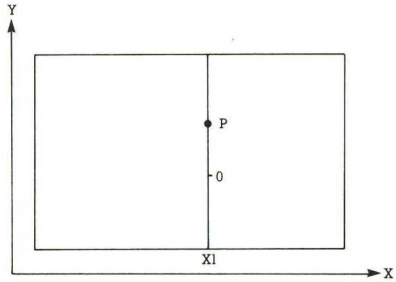


Bild 7.2 Der erste Punkt auf der Senkrechten

Wenn alle X-Werte durchlaufen wurden, erhöht sich der Z-Wert in der äußeren Schleife und von neuem wird für X1 ein Y-Wert berechnet. Nehmen wir an, der läge bei 1,5. Auch dieser Punkt wird gesetzt und die Schleife geht weiter.

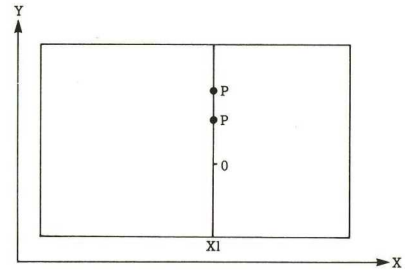


Bild 7.3 Der zweite Punkt P' ist gesetzt

Wenn nun beim nächsten Schleifendurchlauf X1 erreicht ist, wird es spannend. Es gibt nun drei Möglichkeiten.

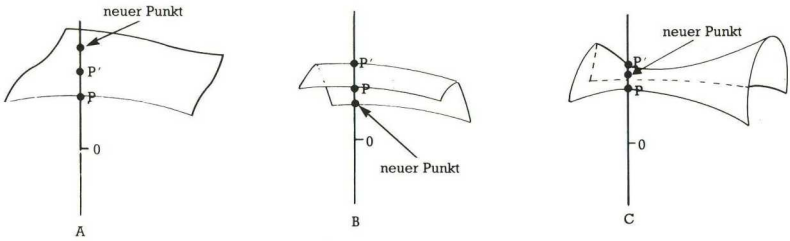


Bild 7.4 Die drei Möglichkeiten für einen neuen Punkt

Im Fall A liegt der neue Punkt oberhalb von P', ist sichtbar und wird gezeichnet. Im Fall B liegt er unterhalb von P, ist ebenfalls sichtbar und wird gezeichnet. Im Fall C aber liegt er zwischen P und P', er ist nicht sichtbar, weil er auf einem uns abgewandten Hang der Raumfläche liegt und wird daher nicht gezeichnet. Sie sehen also, daß wir uns lediglich immer auf jeder dieser Senkrechten den bisher größten und den bisher kleinsten Wert von Y merken müssen. Bei jedem neuen Y-Wert können wir feststellen, ob er innerhalb des damit aufgespannten Bereichs liegt (dann wird er nicht gezeichnet) oder außerhalb (dann wird er gezeichnet und dieser Wert als der größte oder kleinste bisherige gemerkt). Damit ist für uns das Problem gelöst! Wir richten ein zweidimensionales Feld ein: G(319,2), wo für jede der 30 Senkrechten drei Werte gespeichert werden können:

G(X,0)	als bisher höchster Y-Wert,
G(X,1)	als bisher kleinster Y-Wert,
G(X,2)	als Kennmarke, ob für diesen X-Wert schon ein Y-Wert aufgetaucht ist (in dem Fall ist G(X,2)>0) oder noch nicht (dann ist G(X,2)=0).

Durch das RUN-Kommando sind alle Variablen gleich 0, also auch G(X,2). Genau besehen benötigen wir diese Kennmarke, ob auf der Senkrechten durch X schon ein Punkt gesetzt wurde, nicht unbedingt in unserem Programm. Es ist so gestaltet, daß beim Durchlauf keine Lücken gelassen werden. Sinnvoll ist es trotzdem, sie einzurichten, denn durch die 45°-Verschiebung erhöht sich der maximale X-Wert ständig und außerdem könnte man sich überlegen, ob man den X-Durchlauf mit Lücken macht. Im letzten Kapitel hatten wir im Flußdiagramm unsere Doppelschleife entwickelt. Dabei war ein Teil "Zeichnen des Punktes X,Y". Diesen Teil ersetzen wir durch den in Bild 7.5.

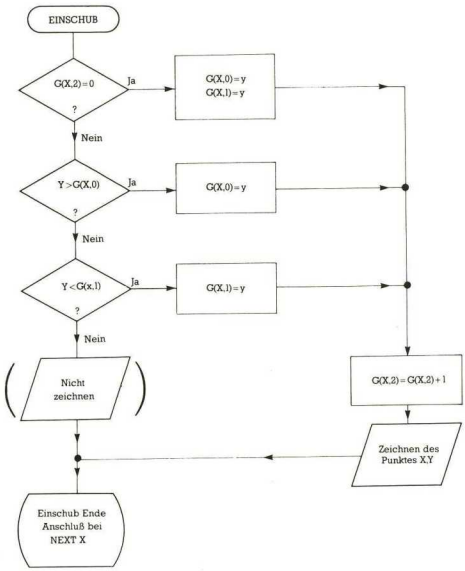


Bild 7.5 Flußdiagramm zum Hinterschneidungsproblem

Jetzt sehen wir uns das als Programmteil an. Also Computer einschalten. Laden des 3D-Programms aus dem letzten Kapitel, sowie der Grafik-Unterprogramme aus Kapitel 4 (entweder - falls Sie das haben - durch MERGEn oder als kombiniertes Programm aus der Arbeit im letzten Kapitel. Zeile 230 des 3D-Programms enthält als letzten Befehl GOSUB 50040, den Aufruf zum "Punkt zeichnen". Den löschen wir jetzt und fügen die folgenden Zeilen ein:

```
232 IF G(X,2)=0 THEN G(X,0)=Y:G(X,1)=Y:GOTO 238
234 IF Y>G(X,0) THEN G(X,0)=Y:GOTO 238
236 IF Y<G(X,1)THEN G(X,1)=Y:GOTO 238
237 GOTO 240
238 G(X,2)=G(X,2) + 1:GOSUB 50040
```

Außerdem muß natürlich zu Beginn dieses Feld noch dimensioniert werden:

```
147 DIM G(319,2)
```

Mit der Beispielfunktion Y = COS(Z)*SIN(X) und den Eingaben:

XU=-1, XO=10, YU=-2, YO=5, ZU=-1, ZO=7, Schrittweite=8, XA=0, XE=6, ZA=1, ZE=7

sieht man den Effekt jetzt ganz deutlich. Das komplette 3D-Programm finden Sie als Programm 5 auf der Diskette. Das einzige - außer der Geschwindigkeit (und naja - vielleicht noch ein paar 'Kleinigkeiten') - was unsere 3D-Grafik jetzt von professionellen Systemen unterscheidet, ist die Möglichkeit der Netzgrafik.

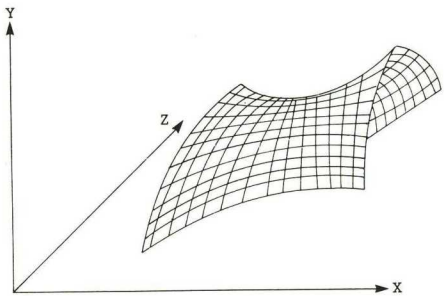


Bild 7.6 Netzdarstellung

Dieses Thema soll nicht ausführlich behandelt werden, sondern wir werden nur zwei Wege zur Netzgrafik anschauen:

Weg 1:
Wir machen die Schrittweite in Z-Richtung sehr klein (so wie die in X-Richtung), setzen aber nicht jeden Punkt:

a) erster Z-Wert: Jeder Punkt der X-Schleife wird gesetzt, dann

b) zweiter bis siebter Z-Wert: Nur jeder achte Punkt der X-Schleife wird gesetzt, dann

c) wieder weiter wie beim ersten Z-Wert und so weiter.

Stark vergrößert hätten wir dann etwa Verhältnisse wie in Bild 7.7.

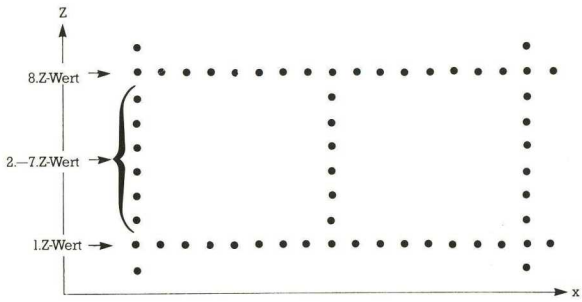


Bild 7.7 Eine Möglichkeit der Netzdarstellung

Diese Lösung ist programmtechnisch einfacher als Weg 2 und ganz gut an unser bisheriges 3D-Programm anzuhängen. Nun aber noch zu

Weg 2:
Man läßt außer der Z-X-Doppelschleife zur Berechnung von Y auch noch eine X-Z-Doppelschleife laufen. Wenn Sie das ausprobieren, gibt es allerdings wieder Schwierigkeiten mit den nicht sichtbaren Linien, denn unser Verfahren zur Lösung des Hinterschneidungsproblems läßt nichts mehr zeichnen, was innerhalb von G(X,0) bis G(X,1) liegt!

7.2 Grafik und Schrift in einem Bild

Anscheinend hat man bisher noch keine Möglichkeit Erklärungen auf das Grafikbild zu schreiben. Wenn wir im hochauflösenden Modus zum Beispiel das Programm durch ‹RUN/STOP› anhalten, dann tauchen alle Meldungen als farbige Quadrate auf dem Bildschirm auf. Der Inhalt des Bildschirmspeichers dient jetzt als Farbinformation. Was der Computer mit dieser Farbkombination dann jeweils auf dem Bildschirm zeigen soll, holt er sich aus der Bit-Map. Man kann natürlich ohne weiteres unter Verwendung unserer Grafik-Unterprogramme die wichtigsten Zeichen aus Punkten, Linien und Ellipsenbögen zusammenbauen. Das wäre sozusagen der "harte" Weg. Aber wozu haben wir im Speicher schon die fertigen Zeichenmuster liegen! Wir müßten nur auf sinnvolle Weise an sie herankommen. Prinzipiell gibt es zwei "weiche" Wege:

7.2.1 Der erste "weiche" Weg

Herbert Kunz hat den einen davon in der Zeitschrift Computer Persönlich, Ausgabe 2(1984), Seite 78 vorgestellt. Er kopiert zunächst den Zeichensatz in einen RAM-Bereich, schaltet dann - wie gewohnt - in den hochauflösenden Modus (bei ihm liegt der Bildschirm bei 1024 und die Bit-Map richtet er bei 8192 ein) und druckt den Text auf den Bildschirm, wo dieser erst einmal in farbigen Quadraten auftaucht. Nun sieht er mittels PEEK nach, welches Zeichen an der Bildschirmstelle steht (wo jetzt natürlich nur ein farbiges Quadrat zu sehen ist). Was Herbert Kunz dadurch erhält, ist die Kennzahl (der Bildschirmcode, zum Beispiel für ein A eine 1), die uns sagt, an welcher Stelle der Zeichentabelle das gefragte Zeichen steht. Dabei ist allerdings zu bedenken, daß es auch eine nullte Stelle gibt. Jedes Zeichen besteht aus 8 Bytes und deshalb multipliziert er die Kennzahl mit 8 und addiert sie zur Startadresse des RAM-Bereiches hinzu, in den er das Zeichen-ROM kopiert hat. Von da an überträgt er Byte für Byte das Zeichen in die entsprechende Stelle der Bit-Map (die er aus Zeilen und Spaltenangabe berechnet). So wird es dann sichtbar.

Das macht er Zeichen für Zeichen bis der gesamte - in einem String definierte - Text in der Bit-Map und damit für uns lesbar auf dem Bildschirm steht. Dieses Prinzip können wir in unsere Programme übernehmen. Dazu sind nur wenige Änderungen nötig. Zunächst schließen wir den neuen Zeichen-Speicher im RAM direkt an unsere Bit-Map an: ab 32768. Das Maschinenprogramm von Herbert Kunz verändern wir deshalb etwas und legen es nicht in den Kassettenpuffer, sondern von dez. 673 bis 715. Wie im Programm von Herbert Kunz brauchen wir dann nur noch die Angaben, in welcher Zeile und Spalte welcher Text geschrieben werden soll.

Zunächst einmal wird das Maschinenprogramm eingelesen und ausgeführt. In das 2D- und in das 3D-Programm fügen wir ein:

```
500 FOR I=673 TO 715:READ A:POKE I,A:NEXTI:SYS673: RETURN
510 DATA 120,169,49,133,1 169,0,133,98,133,100,169,208,133,99,169,
128,133
520 DATA 101,162,16,160,0,177,98,145,100,200,208,249,230,99,230,101,
202,208
530 DATA 242,169,55,133,1,88,96
```

In Zeile 5 hängen wir an die beiden POKE-Kommandos noch

```
GOSUB 500.
```

In Zeile 120 definieren wir einen String, der einen senkrechten Tabulator ermöglicht:

```
120 CP$=CHR$(19):FOR I=1 TO 24:CP$= CP$+CHR$(17):NEXT I
```

Dann definieren wir den zu druckenden String und Zeile und Spalte des Druckortes:

```
122 TE$="Y=" + F$:ZE=0:SP=0
```

Jetzt müssen wir nur noch dafür sorgen, daß nach dem Zeichnen der Kurve oder Raumfläche der Text gedruckt wird. Dazu gleichen wir zunächst mal das 2D- an das 3D-Programm an. In den Zeilen 200, 210, 220 des 2D-Programmes ändern wir die Zeilennummern zu 300, 310, 320 (dann muß natürlich in Zeile 300 die Anweisung THEN 200 zu THEN 300 umgeschrieben werden und die alten Zeilen 200 bis 220 gelöscht werden). Im 2D-Programm hört das Zeichnen in Zeile 195, im 3D-Programm in Zeile 250 auf. Deswegen legen wir den Druckvorgang ab Zeile 260:

```
260 PRINT LEFT$(CP$,ZE+1)TAB(SP)TE$;
262 AN=23552*ZE*40+SP:GS=24576+ZE*320+SP*8
264 FOR I=AN TO AN + LEN(TE$)-1
266 L=PEEK(I):Z=32768 + 8*L:POKE I,16*F1+F2
268 FOR J=0 TO 7:POKE GS + J,PEEK(Z+J):NEXT J
270 GS=GS+8:NEXT I
```

Wenn Sie das Programm (2D oder 3D) jetzt starten, bekommen Sie als Kopf des Bilds noch die gezeichnete Funktion als Gleichung gedruckt. Sie können sich leicht aus diesen Angaben ein Programm schreiben, in dem die Zeilen 260 bis 270 ein Unterprogramm bilden, das jeweils mit neuem Text TE$, neuer Zeile ZE und Spalte SP aufgerufen werden kann. So können Sie beliebigen Text in hochauflösende Grafikbilder schreiben.

7.2.2 Der zweite "weiche" Weg

Nun zum zweiten "weichen" Weg. Weil sich dieser nahezu völlig in Maschinensprache abspielt, soll hier nur das Prinzip erklärt werden. Sie finden das Programm "Rasterinterrupt" auch auf der Diskette.

Starten Sie mit RUN und Ihr Bildschirm ist in drei Zonen aufgeteilt. Ist das Programm beendet, drücken Sie eine Taste. Sie sind auch nach dem END noch in dieser Bildschirm-Aufteilung. Sie können das leicht feststellen, wenn Sie den Cursor nach oben bewegen und ein paar Schreibversuche machen. Oder versuchen Sie einmal das LIST-Kommando! Wenn Sie davon genug haben, geben Sie ‹RUN/STOP› und ‹RESTORE› ein, und Sie sind wieder im Normalzustand. Wenn man diese Technik beherrscht, kann man mit dem Bildschirm praktisch alles machen, was man will! Wir bedienen uns eines sogenannten Rasterzeilen-Interrupts.

```
10 REM** PROGRAMM B--RASTERINTERRUPT **
15 FORI=828TO954:READA:POKEI,A:NEXT:SYS828
20 BA=8192:DEFFNY(X)=INT(A+20*SIN(X/10+B))
25 PRINTCHR$(147):FORI=0TO8:PRINT:NEXT
30 PRINT"OBEN:HOCHAUFLOESUNGS-MODUS"
35 PRINT:PRINT"MITTE:NORMALER TEXT-MODUS"
40 PRINT:PRINT"UNTEN:MEHRFARBEN-BIT-MAP-MODUS"
45 REM* FARBEN SETZEN  BITMAP LOESCHEN*
50 FORI=0TO359
55 POKEI+1024,114:POKEI+1664,234:POKEI+55936,13
60 NEXTI
65 FORI=0TO39
70 POKEI+1384,6
75 NEXTI
80 FORI=BATOBA+3199
85 POKEI,0:POKEI+4800,0
90 NEXTI
95 REM*****   KURVEN ZEICHNEN    *****
100 A=40:B=0:FORX=0TO319:GOSUB135:NEXTX
105 A=160:B=0:FORX=0TO319STEP2:GOSUB135:NEXTX
110 B=40:FORX=1TO319STEP2:GOSUB135:NEXTX
115 B=80:FORX=0TO319STEP2:C=0:GOSUB135:C=1:GOSUB135:NEXTX
120 GETA$:IFA$=""THEN120
125 END
130 REM*****    FUNKT SETZEN    *****
135 BY=(XAND504)+40*(FNY(X)AND248)+(FNY(X)AND7):BI=ABS(7-(XAND7)-C)
140 POKEBA+BY,PEEK(BA+BY)OR(2↑BI):RETURN
145 REM****   MASCHINENPROGRAMM   ****
150 DATA120,169,127,141,13,220,169,1,141,26,208,169,3,133,251,173,172,3,141,18
155 DATA208,169,24,141,17,208,173,20,3,141,170,3,173,21,3,141,171,3,169,110
160 DATA141,20,3,169,3,141,21,3,88,96,173,25,208,141,25,208,41,1,240,43,198
165 DATA251,16,4,169,2,133,251,166,251,189,175,3,141,33,208,189,178,3,141
170 DATA17,208,189,181,3,141,22,208,189,184,3,141,24,208,189,172,3,141,18
175 DATA208,138,240,6,104,168,104,170,104,64,76,49,234,49,170,129,0,6,0,59
180 DATA27,59,24,8,8,24,20,24
READY.
```

Listing 7.1 Gleichzeitige Darstellung von Grafik, Text und Mehrfarb-Bit-Map

Ich habe Ihnen erklärt, daß der bildaufbauende Elektronenstrahl über Rasterzeilen huscht und 50mal in der Sekunde ein komplettes neues Bild aufbaut. Es ist ein enorm schneller Geselle, der über den Bildschirm huscht! Die akuelle Rasterzeile wird in den Registern 53266 (LSB) und 53265, Bit 7 (msb) mitgezählt. Interrupts bringen den Computer dazu, neben seiner uns sichtbaren Arbeit (zum Beispiel Programmablauf) noch eine Anzahl anderer Dinge zu tun. Eines davon ist die ständige Wiederauffrischung des Fernsehbildes durch Informationen an den Rasterstrahl. Der C 64 löst solche Interrupts auch per Programm aus. Zu diesem Zweck dienen die Register 53273 und 53274. Bit 7 von 53272 sagt uns, daß ein Interrupt aufgetreten ist (Bit 7 ist dann 1), mit einer der Ursachen, die noch in den Bits 0 bis 3 einzeln angegeben werden.

Bit 0=1 Rasterzeilen-Interrupt
Bit 1=1 Sprite/Hintergrund-Kollision
Bit 2=1 Sprite/Sprite-Kollision
Bit 3=1 Interrupt durch Lichtgriffel.

Register 53274 bietet uns die Möglichkeit einer sogenannten Interrupt-Maske. Bis auf Bit 7 ist es genauso aufgebaut wie 53273. Wenn wir hier zum Beispiel in Bit 0 eine Eins setzen, dann weiß der VIC-II-Chip, daß er einen sogenannten Rasterzeilen-Interrupt auslösen soll. Nur dies alleine würde kaum Wirkung haben, denn nun erfolgt beim Auslösen des Interrupts ein Sprung an die Adresse, die vom Interrupt-Vektor (Speicherstellen 788(LSB) und 789 (MSB)) angezeigt wird. Das ist im Normalfall ein Maschinenprogramm im Betriebssystem (Start bei 59963). Weil dieser Zeiger im RAM liegt, kann er verändert werden, so daß er auf ein eigenes Maschinenprogramm weist, das nun die Interruptbehandlung nach unserem Gutdünken ausführt. Außerdem müssen wir noch angeben, in welcher Rasterzeile der Interrupt stattfinden soll. Dazu schreiben wir in das Rasterregister diese Zeilennummer ein. Das ist das Prinzip, und weil diese ganze Angelegenheit sehr schnell erledigt sein muß, ist das nur in Maschinensprache möglich.

7.3 Grafik und Maschinensprache

Wer Grafik in Basic betreibt, braucht Sitzfleisch. Das haben Sie sicherlich am eigenen Leibe bemerkt. 20 Minuten für ein fertiges 3D-Bild, das ist schon ziemlich lange! Die meisten brauchbaren Grafikprogramme sind deshalb in Maschinensprache geschrieben. In Kapitel 8 wird das auf der Diskette enthaltene Grafik-Unterprogramm-Paket in Maschinensprache "HIRES-3" beschrieben, mit dem Sie den Zeitbedarf erheblich reduzieren können. Verantwortlich für diese lange Zeitdauer von Basic-Programmen ist der Basic-Interpreter, der jeden Befehl übersetzen muß und dann ein zum Befehl gehörendes, oft recht verwickeltes Maschinenprogramm ausführt, dann den nächsten Befehl übersetzt, und so weiter... Wenn das zum Beispiel in einer FOR...NEXT-Schleife mit 320 Durchläufen passiert, dauert das...! Je einfacher und auch allgemein verwendbarer ein Maschinenprogramm ist (jedenfalls für Grafik), desto umfangreicher muß das Basic-Aufrufprogramm sein. Oder: Je spezialisierter ein Maschinenprogramm ist, desto weniger Basic-Aufrufprogramm ist nötig. Ein Beispiel: Wenn das Maschinenprogramm lediglich die Routine zum Berechnen und Zeichnen eines Punktes enthält, muß vom Basic-Aufrufprogramm eine FOR...NEXT-Schleife 320 mal durchlaufen werden, in der jedesmal X variiert, Y aus der Funktionsgleichung berechnet wird, beide auf das Bildschirmsystem transformiert werden, die transformierten Werte an die Speicheradressen gePOKEt werden, von denen es die Maschinenroutine nach dem SYS-Aufruf abholt. Man hat im Aufrufprogramm allerlei Freiheiten: Man kann nach Belieben das Koordinatensystem ändern, die zu zeichnende Funktion, den Start- und den Endwert der FOR...NEXT-Schleife und das alles relativ einfach durch einige INPUT-Anweisungen oder notfalls Programmzeilenänderungen erreichen. Dafür muß man die lange Zeitdauer des Aufrufprogrammes hinnehmen. Ein Maschinenprogramm wäre zwar schnell und erforderte von Basic aus unter Umständen nur einen SYS-Befehl. Aber es wäre mehr oder weniger festgelegt auf immer diese eine Aufgabe und damit recht unbeweglich und auch ziemlich lang. Maschinenprogramme andererseits, die dieselben

Eingabe- und Variationsmöglichkeiten wie das vorhin erwähnte Basic-Aufrufprogramm bieten, sind schon etwas für Feinschmecker der Assemblerprogrammierung und äußerst rar.

Ein anderes Phänomen ist die Tatsache, daß man vor der Wahl steht: Geschwindigkeit oder Speicherplatz sparen. Man kann Grafik-Maschinenprogramme enorm durch spezielle Programmier-Techniken beschleunigen. Sie werden dann aber häufig so lang, daß man sie kaum mehr als DATA-Zeilen-Listing abdrucken kann.

7.4 Bewegte Grafik

Wie man einen Trickfilm mit Hilfe von Sprites drehen kann, habe ich Ihnen in Kapitel 5 gezeigt. Welche Möglichkeiten gibt es ohne Sprites? Schneller Bildaufbau - kurze Verzögerung - Bild löschen - neues Bild aufbauen mit veränderter Ansicht - und so weiter... Wenn Sie das in Basic überlegen, können Sie das gleich wieder vergessen. Um den Eindruck von Bewegung zu vermitteln, muß alles viel schneller gehen. Aber auch ein Maschinenprogramm muß sehr sorgfältig entwickelt werden, um die nötige Geschwindigkeit des Bildaufbaues zu erhalten. Für einfache Darstellungen könnte dieses Konzept aber funktionieren. Denken Sie zum Beispiel an unsere 3D-Bilder! Mit dem C 64 - behaupte ich - geht's so nicht.

Stellen Sie sich vor, wir benützen mehrere Bit-Maps und Bildschirme und zeichnen in je eine Bit-Map einen Bewegungszustand unseres Bildes. Dann müssen wir nur noch in einer Aufrufschleife von Bit-Map zu Bit-Map umschalten. Das ist auch in Basic möglich. Wir hätten dann drei Bit-Maps zur Verfügung. Das ist zwar nicht viel, aber immerhin könnte man damit schon eine Raumfläche in zwei Richtungen kippen oder ähnliches. Wenn man in Maschinensprache programmiert, hat man sogar 5 Bit-Maps (mindestens) zur Verfügung und weil auch noch alles schneller geht, kann man mit einigem Geschick vielleicht sogar während man vier Bilder nacheinander zeigt, das fünfte Bild ungesehen aufbauen und auf diese Weise mehr als fünf Bewegungsstadien realisieren.

Sie sehen aber schon: Das ist die hohe Schule der Programmierkunst und hier gibt es noch viel zu tun. Einen 3D-Trickfilm auf diese Weise zu erzeugen, ist heute auch auf großen Computern noch kaum möglich. Der Film TRON beispielsweise setzt sich aus lauter per Computer erzeugten Einzelbildern zusammen, die erst filmtechnisch zum Bewegungsablauf aneinandergehängt wurden.

Dort wo Echtzeit-Darstellung notwendig ist, bei Simulationen beispielsweise in Flugtrainern, reduziert man die Darstellung auf das unbedingt notwendige und hat außerdem dazu Computer zur Verfügung, die uns C 64-Benutzern das Wasser im Mund zusammenlaufen lassen. Aber was soll's. Die Kunst des Programmierens liegt vielleicht

darin, daß man mit einem Minimum an Aufwand einen Maximaleffekt erzielt. Und wenn man sich dann mal ansieht, was wir aus unserem C 64 alles herausholen können, stehen wir eigentlich ganz gut da, meinen Sie nicht auch?

8

HIRES-3 — die Super-Grafikerweiterung

8 HIRES-3 - die Super-Grafikerweiterung

Nachfolgend finden Sie das Listing (am Ende des Kapitels) und eine genaue Beschreibung (mit Gebrauchsanweisung) des Grafik-Hilfsprogramms, das Sie auf der Diskette unter dem Namen HIRES-3 finden. Sollte jemand den Mut besitzen, die riesigen Zahlenkolonnen in den Rechner eingeben zu wollen, sollte er wissen, daß dies nur mit Hilfe des MSE (ab Ausgabe 2/85 des 64er Magazins) möglich ist. Auf der Diskette sind außerdem noch zwei kleine Programme (TEST-DEMO-1 und TEST-DEMO-2), mit denen Sie alle Grafikbefehle auf ihre Funktionstüchtigkeit testen können.

HIRES-3 wurde mit dem Gedanken entworfen, das Schreiben von Grafikprogrammen einfacher zu gestalten. Deswegen sind nicht nur ausgesprochene Grafikbefehle dabei, sondern auch OLD, MERGE, RENUMBER, AUTONUMBER, PAUSE, UHR etc. Die Grafik-Unterstützung ist für alle Anwendungsbereiche gedacht, wobei der Schwerpunkt allerdings weniger auf "Spielen" und "künstlerische Anwendungen" liegt. Deswegen sind keine Joystick-, keine Lightpen- und keine Sprite-Befehle enthalten. Schließlich spielte es noch eine große Rolle, daß viel Speicherplatz für Basic-Programme und Daten frei bleiben sollte, ebenso wie der für Maschinensprache interessante Bereich von $C000 bis $D000.

Deswegen liegt das Programm HIRES-3 ab $8000 (dez.32768) bis $89B5 (dez.35253) und von $9000 (dez.36864) bis $9DCB(dez.40395). Es fängt also dort an, wo auch Module ihren Platz haben. Der Bildschirm für hochauflösende Grafik liegt von $8C00 (dez.35840) bis $8FFF (dez.36863), der für den Normalbetrieb bleibt an der gewohnten Stelle (1024...). Die 8000 Byte lange Bit-Map kommt uns nicht mehr in die Quere: Sie ist unter dem Basic-ROM versteckt. Wir haben trotz 44 neuer Befehle und Funktionen und einer stets präsenten hochauflösenden Grafik auf einem zweiten Bildschirm immer noch 30 KByte RAM für Basic frei und zusätzlich auch noch den Bereich $C000 bis $D000 für Maschinenprogramme.

Laden Sie das Programm mit LOAD "HIRES-3 ML$8000",8,1 von der Diskette. Es wird absolut geladen, also direkt ab $8000 in den Speicher, um die Zeiger zurückzustellen, geben Sie danach bitte NEW ein.

Bevor Sie ein neues Basic-Programm schreiben, sichern Sie bitte noch HIRES-3 durch die in Kapitel 2 bereits vorgestellten Schutz-POKEs:

```
POKE52,128:POKE56,128
```

Mit SYS36864 starten Sie HIRES-3. Außer der READY-Meldung werden Sie erstmal nichts Neues sehen. Sie haben nämlich nichts anderes getan, als den Funktionstasten Befehle zuzuteilen. Deshalb fangen wir jetzt mit der Erklärung der Befehle an und zwar mit eben diesen Funktionstasten:

8.1 Hilfsfunktionen

8.1.1 Funktionstastenbelegung

Genau genommen schaltet man durch SYS36864 nur die Belegung der Funktionstasten ein. Ausschalten erfolgt durch gleichzeitiges Drücken der RUN/STOP- und der RESTORE-Tasten. Nun zur Belegung im einzelnen:

F1 RUN

Startet ein im Basic-Speicher stehendes Programm sofort.

F2 RENUMBER-Befehl

Auf dem Bildschirm erscheint SYS33256. Man schreibt nun:

SYS33256, Abz,Nbz,Sch.

Dabei bedeuten:

Abz = erste alte Basic-Zeile, von der an neu numeriert werden soll.
Nbz = erste neue Basic-Zeilennummer
Sch = Schrittweite

Dieser Befehl numeriert auch alle GOTO, GOSUB, IF...THEN..., RUN... etc. neu.

Beispiel:

SYS33256,1,10,5 numeriert ein Programm ab der alten Zeilennummer 1, welche nun 10 heißt, in 5er-Schritten neu durch.

F3 Aktivieren der neuen Basic-Befehle

Auf dem Bildschirm erscheint SYS37498 und READY. Jetzt sind alle neuen Basic-Befehlsworte verfügbar. Verwendet man eines der neuen Befehlsworte vorher, erzeugt das einen SYNTAX-ERROR. Das Abschalten der neuen Befehle geschieht mittels "AUS".

F4 OLD-Befehl

Auf dem Bildschirm erscheint SYS37227 und ein durch NEW oder einem RESET gelöschtes Basic-Programm wird wieder gelistet und funkionsfähig (vorausgesetzt, daß es nicht durch Überschreiben zerstört wurde). Störungen können auftreten, wenn noch kein Basic-Programm im Speicher enthalten ist. Nach der Verwendung dieses Befehls sollte überprüft werden, ob durch das vorangegangene Löschen die Schutz-POKEs 52 und 56 verändert worden sind.

F5 AUTONUMBER-Befehl

Auf dem Bildschirm erscheint SYS37018 und eine Zeilennummer. Diese Funktion setzt automatisch nach jedem RETURN eine neue Zeilennummer, die sich an die letzte im Programm vorhandene anschließt. Im Normalfall wird dabei in 10er-Schritten gearbeitet, eine andere Schrittweite wird durch POKE37169,Sch erzeugt (dabei ist Sch die Schrittweite). Die AUTONUMBER-Funktion wird durch Drücken von RETURN direkt nach einer Zeilennummer abgeschaltet.

F6 MERGE-Befehl

SYS37306 erscheint auf dem Bildschirm und die Information:"***MERGE***, BEREIT ZUM KOPPELN! READY". Das Basic-Programm läßt sich allerdings nicht Listen oder Starten. Denn die Zeiger sind nun auf das Ende des Programms gestellt. Der Computer wartet auf das Laden des anzuhängenden Programms, was durch eine normale Ladeoperation stattfindet. Man drückt nun erneut F6 und auf dem Bildschirm erscheint: SYS37306 und der Text: "***MERGE*** PROGRAMME GEKOPPELT! READY". Wenn man nun listet, zeigt es sich, daß beide Programme aneinandergehängt sind. Man sollte darauf achten, daß das anzuhängende Programm höhere Zeilennummern aufweist als das erste, denn sonst kommt es bei GOTO oder GOSUB-Sprüngen zu Fehlern.

F7 Startet Maschinenprogramme

die bei $C000(dezimal49152) liegen. Auf dem Bildschirm erscheint SYS49152.

F8 Startet Maschinenprogramme,

die bei $7000(dez.28672) liegen. Auf dem Bildschirm erscheint SYS 28672.

Achtung! Diese beiden Tasten sollte man nur betätigen, wenn an den Ansprungadressen auch wirklich ein Programm startet, denn sonst stürzt das Programm unter Umständen ab. Sehr praktisch sind diese beiden Funktionstasten, wenn an den Startadressen andere Sprachen verfügbar sind, zum Beispiel bei $C000 ein Maschinensprache-Monitor

liegt oder ähnliches. Auch zum schnellen Starten von unfertigen Assembler-Programmen, die an einer der beiden Stellen anfangen. für solche Tests sind F7- und F8 gut einzusetzen.

8.1.2 Hilfsfunktionen als neue Basic-Befehle

Das sind fünf Befehle: AUS, DEEK, PAU, DUMP, UHR

AUS

Schaltet die Befehlserweiterungen aus. Das beschleunigt den Ablauf eines Basic-Programms. Der Interpreter muß nun nicht mehr den Umweg über die hier vorgestellten Befehlserweiterungen laufen. Der Zeitunterschied ist allerdings minimal und nur bei zeitkritischen Abläufen interessant.

DEEK

Ist eine modifizierte PEEK-Funktion, die das RAM unter den ROM-Bausteinen ausliest und in Speicherstelle 2 ablegt.

Syntax: DEEK,Speicherstelle

Beispiel: DEEK,53272:PRINT PEEK(2)

ergibt 255, den Inhalt des RAM unter dem ROM, wogegen PRINT PEEK(53272) den ROM-Inhalt 21 ergibt.

PAU

Pausen-Befehl. PAU,10 erzeugt eine Pause von 10 Sekunden.

DUMP

Gibt alle definierten Variablen mit ihren aktuellen Werten aus. Arrays werden nicht berücksichtigt.

UHR

Zeigt in der rechten oberen Bildschirmecke eine Uhr an.

Stellen: UHR, "hhmmss",Zeichenfarbe
Ausschalten: UHR
Wieder einschalten: UHR

Vor der Verwendung des hochauflösenden Modus sollte die UHR ausgeschaltet werden, ebenso vor dem Aufruf des Hardcopy-Befehls.

8.2 Grafik-Befehle

8.2.1 Einrichten der Grafik

HFL,Zf,Hf

Der Befehl richtet den hochauflösenden Grafikmodus ein, setzt die Zeichenfarbe (Zf) und die Hintergrundfarbe (Hf) und löscht ein eventuell vorhandenes Bild. Dabei liegt der Bildschirm von $8C00 bis $8FE7 (dezimal 35840 bis 36839) und die Bit-Map unter dem Basic-ROM ($A000-$C000).

HAN

Schaltet den hochauflösenden Modus ein.

FAR,Zf,Hf

Setzt die Farben im Hochauflösungsbild.

LOE

Löscht nur Hochauflösungsbild.

HOF

Schaltet den hochauflösenden Modus aus und richtet den Normalmodus wieder ein.

8.2.2 Zeichnen im Bildschirmsystem

(Mit x von 0 bis 319, y von 0 bis 199). Erinnern Sie sich bitte an das Bildschirmkoordinatensystem. Alle X- und Y-Koordinaten, die unter 8.2.2 und 8.2.3 benutzt werden, beziehen sich auf dieses System.

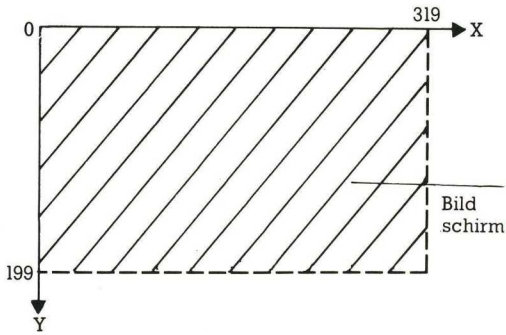


Bild 8.1 Die Gestalt des Bildschirmsystems

PKT,x,y

Zeichnet an der durch x und y angegebenen Stelle des Bildschirms einen Punkt. Die Bedeutung der Bezeichnungen sind im Bild 8.2 zu finden. Wegen der Eigenart des Bildschirmsystems ist darauf zu achten, daß x und y nie kleiner als 0, x nie größer als 319 und y nie größer als 199 werden. Koordinaten-Eingaben, die größer als 399 (beziehungsweise 199) sind, führen lediglich dazu, daß kein Punkt gezeichnet wird. Eingaben kleiner als Null ergeben einen SYNTAX ERROR.

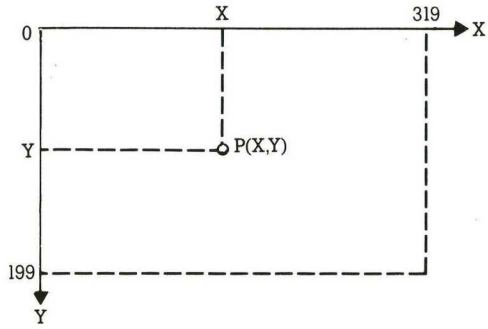


Bild 8.2 Bezeichnungen beim PKT-Befehl

LIN,xa,ya,xb,yb

Zeichnet eine Linie vom Punkt A mit den Koordinaten xa,ya bis zum Punkt B mit den Koordinaten xb,yb (siehe Bild 8.3). Die Richtung der Linie ist beliebig. Die Bemerkung zur Größe der Koordinaten gilt hier entsprechend.

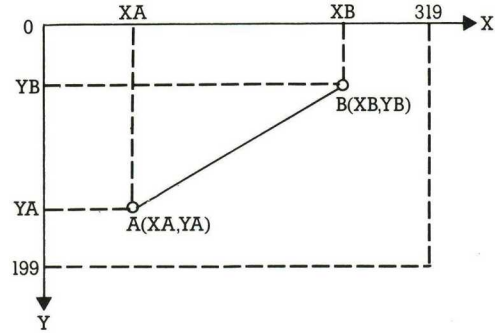


Bild 8.3 Bezeichnungen beim LIN-Befehl

REC,xa,ya,xb,yb

Zeichnet ein Rechteck, das durch den linken oberen Punkt A(xa,ya) und den rechten unteren Punkt B(xb,yb) gekennzeichnet ist (siehe Bild 8.4). Für die Koordinatengrenzwerte gilt dasselbe wie beim Befehl PKT.

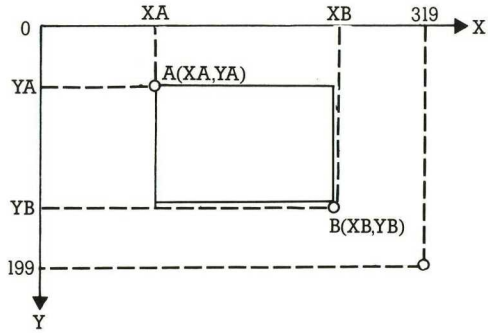


Bild 8.4 Bezeichnungen beim REC-Befehl

BLO,xa,ya,xb,yb

Füllt ein Rechteck der angegebenen Maße (siehe REC für die Bezeichnungen) mit der Zeichenfarbe aus.

CIR,xm,ym,rx,ry,w

Zeichnet eine Ellipse oder einen Kreis (Sonderfall der Ellipse mit rx=ry) mit den folgenden Merkmalen:

xm,ym= Koordinaten des Mittelpunktes M,
rx,ry = Halbmesser in x- beziehungweise y-Richtung.

w = Zeichenwinkel (Bogenmaß). Zur Erläuterung der einzelnen Bezeichnungen dient Bild 8.5. Außer negativen xm- und ym-Koordinaten sind alle Eingaben zulässig.

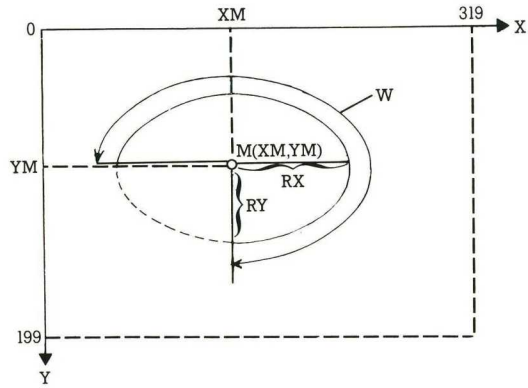


Bild 8.5 Bezeichnungen beim CIR-Befehl

RAD,xm,ym,rx,ry,w

Zeichnet in die Ellipse einen Radius ein. Die Bezeichnungen sind dieselben wie für den CIR-Befehl. w zeigt hier, an welche Stelle im Ellipsenbogen der Radius gezeichnet werden soll (siehe Bild 8.6).

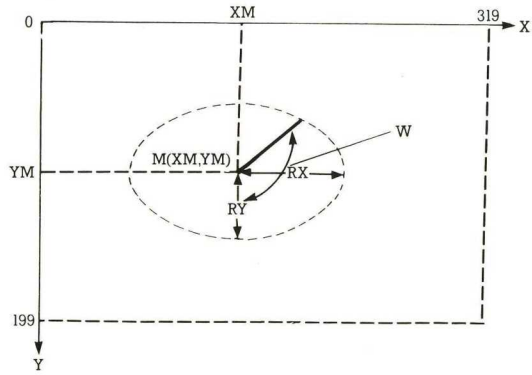


Bild 8.6 Bezeichnungen beim RAD-Befehl

8.2.3 Löschen im Bildschirmsystem

LPK,x,y löscht den angegebenen Punkt x,y.

LLN,xa,ya,xb,yb löscht die Linie von xa,ya bis xb,yb.

LRE,xa,ya,xb,yb löscht das Rechteck (Bezeichnungen wie bei REC).

LBK,xa,ya,xb,yb löscht das ausgefüllte Rechteck.

LKR,xm,ym,rx,ry,w löscht die Ellipse (Bezeichnungen wie bei CIR).

LRA,xm,ym,rx,ry,w löscht den Ellipsenradius.

8.3 Abspeichern/Laden von Hochauflösungsbildern

HIS,"Name",gn,sa

SAVEn des Bildes mit "Name" auf dem Gerät mit der Gerätenummer gn. Es gelten die gleichen Regeln wie beim normalen SAVE-Vorgang.

HIL,"Name",gn,sa

Laden eines Bildes vom Speichermedium mit der Gerätenummer gn. Es gelten die gleichen Regeln wie für den normalen Ladevorgang.

8.4 Grafik-Befehle für frei wählbare Koordinatensysteme

8.4.1 Einrichten eines frei wählbaren Koordinatensystems

TRS,XU,XO,YU,YO

Für alle mit T oder LT beginnenden Befehle muß der TRS-Befehl vorher eingegeben worden sein. TRS baut den Bildschirm nach unseren eigenen Wünschen um. Das normale Bildschirmkoordinatensystem, in dem X von 0 bis 319, Y von 0 bis 199 (abwärts) läuft, wird durch ein uns genehmes ersetzt. XU und XO sind dabei die kleinste und größte X-Koordinate, YU und YO entsprechend die kleinste und die größte Y-Koordinate, die wir wünschen.

Beispiel: Wir wollen eine Sinus-Funktion zeichnen lassen. Uns interessiert der Verlauf von XU=-10 bis XO=6. Wir wissen, daß die Y-Werte zwischen +1 und -1 hin- und herpendeln, wählen daher YU=-1.5 und YO=1.5: Wir geben ein TRS,-10,6,-1.5,1.5. Unser Bildschirm ist jetzt so organisiert, wie es Bild 8.7 zeigt.

Dieser Befehl ist wohl der stärkste, der in HIRES-3 enthalten ist. Innerhalb von Programmen kann er mehrmals verwendet werden, was zum Beispiel die Erstellung von 3D-Grafik erleichtert. Als Argumente dürfen beliebige Zahlen, arithmetische Ausdrükke oder mathematische Basic-Funktionen verwendet werden.

Nehmen Sie an, Sie untersuchten eine Funktion, bei der Sie ein Maximum im X-Bereich -200 bis -300 vermuten, und wo der Y-Wert irgendwo zwischen 1000000 und 10000000 sein könnte. Dann benutzen Sie unseren Bildschirm wie ein Kameraobjektiv. Fahren Sie ihn auf diesen interessanten Ausschnitt mit TRS,-300,-200,1E6,1E7! Grenzen sind Ihnen nur insofern gesetzt, als unser Computer keine größere Zahl als 1.70483E38 und keine dem Betrag nach kleinere als 2.93873588E-39 kennt.

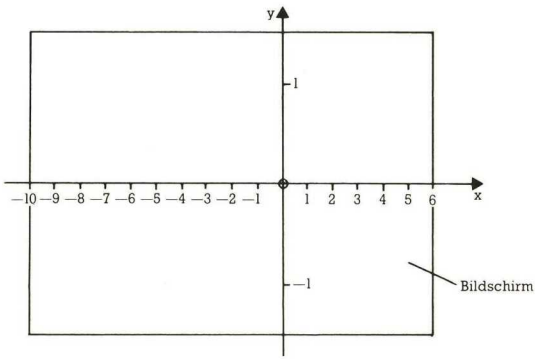


Bild 8.7 Bildschirmorganisation nach: TRS,-10,6,-1.5,1.5

Dazu noch eine Bemerkung. Alle Eingaben werden als Gleitkommazahlen verarbeitet. Der Prozessor 6510 unseres C 64 ist mit nicht ganz 1 MHz getaktet. Gleitkomma-Arithmetik ist eine recht aufwendige Angelegenheit. Je komplizierter die Ausdrücke werden, die als Argumente auftreten, desto längere Interpreterteile werden durchlaufen, und desto langsamer erfolgt die Befehlsausführung. Beispielsweise kann ein Argument 2.234 sofort verarbeitet werden, wohingegen bei einem Argument SIN(SQR(A^COS(B))) ein erheblich längerer Rechengang erforderlich ist. Das sind eben die Grenzen unseres Systems, die wir hier spüren. Trotzdem geht alles noch erheblich schneller als in Basic.

Auch nachdem Sie mittels des TRS-Befehls den Bildschirm neu organisiert haben, können Sie die Befehle benutzen, die sich auf den alten Bildschirm beziehen. Sie haben somit zwei Koordinatensysteme gleichzeitig zur Verfügung: Das normale (X von 0 bis 319, Y von 0 bis 199) und Ihr selbstdefiniertes.

Die Abkürzung TRS kommt von dem Wort Transformation. Hier soll nicht näher erklärt werden, was das ist und wie das funktioniert. Wenn Sie mehr darüber wissen möchten, dann schlagen Sie dies bitte den vorangegangenen Kapiteln nach. Dort ist

der Begriff näher erläutert. Nur soviel soll dazu gesagt werden: Mittels geeigneter Transformationen können Sie wahre Wunderdinge auf dem Bildschirm vollbringen.

8.4.2 Zeichnen im selbstdefinierten System

Bei allen folgenden Befehlen sollte beachtet werden, daß vorher mit TRS das neue System definiert worden ist. Alle X und Y-Werte sollten möglichst innerhalb der damit festgelegten Grenzen XU-XO und YU-YO liegen. Das Überschreiten der Unterschreiten der Grenzen führt im allgemeinen nicht zu Fehlermeldungen. Es können sich aber unter Umständen falsche Bildschirmdarstellungen ergeben. Im allgemeinen kann man diese dann deutlich erkennen und durch geeignete Neuwahl der TRS-Argumente beheben.

TPK,X,Y

zeichnet einen Punkt mit den Koordinaten X und Y in das vorher definierte System.

Beispiel: TRS,-1,5,-2,6:TPK,1,1

zeichnet in der in Bild 8.8 gezeigten Weise einen Punkt auf den Bildschirm.

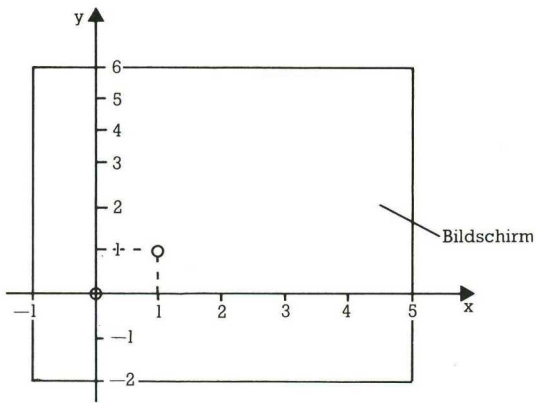


Bild 8.8 Bildschirm nach TRS,-1,5,-2,6 : TPK,1,1

TLN,XA,YA,XB,YB

zeichnet eine Linie in das vorher selbstdefinierte Koordinatensystem ein, die vom Punkt A(XA,YA) bis zum Punkt B(XB,YB) verläuft. Mit TLN läßt sich nun recht einfach das Achsenkreuz des selbstgewählten Systems zeichnen: TLN,XU,0,XO,0 zeichnet die X-Achse und TLN,0,YU,0,YO die Y-Achse.

TRE,XA,YA,XB,YB

zeichnet in das selbstgewählte System ein Rechteck. Der Punkt A(XA,YA) ist der linke obere, B(XB,YB) der rechte untere Eckpunkt.

TBK,XA,YA,XB,YB

zeichnet in das durch TRS definierte System ein ausgefülltes Rechteck. Die Bezeichnungen sind dieselben wie bei TRE.

TKR,XM,YM,RX,RY,W

zeichnet in das neue Koordinatensystem eine Ellipse oder einen Kreis (dann ist RX=RY). Für die Bezeichnungen gelten die Regeln des CIR-Befehls.

TRA,XM,YM,RX,RY,W

zeichnet ins TRS-System einen Radius ein. Es gelten dieselben Bezeichnungen wie beim RAD-Befehl.

FUNKT,Name,XA,XB

zeichnet in das mittels TRS definierte Koordinatensystem die Funktion "Name" ein. Gezeichnet wird dabei der Bereich von von XA bis XB. Im Gegensatz zu den sonstigen Basic-Regeln für Funktionsnamen ist hier nur ein Buchstabe als Name erlaubt. Mehr als 26 (Buchstaben A bis Z) Funktionen gleichzeitig wird aber vermutlich kaum jemand auf dem Bildschirm darstellen wollen, oder?

Beispiel:
```
DEFFNA(X) = SIN(X)
TRS,-10, 10, -2,2
FUNKT, A, 0, 3.14
```

zeichnet die definierte Sinus-Funktion in das durch TRS festgelegte System. Das Ergebnis zeigt Bild 8.9.

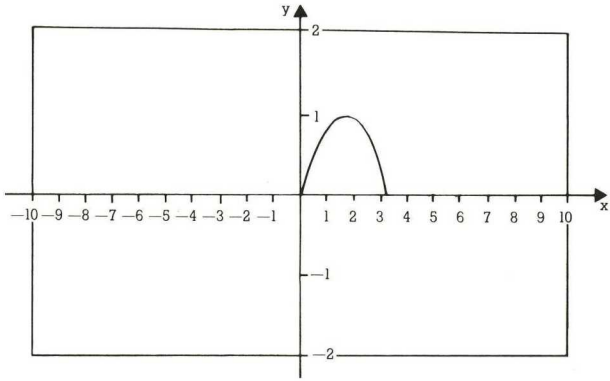


Bild 8.9 Bildschirm nach: DEFFNA(x)=SIN(x) TRS,-10,10,-2,2 FUNKT,A,0,3.14

Auch für diesen Befehl gilt das für TRS gesagte. Die Ausführungszeit steigt stark an, je komplexer die Funktion ist. Trotzdem geht alles noch schneller als in einer FOR...NEXT-Schleife, die Punkt für Punkt die Funktion zeichnet.

8.4.3 Löschen im selbstdefinierten System

LTP,X,Y	löscht einen Punkt im TRS-System.
LTL,XA,YA,XB,YB	löscht eine Linie im neuen System.
LTR,XA,YA,XB,YB	löscht das Rechteck im neudefinierten Koordinatensystem.
LTB,XA,YA,XB,YB	löscht ein ausgefülltes Rechteck im selbstdefinierten System.
LTK,XM,YM,RX,RY,W	löscht eine Ellipse (einen Kreis) im TRS-System.
LTV,XM,YM,RX,RY,W	löscht einen Radius im neuen System.
LFUNK, Name, XA,XB	löscht die Funktion Name im Bereich XA bis XB im selbstgebauten Koordinatensystem.

8.5 Noch zwei Kleinigkeiten

Zwei Programmteile gibt es noch, die über SYS-Befehle anzuspringen sind.

SYS34647

zeichnet auf den Normalbildschirm einen schwarzen Rahmen mit einer Kopfzeile. Den Inhalt dieser Zeile können Sie verändern, indem Sie den neuen Inhalt in die Tabelle ($87FE bis $8830) einschreiben, durch EinPOKEn des Commodore-ASCII-Codes ihres Textes. Nach dem Aufruf dieses Programms ist die Zeichenfarbe Schwarz. Durch POKE 646,"gewünschter Farbcode" können Sie das schnell ändern.

SYS34865, Bit-Map Start

Damit können Sie jede gewünschte Bit-Map per Drucker zu Papier bringen. Die Startadresse der Bit-Map ist in HIRES-3 dez. 40960. Wenn dieser Befehl gegeben wird, während sich am Bildschirm die hochauflösende Bilder tummeln, braucht die Bit-Map-Startadresse nicht angegeben werden.

Zwei Wermutstropfen mischten sich in die freudige Anwendung dieser Hardcopy-Routine:

Erstens muß sich irgendwo im Programm eine besonders gut versteckte Wanze (bug) befinden. Ich bin mir nicht einmal sicher, ob die Ursache nicht etwa im Druckerbetriebssystem versteckt liegt. Wenn man vor jedem Aufruf dieser Hardcopy-Routine den Drucker durch OPEN1,4,10:PRINT#1:CLOSE1 in die Ausgangsstellung bringt, funktioniert der Ausdruck fast fehlerfrei. Lediglich dann, wenn ganz links oben auf dem Bildschirm schon eine Information zu übermitteln ist, gibt der Drucker dort das Pi-Zeichen aus. Der Rest des Bildes ist einwandfrei.

Zweitens ist dieses Programm für den Commodore-Drucker 1526 erstellt worden. Jeder, der dieses Gerät besitzt, wird den Ärger mit den vielen verschiedenen Betriebssystemen des 1526 kennen. Vermutlich werden Sie gar nicht wissen, welches Sie in Ihrem Drucker haben. Da hilft nur eines: Probieren Sie den Aufruf der Hardcopy-Routine aus. Wenn's funktioniert, haben Sie das richtige Betriebssystem, wenn nicht, wenden Sie sich vertrauensvoll an Manfred Böhmel, Am Töbele 2 in 7923 Königsbronn. Der hat meinen nicht grafikfähigen 1526 in ein mich zufriedenstellendes Gerät verwandelt. In Tabelle 8.5 sind die Startadresssen aller HIRES-3 Programmteile angegeben. In vielen Zeitschriften finden sich Maschinenprogramme für Hardcopies auf diversen Drukkern. Sollten Sie solch ein Programm für Ihren Druck erfinden, können Sie es leicht gegen das hier vorgestellte austauschen.

Damit kennen Sie HIRES-3 von der Anwendung her. Auf der Diskette finden Sie zwei kleine Testprogramme, die alle neuen Funktionen überprüfen (Bilder 8.10 bis 8.13). Profis der Programmierung in Maschinensprache wird nun natürlich noch einiges interessieren: Welche Zeropage-Adressen benötigt HIRES-3 und welche Adressen aus den Seiten 1-3? Welche Routinen werden aufgerufen? Wo findet man welche Programmteile? Um all diese Fragen zu klären, sind die Tabellen 8.1 bis 8.5 angefügt.

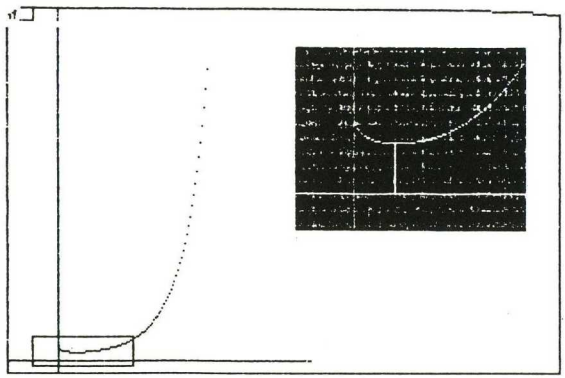


Bild 8.10 Das erste Bild des Testprogramms

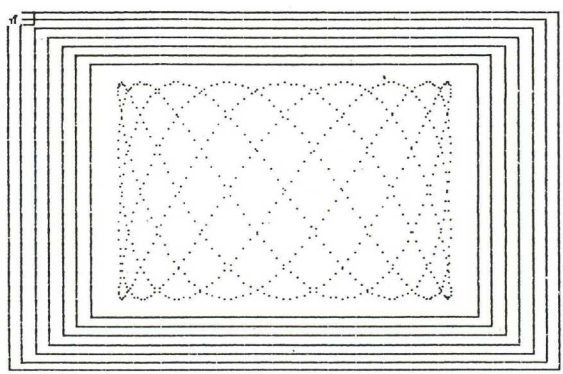


Bild 8.11 Das zweite Bild des Testprogramms

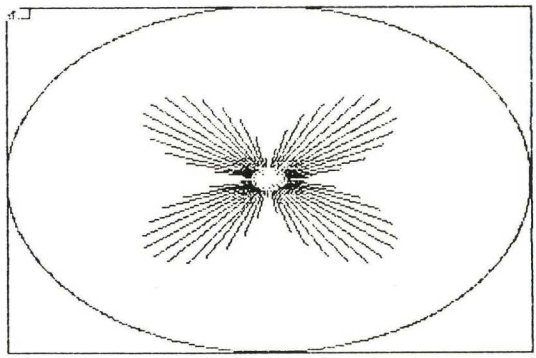


Bild 8.12 Das dritte Bild des Testprogramms

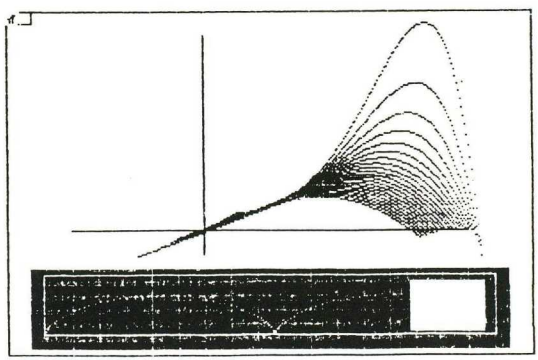


Bild 8.13 Das letzte Bild des Testprogramms

Tabelle 8.1 gibt alle durch HIRES-3 ausdrücklich verwendeten Zeropage-Adressen an. Ausdrücklich deshalb, weil im Rahmen von Interpreter- oder Betriebssystem-Routinen ebenfalls Zeropage-Adressen angesprochen werden, die aber hier nicht benannt sind (es handelt sich um das normale Ansprechen dieser Speicherstellen, wie es ständig auch vom herkömmlichen Basic aus geschieht). Die meisten Zeropage-Speicherplätze werden in derselben Weise benutzt, wie es Interpreter und Betriebssystem vorgeben. Nur dort, wo HIRES-3 eine andere Bedeutung einführt, ist das dann angegeben.

Speicher-stelle (hex.)	Normale Verwendung	Hires-3-Verwendung
01	Prozessor-Port	
02	leer	DEEK-Ergebnis Farbcodespeicher
10	Flagge: Sperren der Annahme von Integer- oder Feldvariablen	
14/15	Zwischenspeicher für Adressen und Integers	
22/23	Diverse Zeiger	
24/25	Zwischenspeicher bei NEXT	
26	Ergebnis bei Gleitkommarechungen (*,/)	Zwischenspeicher bei DUMP
2B/2C	Zeiger auf Basic-Start	
2D/2E	Zeiger auf Basic-Variablen-Start	
2F/30	Zeiger auf Basic-Array-Start	
37/38	Zeiger auf Basic-Speicher-Ende	
3D/3E	Zeiger: Aktueller Basic-Befehl (für CONT)	Zwischenspeicher für RENUMBER
3F/40	Zeiger: Aktuelle DATA-Zeile	Zwischenspeicher für RENUMBER
41/42	Zeiger: Nächstes DATA-Element	Zwischenspeicher für RENUMBER
43/44	Zeiger: Auswertung einer Eingabequelle (INPUT, GET, READ)	Zwischenspeicher für RENUMBER
45/46	Zeiger: Aktueller Basic-Variablen-Name	
47	Tabellenzeiger Zwischenspeicher	
47/48	Zeiger: Aktueller Basic-Variablen-Wert	
4E/4F	Zeiger: FN-Variable Auch als Zwischenspeicher bei Gleitkommaoperationen	
57-5B	Fließkommaregister (Polynomauswertung, TAN)	
58/59	Diverse Zeiger	
57/58		Ergebnis des UP 16-Bit-Multiplikation Diverse Zwischenspeicher
5C-60	Fließkommaregister (Zwischenspeicher Polynomauswertung)	
5D,5F	Diverse Zähler und Flaggen (TI$, 16-Bit-Multiplik., STRFAC, FACSTR)	
5F,60	Startadresse einer Programmzeile, Variablenverw., Stringtransfer	
60	Vorzeichen des Exponenten	
5C,5D		Ergebnis des UP 16-Bit-Division Diverse Zwischenspeicher
61-66	FAC	
69-6E	ARG	
7A/7B	CHRGET-Zeiger	
8B-8F	Gleitkommawert der letzten RND-Zahl	
8E,8F		UP 16-Bit-Division: Divisionsrest
B9	Aktuelle Sekundäradresse	
C5	SCNKEY (Aktueller Wert des Tastatur-Matrixcodes) = gedrückte Taste	
C6	Anzahl gültiger Zeichen im Tastaturpuffer	
CB	Tastatur-Matrixcode, Zeiger in Decodierungstabelle für Tastatur	
CE	Zeichen unter dem Cursor	Zwischenspeicher für RENUMBER
D1/D2	Zeiger: Aktueller Anfang der Cursor-Zeile im Bildschirm-RAM	
D3	Cursorspalte innerhalb der Cursorzeile	
FB-FE	leer	Diverse Zwischenspeicher und Zeiger

Die Hires-3-Verwendung ist nur dann angegeben, wenn sie von der normalen Verwendung abweicht.

Tabelle 8.1 Von HIRES-3 verwendete Zeropage-Adressen

Die Seiten 1 bis 3 unseres Speichers bergen ebenfalls wichtige Parameter unseres Computers. Deshalb ist die Tabelle 8.2 angefügt, die Auskunkt über Aktivitäten von HIRES-3 in diesem Bereich gibt.

Speicher-stelle (hex.)	Normale Verwendung	Hires-3-Verwendung
Page 1		
101 und folgende	Prozessorstack (auch für Datenspeicherung bei FOR...NEXT und GOSUB	Zwischenspeicher für RENUMBER
Page 2		
277 ff.	Tastaturpuffer	
288	Startseite der Videomatrix	
28D	Flagge für Shift, Commodore	
28F/290	Vektor: Tastaturabfrage	Wird verbogen auf 900B
2A7-2FF 2A7,2AF-2B5	leer	Diverse Zwischenspeicher und Zeiger
Page 3		
302/303	Vektor: Eingabewarteschleife (nach READY)	AUTONUMBER verbiegt nach $90BC und zurück zum Normalwert $A483
308/309	Vektor: Routinenaufruf (Interpreter)	Die Basic-Befehlserweiterungen verbiegen nach $9292 und zurück zum Normalwert $A7E4
314/315	IRQ-Vektor	UHR verbiegt nach $852D und zurück zum Normalwert $EA31

Die Hires-3-Verwendung ist nur dann angegeben, wenn sie von der normalen Verwendung abweicht.

Tabelle 8.2 Von HIRES-3 verwendete Adressen der Seiten 1 bis 3

Einige Zeiger werden 'verbogen' und durch HIRES-3 auch wieder gerade gerückt. Das geschieht zum Beispiel mittels AUS oder aber spätestens bei ‹RUN/STOP› und ‹RESTORE›.

Tabelle 8.3 gibt einen Überblick über alle durch HIRES-3 angesteuerten Interpreter-Routinen. Es wird jeweils die verwendete Einsprung-Adresse und die Bedeutung dieser Programmsegmente angegeben.

Einsprung-adresse (hex.)	Interpreter-Routine
A3BF	Block-Verschiebe-Routine
A437	Ausdruck einer Fehlermeldung und READY
A483	Eingabewarteschleife
A533	Neuberechnung des Linkpointers für Basic-Zeilen
A613	Berechnung der Startadresse einer Basic-Zeilennummer
A655	Zurücksetzen des CHRGET-Zeigers und Ausführen von CLR
A660	Ausführen von CLR
A68E	Zurücksetzen des CHRGET-Zeigers auf den Programmstart
A69C	LIST-Routine
A7AE	Interpreterschleife, Routinenaufruf
A7E7	Interpretation und Routinenaufruf, danach Interpreterschleife
A96B	Lesen eines Zeichens aus einer Basic-Textzeile und Umwandlung in 16-Bit-Integer
AB1E	Ausdrucken eines Strings
AD8A	Auswerten und Prüfen von Ausdrücken
AD9E	Auswerten von Ausdrücken
AEFD	Prüfen auf Komma
AF08	Ausdrucken von SYNTAX ERROR und Herstellen des READY-Zustandes
B113	Prüfung, ob Zeichen im Akku ein Buchstabe ist
B248	Ausdrucken von ILLEGAL QUANTITY ERROR und Herstellen des READY-Zustandes
B391	Umwandlung einer 16-Bit-Integer-Zahl in eine Gleitkommazahl
B3A2	Umwandlung einer 8-Bit-Integer-Zahl in eine Gleitkommazahl

Einsprung-adresse (hex.)	Interpreter-Routine
B44F	Alten Wert der FN-Variablen wiederherstellen = Abschluß der FN-Auswertung
B6A3	Stringverwaltungsroutine
B79B	Liest eine Zahl zwischen 0 und 255 aus dem Basic-Text ins X-Register ein
B79E	Wertet einen numerischen Ausdruck (0 bis 255) aus
B7EB	Lesen einer Adresse und eines Bytes aus dem Basic-Text
B7F7	Lesen einer Adresse (0 bis 65535) aus dem Basic-Text
B850	Gleitkommasubtraktion, Variante 1
B853	Gleitkommasubtraktion, Variante 2
B867	Gleitkommaaddition
BA28	Gleitkommamultiplikation
BB05	Gleitkommadivision, Variante 1
BB0F	Gleitkommadivision, Variante 2
BBA2	Konstante in den FAC laden als Gleitkommazahl
BBD4	FAC abspeichern
BC0C	FAC nach ARG übertragen
BC2B	Vorzeichen der Gleitkommazahl im FAC prüfen
BC44	Umwandlung Integerzahl in Gleitkommazahl im FAC
BC5B	Zahlenvergleich FAC mit Konstante
BC9B	Umwandlung einer Gleitkommazahl in eine 4-Byte-Integerzahl
BDCD	Ausdruck der laufenden Zeilennummer
BDDD	Umwandlung des FAC-Inhaltes in einen Zahlenstring

Tabelle 8.3 Von HIRES-3 verwendete Interpreter-Routinen

Aus zwei Gründen ist diese Tabelle interessant: Zum einen gibt sie Assembler-Programmierern eine kleine Orientierungshilfe für eigene Programme. Das kann in diesem Zusammenhang aber nur in relativ kurzer Form geschehen.

Zum anderen haben mich verschiedene Gerüchte erreicht, daß Commodore klammheimlich mehrere Änderungen des Betriebssystems und eventuell auch des Interpreters durchgeführt habe. Ich habe gerade ein neueres Betriebssystem untersucht, um die Unterschiede benennen zu können. Es könnte deswegen sein, daß im Falle eines Falles auch Routinen geändert wurden, die HIRES-3 verwendet. Ob dadurch Fehlfunktionen auftreten und wenn ja, welche, kann man kaum vorhersagen - es kommt eben sehr auf die Art der Veränderungen an. Testläufe auf verschiedenen C 64 gaben keinen Hinweis darauf: Alles lief - wie bei mir - einwandfrei.

Routinen aus dem oberen ROM-Bereich $E000 bis $FFFF, die HIRES-3 anwendet, zeigt die Tabelle 8.4.

Einsprung-adresse (hex.)	**ROM-Routine**
E1D4	Parameter für LOAD, SAVE, VERIFY aus Basic-Text lesen
E264	Basic-Funktion COS
E26B	Basic-Funktion SIN
E716	Ausgabe eines Zeichens auf den Bildschirm
EA31	IRQ-Routine
EB42	Abschluß der Tastaturabfrage
EB48	Umschaltung auf Decodierungstabelle per Shift, Commodore, CTRL
F6ED	Abfrage der RUN/STOP-Taste
Kernal-Routinen	
FFBA	SETLFS: Festsetzung der Parameter für OPEN
FFBD	SETNAM: Festsetzung des File-Namens
FFC0	OPEN: Spezifiziertes File setzen
FFC3	CLOSE: File schließen
FFC9	CHKOUT: Vorbereitung der File-Ausgabe
FFCC	CLRCHN: Schließen der aktiven Ein-/Ausgabekanäle
FFD2	CHROUT: Ausgabe des Akku-Inhaltes auf aktiven Ausgabekanal
FFD5	LOAD: Load und Verify von Programmen
FFD8	SAVE: Abspeichern von Programmen.

Tabelle 8.4 Von HIRES-3 verwendete Routinen aus dem oberen ROM-Bereich

Der untere Teil dieser Tabelle enthält die sogenannten Kernal-Sprungadressen. Für diesen Bereich garantiert Commodore, keine Änderungen durchzuführen. Leider - wie man aus dem vorangegangenen schnell ersehen kann - ist die Aufgabenvielfalt dieser Kernal-Adressen recht begrenzt. Man kommt ohne die anderen Routinen kaum aus.

In Tabelle 8.5 ist eine Liste aller Teile von HIRES-3 angegeben.

Startadresse	Unterprogramm	
9000	Belegung der Funktionstasten	
909A	Autonumber	
916B	Old	
91BA	Merge	
927A	Basic-Befehlserweiterungen	
93E7	DEEK	PEEK für RAM unter ROM
9410	16-Bit-Multiplikation	
9434	16-Bit-Division	
9522	FAR	Farbgebung für Hochauflösungsbild
9546	LOE	Bit-Map löschen
955D	HAN	Hochauflösung an
9572	HOF	Hochauflösung aus
9587	HFL	Kombination aus HAN, FAR und LOE
9590	PKT	Punkt setzen
961C	LIN	Linie zeichnen
97D2	REC	Rechteck zeichnen
9888	BLO	Rechteck ausfüllen
98FA	CIR	Ellipse (Kreis) zeichnen
99E8	RAD	Radiusvektor zeichnen
9A74	PAU	Pause machen
9AC4	HIS	Hochauflösungsbild extern abspeichern
9AE8	HIL	Hochauflösungsbild einladen
9AF7	LPK	Punkt löschen
9AFF	LLN	Linie löschen
9B07	LRE	Rechteck löschen
9B0F	LBK	Ausgefülltes Rechteck löschen
9B17	LKR	Ellipse (Kreis) löschen
9B1F	LRA	Radiusvektor löschen
9B27	TRS	Transformation

Startadresse	Unterprogramm	
9BA6	TPK	Punkt in transformiertes System zeichnen
9C0B	TLN	Linie in transformiertes System zeichnen
9CEB	TRE	Rechteck in transformiertes System zeichnen
8000	TBK	Ausgefülltes Rechteck in transformiertes System zeichnen
80E0	TKR	Ellipse (Kreis) in transformiertes System zeichnen
814C	TRA	Radiusvektor in transformiertes System zeichnen
81B8	LTP	Punkt im transformierten System löschen
81C0	LTL	Linie im transformierten System löschen
81C8	LTR	Rechteck im transformierten System löschen
81D0	LTB	Ausgefülltes Rechteck im transformierten System löschen
81D8	LTK	Ellipse (Kreis) im transformierten System löschen
81E0	LTV	Radiusvektor im transformierten System löschen
81E8	Renumber	
83DA	DUMP	Anzeige aller einfachen Variablen
849D	UHR	Anzeige einer Uhrzeit
85B8	FUNKT	Zeichnen einer vorher definierten Funktion im transformierten System
874F	LFUNK	Löschen einer vorher definierten Funktion im transformierten System
8757 87FE-8830	Rahmen mit Kopfzeile Texttabelle für Kopfzeile	
8831	Hardcopy des Hochauflösungsbildes auf Drucker Commodore 1526	

Tabelle 8.5 Die einzelnen Routinen von HIRES-3

Sollten Sie bei der Gelegenheit noch Bugs (Programmfehler) finden, dann würde ich mich über Informationen darüber freuen. Außerdem sollen Sie durch einige schwächere Programmteile animiert werden, sich bessere Lösungen auszudenken. Da gibt es viele Möglichkeiten. Veröffentlichungen - denke ich - können hervorragend dazu dienen, daß jeder etwas neues lernen kann, Anregungen für seine eigenen Programme erhält, etc.

In HIRES-3 sind viele solcher Anregungen verarbeitet. Als besonders hilfreich haben sich erwiesen:

1. Schneider, Eberl "Das Commodore 64-Buch" Band 1, Markt & Technik Verlag
2. "Computerspiele und Wissenswertes, Commodore 64" Markt & Technik Verlag
3. Angerhausen, Englisch, Gerits, "64 Tips & Tricks", Data Becker
4. H.J. Kutz, mc6(1984) S. 78f.

Murphys Gesetze gehen natürlich nicht an so einer guten Gelegenheit wie HIRES-3 vorbei. Wenn ich also weiter oben sagte, ich würde mich freuen, wenn Sie noch Bugs entdecken, ist das durchaus ernst gemeint.

Ich benütze HIRES-3 in dieser Form seit mehreren Monaten ausgiebig. Dasselbe tun einige meiner Freunde. Einer hat festgestellt, daß - aus welchen Gründen auch immer - sich Probleme ergeben können, wenn man direkt nach der Zeilennummer einen Doppelpunkt eingibt. Ich habe bisher noch keine Notwendigkeit gefunden, so etwas zu tun.

Sollten Sie's vorhaben oder aber gewohnt sein, achten Sie auf mögliche Störungen.

Nun noch zur Praxis: Bedenken Sie bitte beim Ausprobieren dieser neuen Befehle, daß Sie alle auch im Direktmodus verwenden können. Das kann - gerade beim Testen - mitunter ganz bequem sein. Allerdings sind die Befehlsworte auf dem Hochauflösungsbildschirm nur als farbige Quadrate zu erkennen (Warum? Siehe Kapitel 3). Wenn Sie mal das Gefühl haben, Sie hätten sich beim Eintippen eines Befehls vergallopiert, können Sie auch im hochauflösenden Modus den Bildschrim auf die gewohnte Weise mit ((SHIFT/CLR-HOME)) löschen. Sie werden dann immer noch das Hochauflösungsbild sehen (die Bit-Map wird ja nur durch LOE gelöscht). Mit dem FAR-Befehl bringen Sie wieder neue Farbe ins Bild. Sollte ein Programm im Hochauflösungsmodus auf einen Fehler laufen und aussteigen, kommen Sie mittels HOF leicht wieder zurück in den Normalmodus.

Zur Selbstkritik: Einige Befehle fehlen, nämlich etwas zum Ausfüllen von umrandeten Flächen, dann eine Möglichkeit, Texte ins Bild zu schreiben und anderes. Außerdem ist in puncto Geschwindigkeit noch lange nicht das Optimum erreicht... aber irgendetwas muß ja für HIRES-4 auch noch zu tun bleiben, oder?

```
PROGRAMM : HIR.*                8000 89B6
-----------------------------------------

8000 : A9 00 8D A8 94 20 FD AE  C1
8008 : 20 8A AD A9 6C A0 94 20  6C
8010 : 5B BC C9 FF D0 0B A9 00  48
8018 : 8D 9F 94 8D A0 94 4C 42  B0
8020 : B0 20 0C BC A9 6C A0 94  F5
8028 : 20 A2 BB 20 53 B8 A9 62  F3
8030 : A0 94 20 28 BA 20 9B BC  BC
8038 : A5 65 8D 9F 94 A5 64 8D  0A
8040 : A0 94 20 FD AE 20 8A AD  64
8048 : A9 71 A0 94 20 5B BC C9  C8
8050 : 01 D0 08 A9 00 8D A1 94  0D
8058 : 4C 71 80 A9 71 A0 94 20  61
8060 : 50 B8 A9 67 A0 94 20 28  E3
8068 : BA 20 9B BC A5 65 8D A1  B0
8070 : 94 20 FD AE 20 8A AD A9  CA
8078 : 6C A0 94 20 5B BC C9 FF  20
8080 : D0 0B A9 00 8D A2 94 8D  9C
8088 : A3 94 4C AE 80 20 0C BC  11
8090 : A9 6C A0 94 20 A2 BB 20  70
8098 : 53 B8 A9 62 A0 94 20 28  7D
80A0 : BA 20 9B BC A5 65 8D A2  EA
80A8 : 94 A5 64 8D A3 94 20 FD  35
80B0 : AE 20 8A AD A9 71 A0 94  98
80B8 : 20 5B BC C9 01 D0 08 A9  F8
80C0 : 00 8D A4 94 4C DD 80 A9  4B
80C8 : 71 A0 94 20 50 B8 A9 67  F3
80D0 : A0 94 20 28 BA 20 9B BC  5C
80D8 : A5 65 8D A4 94 4C B3 98  D3
80E0 : A9 00 8D A8 94 20 FD AE  A1
80E8 : 20 8A AD 20 0C BC A9 6C  E3
80F0 : A0 94 20 A2 BB 20 53 B8  B2
80F8 : A9 62 A0 94 20 28 BA 20  FB
8100 : 9B BC A5 65 8D A5 94 A5  B3
8108 : 64 8D A6 94 20 FD AE 20  5C
8110 : 8A AD A9 71 A0 94 20 50  D9
8118 : B8 A9 67 A0 94 20 28 BA  F3
8120 : 20 9B BC A5 65 8D A7 94  7C
8128 : 20 FD AE 20 8A AD A9 62  78
8130 : A0 94 20 28 BA A0 94 A2  70
8138 : 79 20 D4 BB 20 FD AE 20  5B
8140 : 8A AD A9 67 A0 94 20 28  78
8148 : BA 4C 25 99 A9 00 8D A8  C7
8150 : 94 20 FD AE 20 8A AD 20  97
8158 : 0C BC A9 6C A0 94 20 A2  2F
8160 : BB 20 53 B8 A9 62 A0 94  71
8168 : 20 28 BA 20 9B BC A5 65  50
8170 : 8D 92 94 A5 64 8D 93 94  4A
8178 : 20 FD AE 20 8A AD A9 71  E6
8180 : A0 94 20 50 B8 A9 67 A0  34
8188 : 94 20 28 BA 20 9B BC A5  AB
8190 : 65 8D 94 94 20 FD AE 20  60
8198 : 8A AD A9 62 A0 94 20 28  2F
81A0 : BA A0 94 A2 79 20 D4 BB  87
81A8 : 20 FD AE 20 8A AD A9 67  02
81B0 : A0 94 20 28 BA 4C 13 9A  37
81B8 : A9 FF 8D A8 94 4C AB 9B  6B

81C0 : A9 FF 8D A8 94 4C 10 9C  06
81C8 : A9 FF 8D A8 94 4C F0 9C  92
81D0 : A9 FF 8D A8 94 4C 05 80  B2
81D8 : A9 FF 8D A8 94 4C E5 80  3D
81E0 : A9 FF 8D A8 94 4C 51 81  F5
81E8 : 20 AD 83 86 41 85 42 20  1A
81F0 : B8 83 20 AD 83 86 3D 85  94
81F8 : 3E E4 41 E5 42 B0 05 A2  B8
8200 : 0E 4C 37 A4 20 AD 83 86  21
8208 : 3F 85 40 A0 01 B1 43 F0  BB
8210 : 37 A9 FF 85 15 85 14 20  DA
8218 : 6B 83 20 8E A6 B0 32 A0  19
8220 : 00 B1 43 AA C8 B1 43 F0  28
8228 : 1F 48 C8 A5 3D 91 43 C8  51
8230 : A5 3E 91 43 86 43 68 85  F0
8238 : 44 A5 3D 18 65 3F 85 3D  82
8240 : A5 3E 65 40 85 3E 90 D7  A2
8248 : 4C 60 A6 E6 7A D0 02 E6  4F
8250 : 7B A0 00 B1 7A D0 13 A0  0D
8258 : 02 B1 7A F0 C2 A5 7A 18  63
8260 : 69 05 85 7A 90 EB E6 7B  F7
8268 : B0 E7 C9 22 D0 0B 20 73  8F
8270 : 00 C9 00 F0 E2 C9 22 D0  1A
8278 : F5 C9 89 F0 17 C9 8D F0  AA
8280 : 13 C9 A7 F0 0F C9 8A F0  CB
8288 : 0B C9 CB D0 BE 20 73 00  3F
8290 : C9 A4 D0 BF 20 73 00 B0  D6
8298 : BA 84 14 84 15 E9 2F 90  A8
82A0 : 33 AA A5 15 85 22 C9 19  F7
82A8 : B0 ED A5 14 0A 26 22 0A  A9
82B0 : 26 22 65 14 85 14 A5 22  97
82B8 : 65 15 85 15 06 14 26 15  70
82C0 : 8A 65 14 85 14 90 02 E6  4E
82C8 : 15 C8 B1 7A C9 20 F0 F9  52
82D0 : C9 3A 90 C9 A5 14 C5 41  A8
82D8 : A5 15 E5 42 90 48 84 CE  C4
82E0 : 20 6B 83 38 A2 90 20 44  55
82E8 : BC 20 DD BD A0 FF C8 B9  84
82F0 : 01 01 D0 FA A5 7A A6 7B  C5
82F8 : 85 5F 86 60 38 98 E5 CE  58
8300 : 30 26 F0 40 85 CE A5 2D  47
8308 : 85 5A 18 65 CE 85 58 A5  33
8310 : 2E 85 5B 69 00 85 59 C5  22
8318 : 38 90 06 A5 58 C5 37 B0  C1
8320 : 43 20 BF A3 F0 1E 90 30  7A
8328 : 49 FF A8 C8 A2 00 A5 5F  33
8330 : C5 2D A5 60 E5 2E B0 0C  AC
8338 : B1 5F 81 5F E6 5F D0 EE  70
8340 : E6 60 D0 EA A0 00 B9 01  DB
8348 : 01 F0 05 91 7A C8 D0 F6  54
8350 : 20 B8 83 20 73 00 90 FB  23
8358 : A0 00 C9 2C D0 02 A9 B9  C7
8360 : AA 4C 55 82 A9 45 20 D2  C1
8368 : FF D0 E8 A5 3D A6 3E 85  CB
8370 : 63 86 62 A5 43 A6 44 85  E9
8378 : 22 86 23 A0 02 B1 22 C5  7C
```

```
83B0 : 14 C8 B1 22 E5 15 B0 2F D1
83B8 : A0 00 B1 22 AA C8 B1 22 D5
83C0 : 86 22 85 23 B1 22 F0 1F 1B
83B8 : A5 63 65 3F 85 63 A5 62 FF
83A0 : 65 40 85 62 B0 04 C9 FA 1B
83A8 : 90 D1 4C FF B1 20 FD AE A2
83B0 : 20 6B A9 A6 14 A5 15 60 48
83B8 : 20 33 A5 18 A5 22 69 02 F3
83C0 : 85 2D A5 23 69 00 85 2E B3
83C8 : A5 41 A6 42 85 14 86 15 3D
83D0 : 20 13 A6 A5 5F 85 43 86 14
83D8 : 44 60 A5 2D A4 2E 85 14 55
83E0 : 84 15 C4 30 D0 02 C5 2F B9
83E8 : B0 18 69 02 90 01 C8 85 7E
83F0 : 22 84 23 20 20 84 20 54 71
83F8 : 84 8A 10 07 20 5D 84 4C 3E
8400 : 0F 84 60 98 30 06 20 6D 0B
8408 : 84 4C 0F 84 20 76 84 A9 22
8410 : 0D 20 D2 FF A5 14 A4 15 9A
8418 : 18 69 07 90 C1 C8 B0 BE 5B
8420 : A0 00 B1 14 AA 29 7F 20 E1
8428 : D2 FF C8 B1 14 A8 29 7F 8C
8430 : F0 03 20 D2 FF 8A 10 11 BB
8438 : 98 30 0A A9 21 20 D2 FF FE
8440 : 68 68 4C 0F 84 A9 25 D0 9D
8448 : 4E 98 10 04 A9 24 D0 47 F4
8450 : 60 20 D2 FF A9 20 20 D2 37
8458 : FF A9 3D D0 3A A0 00 B1 A1
8460 : 22 AA C8 B1 22 A8 8A 20 11
8468 : 95 B3 4C 70 84 20 A6 BB 53
8470 : 20 DD BD 4C 1E AB 20 95 63
8478 : 84 A0 02 B1 22 85 25 88 F7
8480 : B1 22 85 24 88 B1 22 85 D2
8488 : 26 F0 0A B1 24 20 D2 FF 6E
8490 : C8 C4 26 D0 F6 A9 22 4C 3C
8498 : D2 FF 00 00 00 AD 0E DC C9
84A0 : 09 80 8D 0E DC AD 0F DC 40
84A8 : 29 7F 8D 0F DC 20 79 00 8B
84B0 : F0 65 20 FD AE 20 9E AD DC
84B8 : 20 A3 B6 C9 06 D0 6B A0 66
84C0 : 00 B1 22 38 E9 30 C9 03 76
84C8 : B0 60 0A 0A 0A 0A 85 FB 6B
84D0 : C8 B1 22 38 E9 30 C9 0A 5C
84D8 : B0 50 05 FB D0 04 A9 92 6A
84E0 : D0 0F C9 24 B0 44 C9 13 A9
84E8 : 90 07 38 F8 E9 12 D8 09 CE
84F0 : 80 8D 0B DC 20 9A 85 8D 9D
84F8 : 0A DC 20 9A 85 8D 09 DC 6E
8500 : A9 00 8D 08 DC 20 79 00 C2
8508 : F0 0D 20 FD AE 20 9E B7 1C
8510 : E0 10 B0 16 8E A7 02 78 06
8518 : AD 14 03 49 1C 8D 14 03 3E
8520 : AD 15 03 49 6F 8D 15 03 00
8528 : 58 60 4C 48 B2 A5 FB 48 A5
8530 : A5 FC 48 AD 88 02 85 FC C4
8538 : A9 00 85 FB A0 1E AD 0B 8A
8540 : DC C9 12 F0 11 C9 80 90 26
8548 : 0F 29 7F C9 12 F0 09 F8 C4

8550 : 18 69 12 D8 D0 02 A9 00 80
8558 : 20 78 85 AD 0A DC 20 78 C4
8560 : 85 AD 09 DC 20 78 85 AD D1
8568 : 08 DC 09 30 20 90 85 68 94
8570 : 85 FC 68 85 FB 4C 31 EA FB
8578 : 48 29 F0 4A 4A 4A 4A 1B 2B
8580 : 69 30 20 90 85 68 29 0F 7A
8588 : 18 69 30 20 90 85 A9 3A B5
8590 : 91 FB AD A7 02 99 00 D8 1E
8598 : C8 60 C8 B1 22 38 E9 30 E5
85A0 : C9 06 B0 86 0A 0A 0A 0A 96
85A8 : 85 FB C8 B1 22 38 E9 30 7F
85B0 : C9 0A B0 EE 05 FB 60 00 3A
85B8 : A9 00 8D A8 94 20 FD AE 79
85C0 : 20 79 00 20 13 B1 B0 03 28
85C8 : 4C 0B AF 09 80 48 E6 7A 82
85D0 : A5 7A D0 02 E6 7B 20 FD ED
85D8 : AE 20 8A AD A9 6C A0 94 98
85E0 : 20 5B BC C9 FF D0 07 A9 0C
85E8 : 6C A0 94 20 A2 BB A2 79 53
85F0 : A0 94 20 D4 BB 20 17 86 A3
85F8 : 8D 92 94 8C 93 94 20 FD DF
8600 : AE 20 8A AD A2 7E A0 94 E0
8608 : 20 D4 BB 20 17 86 8D 95 8C
8610 : 94 8C 96 94 4C 33 86 20 DB
8618 : 0C BC A9 6C A0 94 20 A2 EF
8620 : BB 20 53 B8 A9 62 A0 94 31
8628 : 20 2B BA 20 9B BC A5 65 10
8630 : A4 64 60 68 85 45 85 10 E4
8638 : A0 00 84 46 A5 2D A6 2E 7D
8640 : 86 60 85 5F E4 30 D0 04 5F
8648 : C5 2F F0 19 A5 45 D1 5F 8F
8650 : D0 08 A5 46 C8 D1 5F F0 D1
8658 : 11 88 18 A5 5F 69 07 90 E7
8660 : E1 E8 4C 40 86 A2 1B 4C 53
8668 : 37 A4 A5 5F 18 69 02 A4 65
8670 : 60 90 01 C8 85 4E 84 4F ED
8678 : A0 02 B1 4E 85 47 C8 B1 69
8680 : 4E 85 48 C8 B1 47 48 88 43
8688 : 10 FA A9 79 A0 94 20 A2 23
8690 : BB A5 7A 48 A5 7B 48 A5 68
8698 : 47 48 A5 48 48 A5 4E 48 F1
86A0 : A5 4F 48 A0 00 68 85 4F 0B
86A8 : 68 85 4E B1 4E 85 7A C8 29
86B0 : B1 4E 85 7B 68 85 48 A8 7E
86B8 : 68 85 47 AA 20 D4 BB A5 ED
86C0 : 47 48 A5 48 48 A5 4E 48 19
86C8 : A5 4F 48 20 8A AD A9 71 CB
86D0 : A0 94 20 5B BC C9 01 D0 EE
86D8 : 05 A2 00 4C FC B6 A9 71 45
86E0 : A0 94 20 50 B8 A9 67 A0 94
86E8 : 94 20 2B BA 20 9B BC A5 0B
86F0 : 65 AA AD 92 94 85 14 AD 89
86F8 : 93 94 85 15 20 9F 95 18 5F
8700 : AD 92 94 69 01 8D 92 94 39
8708 : AD 93 94 69 00 8D 93 94 B5
8710 : CD 96 94 D0 08 AD 92 94 C9
8718 : CD 95 94 F0 1D AD 93 94 AA
```

```
8720 : AC 92 94 20 91 B3 20 0C 8E
8728 : BC A9 62 A0 94 20 05 BB 3B
8730 : A9 6C A0 94 20 67 B8 4C 83
8738 : A3 86 68 85 4F 68 85 4E D4
8740 : 68 85 4F 68 85 4E 68 85 C3
8748 : 7B 68 85 7A 4C 4F B4 A9 0D
8750 : FF 8D A8 94 4C BD 85 A9 EF
8758 : 4F A2 00 BD 00 04 8A 8D 10
8760 : 00 D8 A9 50 8D 27 04 8A 78
8768 : 8D 27 D8 A9 4C 8D C0 07 36
8770 : 8A 8D C0 D8 A9 7A 8D E7 E1
8778 : 07 8A 8D E7 D8 A0 00 A9 3B
8780 : 77 99 01 04 8A 99 01 D8 B0
8788 : C8 C0 26 D0 F2 A9 6F 99 C1
8790 : C0 07 8A 99 C0 D8 88 D0 58
8798 : F4 A9 74 8D 28 04 8A 8D 18
87A0 : 28 D8 A9 4F 85 FB 85 FD D3
87A8 : A9 04 85 FC A9 D8 85 FE CA
87B0 : A9 6A 91 FB A9 00 91 FD 4F
87B8 : C8 A9 74 91 FB A9 00 91 D4
87C0 : FD A0 00 E8 E0 16 F0 1D E7
87C8 : 18 A5 FB 69 28 85 FB A5 C9
87D0 : FC 69 00 85 FC 18 A5 FD 55
87D8 : 69 28 85 FD A5 FE 69 00 6E
87E0 : 85 FE 4C B0 87 A9 6A 8D 98
87E8 : BF 07 A9 00 8D BF D8 A0 1D
87F0 : 00 B9 FE 87 F0 07 20 D2 EB
87F8 : FF C8 4C F1 87 60 13 90 95
8800 : 12 20 20 48 45 49 4D 4F A6
8808 : 20 50 4F 4E 4E 41 54 48 BF
8810 : 20 20 20 56 45 52 53 49 DA
8818 : 4F 4E 20 31 34 2E 30 36 9E
8820 : 2E 31 39 38 34 20 20 20 41
8828 : 20 92 00 FF FF FF FF FF 91
8830 : FF A5 01 48 A9 37 85 01 B8
8838 : A9 7C A2 04 A0 00 84 FB 5D
8840 : 84 FC 8C A7 02 8C B0 02 A6
8848 : 8C B1 02 8C B2 02 8C B3 94
8850 : 02 8C B4 02 8C AF 02 8C 6D
8858 : B5 02 20 BA FF 98 20 BD 2E
8860 : FF 20 C0 FF A9 7D A2 04 B8
8868 : A0 05 20 BA FF 20 C0 FF EE

8870 : A9 7E A2 04 A0 06 20 BA B2
8878 : FF 20 C0 FF A2 7E 20 C9 EA
8880 : FF A9 15 20 D2 FF 20 CC E4
8888 : FF 20 79 00 C9 2C D0 14 5F
8890 : 20 73 00 20 8A AD 20 F7 F4
8898 : B7 A5 14 A4 15 85 FB 84 32
88A0 : FC 18 90 18 AD 00 DD 49 B4
88A8 : FF 29 03 18 6A 6A 6A 85 AE
88B0 : FC AD 18 D0 29 08 0A 0A B2
88B8 : 05 FC 85 FC 20 46 89 A2 DC
88C0 : 7D 20 C9 FF A2 00 A0 00 6C
88C8 : BD A7 02 F0 01 C8 20 D2 74
88D0 : FF E8 E0 08 D0 F2 20 CC 3B
88D8 : FF A2 7C 20 C9 FF C0 00 EB
88E0 : F0 12 AE AF 02 F0 08 A9 96
88E8 : 20 20 D2 FF CA D0 F8 A9 37
88F0 : FE 20 D2 FF EE AF 02 AD 83
88F8 : AF 02 C9 28 D0 0D EE B5 BC
8900 : 02 A9 00 8D AF 02 A9 0D 54
8908 : 20 D2 FF A9 8D 20 D2 FF EB
8910 : 20 CC FF A5 FB 18 6D B0 E2
8918 : 02 85 FB 90 02 E6 FC AD 94
8920 : B5 02 C9 19 D0 96 A2 7E B5
8928 : 68 85 01 20 C9 FF A9 24 23
8930 : 20 D2 FF 20 CC FF A9 7C 2A
8938 : 20 C3 FF A9 7D 20 C3 FF 57
8940 : A9 7E 20 C3 FF 60 A9 00 52
8948 : 8D B0 02 A9 00 8D B2 02 1E
8950 : F0 45 D0 12 AD B2 02 AC 2B
8958 : B0 02 99 A7 02 C8 8C B0 5F
8960 : 02 C0 08 D0 E6 60 78 A9 85
8968 : 30 85 01 A9 00 8D B1 02 08
8970 : A9 01 D0 23 AC B1 02 B1 F6
8978 : FB 2D B3 02 F0 0A AD B2 B2
8980 : 02 18 6D B4 02 8D B2 02 DC
8988 : C8 8C B1 02 C0 08 D0 E0 94
8990 : A9 37 85 01 58 D0 BD A0 9B
8998 : 01 48 AA A9 07 38 FD B0 29
89A0 : 02 F0 07 AA 98 0A CA D0 D8
89A8 : FC A8 68 AA 98 9D B3 02 B1
89B0 : E0 01 D0 9E F0 BE FF FF 1E
```

```
PROGRAMM : HIRES3.1.*     9000 9DCB
-------------------------------------
9000 : A9 0B A0 90 8D 8F 02 8C  DF
9008 : 90 02 60 A2 06 E4 CB F0  9E
9010 : 08 CA E0 02 D0 F7 4C 48  84
9018 : EB E4 C5 F0 F9 86 C5 AD  4B
9020 : 8D 02 C9 01 D0 04 E8 E8  E3
9028 : E8 E8 D8 A9 00 E0 03 F0  E5
9030 : 08 18 69 09 CA E0 03 D0  21
9038 : F8 AA A0 00 C8 BD 51 90  8E
9040 : 99 76 02 C9 0D F0 05 E8  0C
9048 : C0 09 30 F0 84 C6 4C 42  EB
9050 : EB 53 59 53 34 39 31 35  E2
9058 : 32 0D 52 55 4E 0D 00 00  9D
9060 : 00 00 00 53 59 53 33 37  36
9068 : 34 39 38 0D 53 59 53 33  9C
9070 : 37 30 31 38 0D 53 59 53  8A
9078 : 32 38 36 37 32 0D 53 59  C6
9080 : 53 33 33 32 35 36 00 53  2C
9088 : 59 53 33 37 32 32 37 0D  EA
9090 : 53 59 53 33 37 33 30 36  05
9098 : 0D FF A9 00 8D A4 90 8D  6B
90A0 : A5 90 F0 04 DC 05 AD 11  19
90A8 : A6 2D 8E A6 90 A6 2E 8E  71
90B0 : A7 90 A9 BC 8D 02 03 A9  E9
90B8 : 90 8D 03 03 A6 2D EC A6  05
90C0 : 90 D0 14 A6 2E EC A7 90  9C
90C8 : D0 0D AE A4 90 D0 05 AE  60
90D0 : A5 90 F0 03 4C 5D 91 A6  9D
90D8 : 2D 8E A6 90 A6 2E 8E A7  6D
90E0 : 90 A6 2D CA CA E4 2B D0  8A
90E8 : 06 A6 2E E4 2C F0 3D A6  F6
90F0 : 2B 86 FB 86 14 A6 2C 86  62
90F8 : FC 86 15 A0 00 B1 FB D0  B0
9100 : 14 C8 B1 FB D0 0F C8 B1  70
9108 : 14 8D A4 90 C8 B1 14 8D  A3
9110 : A5 90 4C 2C 91 A5 FB 85  D7
9118 : 14 A5 FC 85 15 A0 00 B1  A8
9120 : FB AA C8 B1 FB 86 FB 85  C7
9128 : FC 4C FB 90 AD A4 90 18  CE
9130 : 69 0A 8D A4 90 AD A5 90  C4
9138 : 69 00 8D A5 90 AE A4 90  EC
9140 : 20 CD BD A9 20 20 D2 FF  3A
9148 : A0 00 B1 D1 99 77 02 E6  BA
9150 : C6 C8 C4 D3 D0 F4 A9 00  81
9158 : 85 D3 4C 83 A4 A9 83 8D  0B
9160 : 02 03 A9 A4 8D 03 03 4C  7B
9168 : 83 A4 00 A5 2B 18 69 04  13
9170 : 85 FD A5 2C 69 00 85 FE  8D
9178 : A0 00 B1 FD F0 08 C8 C0  38
9180 : 58 D0 F7 4C 08 AF C8 98  1A
9188 : A0 00 18 65 FD 91 2B 85  FF
9190 : FD 90 02 E6 FE A5 FE C8  DD
9198 : 91 2B 88 B1 FD AA C8 B1  D3
91A0 : FD F0 07 85 FE 86 FD 38  14
91A8 : B0 F0 A5 FD 18 69 02 85  D9
91B0 : 2D A5 FE 20 55 A6 4C 9C  68
91B8 : A6 00 A9 FF 85 02 A5 2B  1E
91C0 : C9 01 D0 0D 85 FB A5 2C  07
91C8 : C9 08 D0 05 85 FC 38 B0  ED
91D0 : 16 A5 FB 85 2B A5 FC 85  47
91D8 : 2C A2 00 BD 45 92 F0 06  C6
91E0 : 20 16 E7 E8 D0 F5 60 18  91
91E8 : A0 00 B1 2B D0 0C C8 B1  4E
91F0 : 2B D0 07 C8 B1 2B D0 02  1A
91F8 : 85 02 E6 2B D0 02 E6 2C  AE
9200 : A5 02 D0 E4 A2 00 BD 12  BC
9208 : 92 F0 06 20 16 E7 E8 D0  7E
9210 : F5 60 0D 0D 20 20 20 20  DE
9218 : 20 2A 2A 2A 2A 20 4D 45  81
9220 : 52 47 45 20 2A 2A 2A 2A  5C
9228 : 2A 0D 42 45 52 45 49 54  2F
9230 : 20 5A 55 4D 20 4B 4F 50  B7
9238 : 50 45 4C 4E 21 0D 00 00  82
9240 : 00 00 00 00 00 0D 0D 20  1D
9248 : 20 20 20 20 2A 2A 2A 2A  75
9250 : 20 4D 45 52 47 45 20 2A  26
9258 : 2A 2A 2A 2A 0D 50 52 4F  A2
9260 : 47 52 41 4D 4D 45 20 47  D8
9268 : 45 4B 4F 50 50 45 4C 54  3A
9270 : 21 0D 00 00 00 00 00 00  18
9278 : 00 00 A9 92 8D 08 03 A9  AD
9280 : 92 8D 09 03 60 A9 E4 8D  7E
9288 : 08 03 A9 A7 8D 09 03 60  5F
9290 : 28 5F 20 73 00 90 1E C9  6F
9298 : 60 B0 1A C9 41 90 15 8D  18
92A0 : 91 92 A2 00 8E 90 92 A0  1C
92A8 : 00 EE 90 92 BD 40 93 D0  63
92B0 : 07 AD 91 92 38 4C E7 A7  19
92B8 : D1 7A D0 28 C8 E8 BD 40  4B
92C0 : 93 D0 F5 18 98 65 7A 85  E5
92C8 : 7A 90 02 E6 7B AD 90 92  74
92D0 : 0A AA BD EE 92 8D DF 92  B7
92D8 : BD EF 92 8D E0 92 20 BC  80
92E0 : 94 4C AE A7 E8 BD 40 93  DF
92E8 : D0 FA E8 4C A7 92 E7 A7  F7
92F0 : B0 94 B3 94 B6 94 B9 94  8A
92F8 : BC 94 74 9A E7 93 BF 94  B2
9300 : C2 94 C5 94 C8 94 CB 94  9A
9308 : CE 94 D1 94 D4 94 D7 94  A2
9310 : DA 94 DD 94 E0 94 E3 94  AA
9318 : E6 94 E9 94 EC 94 EF 94  B2
9320 : F2 94 F5 94 F8 94 FB 94  B9
9328 : FE 94 01 95 04 95 07 95  98
9330 : 0A 95 0D 95 85 92 DA 83  5A
9338 : 9D 84 B8 85 4F 87 E7 A7  16
9340 : 48 46 4C 00 48 41 4E 00  86
9348 : 46 41 52 00 4C 4F 45 00  18
9350 : 48 4F 46 00 50 41 55 00  36
9358 : 44 45 45 4B 00 50 4B 54  52
9360 : 00 4C 49 4E 00 52 45 43  D1
9368 : 00 42 4C 4F 00 43 49 52  6A
```

```
9370 : 00 52 41 44 00 48 49 53  80
9378 : 00 48 49 4C 00 4C 50 4B  B2
9380 : 00 4C 4C 4E 00 4C 52 45  B9
9388 : 00 4C 42 4B 00 4C 4B 52  DD
9390 : 00 4C 52 41 00 54 52 53  06
9398 : 00 54 50 4B 00 54 4C 4E  B0
93A0 : 00 54 52 45 00 54 42 4B  4A
93A8 : 00 54 4B 52 00 54 52 41  5E
93B0 : 00 4C 54 50 00 4C 54 4C  42
93B8 : 00 4C 54 52 00 4C 54 42  76
93C0 : 00 4C 54 4B 00 4C 54 56  C5
93C8 : 00 41 55 53 00 44 55 4D  3B
93D0 : 50 00 55 48 52 00 46 55  68
93D8 : 4E 4B 54 00 4C 46 55 4E  CA
93E0 : 4B 00 00 00 00 00 00 A5  77
93E8 : 15 48 A5 14 48 20 FD AE  E8
93F0 : 20 8A AD 20 F7 B7 A5 01  9A
93F8 : 48 29 FC 78 85 01 A0 00  06
9400 : B1 14 A8 68 85 01 58 68  85
9408 : 85 14 68 85 15 84 02 60  A0
9410 : A9 00 85 57 85 58 A0 08  B3
9418 : 18 26 5B 90 0D 18 A5 57  03
9420 : 65 59 85 57 A5 58 65 5A  E5
9428 : 85 58 88 F0 06 26 57 26  55
9430 : 58 90 E6 60 A2 00 86 8E  F7
9438 : 86 8F A0 10 06 57 26 58  14
9440 : 26 8E 26 8F 38 A5 8E E5  E0
9448 : 59 AA A5 8F E5 5A 90 06  D1
9450 : 86 8E 85 8F E6 57 88 D0  5D
9458 : E3 A5 57 85 5C A5 58 85  F4
9460 : 5D 60 85 36 49 24 92 87  25
9468 : 04 AA AA AB 83 E0 00 00  21
9470 : 00 81 40 00 00 00 00 00  41
9478 : 05 81 C0 00 00 00 86 3C  01
9480 : 00 00 00 83 49 88 2B 74  5F
9488 : 7B 2E 4C 41 5D 83 49 0F  8B
9490 : DA A1 33 01 75 34 01 6A  FA
9498 : 01 00 FF 0B B8 00 0C 01  B8
94A0 : 00 01 3E 01 C6 A0 00 35  AC
94A8 : 00 88 47 00 00 00 00 00  BE
94B0 : 4C 87 95 4C 5D 95 4C 22  A7
94B8 : 95 4C 46 95 4C 72 95 4C  FF
94C0 : 90 95 4C 1C 96 4C D2 97  F8
94C8 : 4C 88 98 4C FA 98 4C E8  7F
94D0 : 99 4C C4 9A 4C E8 9A 4C  23
94D8 : F7 9A 4C FF 9A 4C 07 9B  8F
94E0 : 4C 0F 9B 4C 17 9B 4C 1F  E2
94E8 : 9B 4C 27 9B 4C A6 9B 4C  E8
94F0 : 0B 9C 4C EB 9C 4C 00 80  07
94F8 : 4C E0 80 4C 4C 81 4C B8  D1
9500 : 81 4C C0 81 4C C8 81 4C  B1
9508 : D0 81 4C D8 81 4C E0 81  C8
9510 : 4C AC 94 4C AC 94 4C AC  5B
9518 : 94 4C AC 94 4C AC 94 4C  A5
9520 : AC 94 20 9B B7 8A 0A 0A  9E
9528 : 0A 0A 85 02 20 9B B7 8A  AC
9530 : 05 02 A0 00 99 00 8C 99  5D

9538 : FA 8C 99 F4 8D 99 EE 8E  FC
9540 : C8 C0 FA D0 EF 60 A9 A0  2B
9548 : 85 FE A9 00 85 FD A8 91  C5
9550 : FD C8 D0 FB E6 FE A4 FE  5C
9558 : C0 C0 D0 F2 60 A9 95 8D  CF
9560 : 00 DD A9 38 8D 18 D0 A9  F0
9568 : 8C 8D 88 02 A9 3B 8D 11  EA
9570 : D0 60 A9 04 8D 88 02 A9  D4
9578 : 15 8D 18 D0 A9 97 8D 00  01
9580 : DD A9 1B 8D 11 D0 60 20  04
9588 : 22 95 20 46 95 4C 5D 95  A2
9590 : A9 00 8D A8 94 20 FD AE  51
9598 : 20 EB B7 EA EA EA EA 18  DB
95A0 : E0 C8 B0 6B A5 15 F0 0A  59
95A8 : C9 01 D0 63 A5 14 C9 40  35
95B0 : B0 5D A5 14 29 07 A8 38  D9
95B8 : A9 00 6A 88 10 FC 4D A8  7C
95C0 : 94 48 A5 14 29 F8 85 14  FD
95C8 : 8A 29 07 18 65 14 85 14  E1
95D0 : A5 15 69 A0 85 15 8A 29  EC
95D8 : F8 85 59 A9 00 85 5A A9  07
95E0 : 28 85 5B 20 10 94 18 A5  F7
95E8 : 14 65 57 85 14 A5 15 65  C3
95F0 : 58 85 15 68 A0 00 85 02  81
95F8 : A5 01 48 29 FE 78 85 01  21
9600 : A5 02 2C A8 94 30 0C 11  E4
9608 : 14 91 14 68 85 01 58 EA  8F
9610 : EA EA 60 31 14 91 14 68  9D
9618 : 85 01 58 60 A9 00 8D A8  62
9620 : 94 20 FD AE 20 EB B7 8E  77
9628 : 94 94 A5 14 8D 92 94 A5  FD
9630 : 15 8D 93 94 20 FD AE 20  70
9638 : EB B7 8E 97 94 A6 14 8E  81
9640 : 95 94 A5 15 8D 96 94 AE  69
9648 : 95 94 AD 96 94 CD 93 94  95
9650 : 30 0B F0 02 B0 2B EC 92  BF
9658 : 94 F0 6A B0 24 AD 92 94  38
9660 : AE 95 94 8D 95 94 8E 92  0D
9668 : 94 AD 93 94 AE 96 94 8D  57
9670 : 96 94 8E 93 94 AD 94 94  99
9678 : AE 97 94 8D 97 94 8E 94  4A
9680 : 94 38 AD 95 94 ED 92 94  7A
9688 : 8D 98 94 AD 96 94 ED 93  29
9690 : 94 8D 99 94 A2 00 8E 9A  7D
9698 : 94 38 AD 97 94 ED 94 94  DB
96A0 : F0 4F EE 9A 94 8D 9B 94  94
96A8 : B0 0B 49 FF 69 01 CA 8E  17
96B0 : 9A 94 8D 9B 94 AD 99 94  B1
96B8 : D0 62 AD 98 94 CD 9B 94  87
96C0 : B0 5A 4C 7D 97 AD 97 94  CE
96C8 : CD 94 94 B0 09 AE 94 94  9C
96D0 : 8E 97 94 8D 94 94 AE 94  D2
96D8 : 94 8A 48 AD 92 94 85 14  85
96E0 : AD 93 94 85 15 20 9B 95  19
96E8 : 68 AA E8 EC 97 94 90 E9  B1
96F0 : 60 AD 92 94 85 14 AD 93  35
96F8 : 94 85 15 AE 97 94 20 9B  40
```

```
9700 : 95 EE 92 94 D0 03 EE 93  4B
9708 : 94 AD 96 94 CD 93 94 90  98
9710 : 0A D0 DE AD 95 94 CD 92  4A
9718 : 94 B0 D6 60 A9 00 8D 9C  D0
9720 : 94 8D 9D 94 18 AD 9C 94  FF
9728 : 85 59 6D 92 94 85 14 AD  29
9730 : 9D 94 85 5A 6D 93 94 85  95
9738 : 15 AD 9B 94 85 5B 20 10  71
9740 : 94 AD 98 94 85 59 AD 99  70
9748 : 94 85 5A 20 34 94 AD 94  01
9750 : 94 2C 9A 94 30 05 18 65  8A
9758 : 5C 90 03 38 E5 5C AA 20  F0
9760 : 9B 95 EE 9C 94 D0 03 EE  CF
9768 : 9D 94 AD 99 94 CD 9D 94  45
9770 : 90 0A D0 B0 AD 98 94 CD  DD
9778 : 9C 94 B0 A8 60 A9 00 8D  0E
9780 : 9E 94 85 5B AD 94 94 2C  5F
9788 : 9A 94 30 06 18 6D 9E 94  CA
9790 : 90 04 38 ED 9E 94 48 AD  F9
9798 : 98 94 85 59 AD 99 94 85  0C
97A0 : 5A 20 10 94 AD 9B 94 85  B6
97A8 : 59 A9 00 85 5A 20 34 94  27
97B0 : 18 A5 5C 6D 92 94 85 14  6C
97B8 : A5 5D 6D 93 94 85 15 68  74
97C0 : AA 20 9B 95 EE 9E 94 AD  A5
97C8 : 9E 94 CD 9B 94 F0 B3 90  58
97D0 : B1 60 A9 00 8D A8 94 20  CC
97D8 : FD AE 20 EB B7 8E A1 94  51
97E0 : A5 14 8D 9F 94 A5 15 8D  CC
97E8 : A0 94 20 FD AE 20 EB B7  A5
97F0 : 8E A4 94 A6 14 8E A2 94  34
97F8 : A5 15 8D A3 94 AD 9F 94  5E
9800 : 8D 92 94 AD A0 94 8D 93  BD
9808 : 94 AD A1 94 8D 94 94 8D  59
9810 : 97 94 AD A2 94 8D 95 94  E6
9818 : AD A3 94 8D 96 94 20 47  8B
9820 : 96 AD A2 94 8D 92 94 8D  A3
9828 : 95 94 AD A3 94 8D 93 94  14
9830 : 8D 96 94 AD A1 94 8D 94  01
9838 : 94 AD A4 94 8D 97 94 20  87
9840 : 47 96 AD 9F 94 8D 92 94  5B
9848 : AD A0 94 8D 93 94 AD A4  FA
9850 : 94 8D 94 94 8D 97 94 AD  A6
9858 : A2 94 8D 95 94 AD A3 94  C9
9860 : 8D 96 94 20 47 96 AD 9F  81
9868 : 94 8D 92 94 8D 95 94 AD  2D
9870 : A0 94 8D 93 94 8D 96 94  69
9878 : AD A4 94 8D 94 94 AD A1  36
9880 : 94 8D 97 94 20 47 96 60  AB
9888 : A9 00 8D A8 94 20 FD AE  49
9890 : 20 EB B7 8E A1 94 A5 14  E3
9898 : 8D 9F 94 A5 15 8D A0 94  38
98A0 : 20 FD AE 20 EB B7 8E A4  6E
98A8 : 94 A6 14 8E A2 94 A5 15  F6
98B0 : 8D A3 94 AD A4 94 CD A1  53
98B8 : 94 B0 09 AE A1 94 8E A4  FF
98C0 : 94 8D A1 94 AE A1 94 8E  7D

98C8 : 9E 94 AD 9F 94 8D 92 94  39
98D0 : AD A0 94 8D 93 94 AD 9E  76
98D8 : 94 8D 94 94 8D 97 94 AD  2E
98E0 : A2 94 8D 95 94 AD A3 94  51
98E8 : 8D 96 94 20 47 96 EE 9E  0C
98F0 : 94 AC A4 94 CC 9E 94 B0  0B
98F8 : D1 60 A9 00 8D A8 94 20  14
9900 : FD AE 20 EB B7 8E A7 94  92
9908 : A5 14 8D A5 94 A5 15 8D  B5
9910 : A6 94 20 FD AE 20 8A AD  3A
9918 : A0 94 A2 79 20 D4 BB 20  B2
9920 : FD AE 20 8A AD A0 94 A2  45
9928 : 7E 20 D4 BB A9 79 A0 94  75
9930 : 20 50 B8 20 2B BC 30 0A  18
9938 : A9 79 A0 94 20 A2 BB 4C  F7
9940 : 49 99 A9 7E A0 94 20 A2  04
9948 : BB A9 BC A0 B9 20 0F BB  6B
9950 : A2 88 A0 94 20 D4 BB A9  DC
9958 : 20 8D 84 94 A9 00 8D 83  CA
9960 : 94 8D 85 94 8D 86 94 8D  29
9968 : 87 94 20 FD AE 20 8A AD  73
9970 : A0 94 A2 8D 20 D4 BB A9  9F
9978 : 83 A0 94 20 A2 BB 20 6B  D4
9980 : E2 A9 79 A0 94 20 28 BA  0A
9988 : A9 11 A0 BF 20 67 B8 20  3A
9990 : 9B BC 18 A5 65 6D A5 94  C6
9998 : 85 14 A5 64 6D A6 94 85  87
99A0 : 15 A9 83 A0 94 20 A2 BB  CB
99A8 : 20 64 E2 A9 7E A0 94 20  68
99B0 : 28 BA A9 11 A0 BF 20 67  19
99B8 : B8 20 9B BC 18 A5 65 6D  1E
99C0 : A7 94 AA 20 9B 95 A9 83  74
99C8 : A0 94 20 A2 BB A9 88 A0  7B
99D0 : 94 20 67 B8 A2 83 A0 94  57
99D8 : 20 D4 BB A9 8D A0 94 20  F7
99E0 : 50 B8 20 2B BC 10 90 60  49
99E8 : A9 00 8D A8 94 20 FD AE  A9
99F0 : 20 EB B7 8E 94 94 A5 14  72
99F8 : 8D 92 94 A5 15 8D 93 94  DD
9A00 : 20 FD AE 20 8A AD A0 94  90
9A08 : A2 79 20 D4 BB 20 FD AE  1C
9A10 : 20 8A AD A0 94 A2 7E 20  8D
9A18 : D4 BB 20 FD AE 20 8A AD  03
9A20 : A0 94 A2 8D 20 D4 BB 20  3C
9A28 : 6B E2 A9 79 A0 94 20 28  1D
9A30 : BA A9 11 A0 BF 20 67 B8  23
9A38 : 20 9B BC 18 A5 65 6D 92  B8
9A40 : 94 8D 95 94 A5 64 6D 93  ED
9A48 : 94 8D 96 94 A9 8D A0 94  8E
9A50 : 20 A2 BB 20 64 E2 A9 7E  B5
9A58 : A0 94 20 28 BA A9 11 A0  CE
9A60 : BF 20 67 B8 20 9B BC 18  22
9A68 : A5 65 6D 94 94 8D 97 94  EB
9A70 : 20 47 96 60 AD 0E DD 09  BA
9A78 : 80 8D 0E DD AD 0F DD 29  1B
9A80 : 7F 8D 0F DD A2 03 A9 00  2E
9A88 : 9D 08 DD CA 10 FA 20 9B  8B
```

```
9A90 : B7 A9 00 8D 77 94 F8 E0  8F
9A98 : 00 F0 0F CA 18 69 01 C9  92
9AA0 : 60 90 F4 EE 77 94 A9 00  26
9AA8 : F0 ED 8D 78 94 D8 20 ED  6E
9AB0 : F6 F0 10 AD 0A DD CD 77  8E
9AB8 : 94 90 F3 AD 09 DD CD 78  EE
9AC0 : 94 90 EB 60 20 FD AE 20  90
9AC8 : D4 E1 A2 00 A0 C0 A9 00  EC
9AD0 : 85 FD A9 A0 85 FE A5 01  BB
9AD8 : 48 29 FE 78 85 01 A9 FD  86
9AE0 : 20 D8 FF 68 85 01 58 60  FC
9AE8 : 20 FD AE 20 D4 E1 A9 61  7C
9AF0 : 85 B9 A9 00 4C D5 FF A9  83
9AF8 : FF 8D A8 94 4C 95 95 A9  96
9B00 : FF 8D A8 94 4C 21 96 A9  FE
9B08 : FF 8D A8 94 4C D7 97 A9  C0
9B10 : FF 8D A8 94 4C 8D 98 A9  79
9B18 : FF 8D A8 94 4C FF 98 A9  15
9B20 : FF 8D A8 94 4C ED 99 A9  90
9B28 : 01 A0 3F 20 91 B3 A2 62  53
9B30 : A0 94 20 D4 BB 20 FD AE  CF
9B38 : 20 BA AD A2 6C A0 94 20  BB
9B40 : D4 BB 20 FD AE 20 BA AD  2B
9B48 : 20 0C BC A9 6C A0 94 20  31
9B50 : A2 BB 20 53 B8 A9 62 A0  E6
9B58 : 94 20 0F BB A2 62 A0 94  20
9B60 : 20 D4 BB A0 C7 20 A2 B3  5D
9B68 : A2 67 A0 94 20 D4 BB 20  50
9B70 : FD AE 20 BA AD A2 79 A0  35
9B78 : 94 20 D4 BB 20 FD AE 20  B6
9B80 : BA AD A2 71 A0 94 20 D4  90
9B88 : BB A9 79 A0 94 20 A2 BB  D6
9B90 : A9 71 A0 94 20 50 B8 A9  67
9B98 : 67 A0 94 20 0F BB A2 67  A1
9BA0 : A0 94 20 D4 BB 60 A9 00  92
9BA8 : 8D A8 94 20 FD AE 20 8A  9D
9BB0 : AD A9 6C A0 94 20 5B BC  92
9BB8 : C9 FF F0 47 20 0C BC A9  4F
9BC0 : 6C A0 94 20 A2 BB 20 53  D5
9BC8 : B8 A9 62 A0 94 20 28 BA  62
9BD0 : 20 9B BC A5 65 85 14 A5  C0
9BD8 : 64 85 15 20 FD AE 20 8A  33
9BE0 : AD A9 71 A0 94 20 5B BC  03
9BE8 : C9 01 F0 1D A9 71 A0 94  E3
9BF0 : 20 50 B8 A9 67 A0 94 20  AA
9BF8 : 28 BA 20 9B BC A5 65 AA  DD
9C00 : 4C A0 95 20 FD AE 20 8A  F1
9C08 : AD 60 FF A9 00 8D A8 94  53
9C10 : 20 FD AE 20 8A AD A9 6C  74
9C18 : A0 94 20 5B BC C9 FF D0  32
9C20 : 0B A9 00 8D 92 94 8D 93  DD
9C28 : 94 4C 4D 9C 20 0C BC A9  72

9C30 : 6C A0 94 20 A2 BB 20 53  45
9C38 : B8 A9 62 A0 94 20 28 BA  D2
9C40 : 20 9B BC A5 65 8D 92 94  48
9C48 : A5 64 8D 93 94 20 FD AE  95
9C50 : 20 8A AD A9 71 A0 94 20  05
9C58 : 5B BC C9 01 D0 08 A9 00  98
9C60 : 8D 94 94 4C 7C 9C A9 71  1C
9C68 : A0 94 20 50 B8 A9 67 A0  1C
9C70 : 94 20 28 BA 20 9B BC A5  93
9C78 : 65 8D 94 94 20 FD AE 20  48
9C80 : 8A AD A9 6C A0 94 20 5B  BF
9C88 : BC C9 FF D0 0B A9 00 8D  5C
9C90 : 95 94 8D 96 94 4C B9 9C  71
9C98 : 20 0C BC A9 6C A0 94 20  81
9CA0 : A2 BB 20 53 B8 A9 62 A0  36
9CA8 : 94 20 28 BA 20 9B BC A5  CB
9CB0 : 65 8D 95 94 A5 64 8D 96  B5
9CB8 : 94 20 FD AE 20 8A AD A9  12
9CC0 : 71 A0 94 20 5B BC C9 01  6F
9CC8 : D0 08 A9 00 8D 97 94 4C  87
9CD0 : E8 9C A9 71 A0 94 20 50  6F
9CD8 : B8 A9 67 A0 94 20 28 BA  B3
9CE0 : 20 9B BC A5 65 8D 97 94  FC
9CE8 : 4C 47 96 A9 00 8D A8 94  EB
9CF0 : 20 FD AE 20 8A AD A9 6C  54
9CF8 : A0 94 20 5B BC C9 FF D0  12
9D00 : 0B A9 00 8D 9F 94 8D A0  A8
9D08 : 94 4C 2D 9D 20 0C BC A9  6A
9D10 : 6C A0 94 20 A2 BB 20 53  25
9D18 : B8 A9 62 A0 94 20 28 BA  B2
9D20 : 20 9B BC A5 65 8D 9F 94  5C
9D28 : A5 64 8D A0 94 20 FD AE  16
9D30 : 20 8A AD A9 71 A0 94 20  E5
9D38 : 5B BC C9 01 D0 08 A9 00  78
9D40 : 8D A1 94 4C 5C 9D A9 71  89
9D48 : A0 94 20 50 B8 A9 67 A0  FC
9D50 : 94 20 28 BA 20 9B BC A5  73
9D58 : 65 8D A1 94 20 FD AE 20  6C
9D60 : 8A AD A9 6C A0 94 20 5B  9F
9D68 : BC C9 FF D0 0B A9 00 8D  3C
9D70 : A2 94 8D A3 94 4C 99 9D  81
9D78 : 20 0C BC A9 6C A0 94 20  61
9D80 : A2 BB 20 53 B8 A9 62 A0  16
9D88 : 94 20 28 BA 20 9B BC A5  AB
9D90 : 65 8D A2 94 A5 64 8D A3  F2
9D98 : 94 20 FD AE 20 8A AD A9  F2
9DA0 : 71 A0 94 20 5B BC C9 01  4F
9DA8 : D0 08 A9 00 8D A4 94 4C  D0
9DB0 : C8 9D A9 71 A0 94 20 50  AF
9DB8 : B8 A9 67 A0 94 20 28 BA  93
9DC0 : 20 9B BC A5 65 8D A4 94  10
9DC8 : 4C FD 97 00 00 00 00 00  F9
```

```
1 REM ********************************    <250>
2 REM *                              *    <229>
3 REM *        GRAFIK TEST-DEMO      *    <044>
4 REM *              ZUM             *    <227>
5 REM *   2.TEIL VON H I R E S - 3   *    <068>
6 REM * (LAEUFT NUR ZUSAMMEN MIT DEM *    <247>
7 REM *      1. TEIL VON HIRES-3)    *    <110>
8 REM * HEIMO PONNATH,2102 HH 93,1984 *    <100>
9 REM *                              *    <236>
10 REM********************************    <003>
20 POKE 52,128:POKE 56,128:SYS 37498
   :PRINT CHR$(147)                        <074>
24 REM                                     <167>
25 REM ++++++ DER RAHMEN +++++++++++++++   <154>
26 REM                                     <169>
30 SYS 34647:POKE 646,14:SP=4:Z=5:GOSUB 1000
                          <140>
35 PRINT"DIES IST DER RAHMEN MIT KOPFZEILE":Z=6
   :GOSUB 1000                             <066>
40 PRINT"----------------------------------"
   :PAU,1:Z=8:GOSUB 1000                   <085>
45 PRINT"TRS UND FUNKT FUER ZOOM-EFFEKT":PAU,2
                          <073>
47 REM                                     <190>
48 REM +++ TRS,FUNKT,TLN,LFUNK,TBK,LTL+++  <089>
49 REM                                     <192>
50 DEF FN A(X)=X↑X                         <103>
55 HFL,6,14:TRS,-1,10,-1,30:FUNKT,A,0,3    <077>
60 TLN,-1,0,5,0:TLN,0,-1,0,30:TRE,-.5,2,1.5,-.5
                          <121>
65 TRS,-3,1.8,-2.5,2.5:TBK,-.5,2,1.5,-.5   <225>
70 LFUNK,A,0,1.5:LTL,-.5,0,1.5,0:LTL,0,-.5,0,2
   :REC,0,0,319,199                        <001>
80 PAU,3:HOF:SP=7:Z=10:GOSUB 1000
   :PRINT"RECHTS AUSSCHNITTVERGROESSERUNG" <200>
85 Z=11:GOSUB 1000:PRINT"MIT MINIMUM BEI .3679"
   :PAU,2:HAN:FOR I=1 TO 5                 <142>
90 TLN,.3679,0,.3679,FN A(.3679):PAU,1
   :LTL,.3679,0,.3679,FN A(.3679):NEXT I   <072>
95 PAU,3:HOF:SP=5:Z=18:GOSUB 1000
   :PRINT"WUENSCHEN SIE EINE HARDCOPY(J/N)?"
                          <048>
100 GOSUB 2000:IF A$="J"THEN GOSUB 3000    <020>
108 REM                                    <251>
109 REM +++++ TPK,TRE ++++++++++++++++++   <223>
110 DEF FN X(T)=3*SIN(4*T):DEF FN Y(T)=3*SIN(9*
   T):PRINT CHR$(147):SP=5:Z=10            <186>
115 SYS 34647:GOSUB 1000:PRINT"LISSAJOUS-FIGURE
   N MIT TPK":PAU,2:HFL,1,6                <220>
120 TRS,-5,5,-5,5:FOR T=0 TO 2*πSTEP.01
   :TPK,FN X(T),FN Y(T):NEXT T             <203>
125 FOR I=5 TO 3.5 STEP-.25:TRE,-I,I,I,-I:NEXT I
                          <165>
130 PAU,3:HOF:POKE 646,14:Z=18:GOSUB 1000
   :PRINT"HARDCOPY ERWUENSCHT(J/N)?"       <231>
135 GOSUB 2000:IF A$="J"THEN GOSUB 3000    <055>
138 REM                                    <025>
139 REM +++++ TRA,TKR,LTV,LTK ++++++++++   <220>
140 PRINT CHR$(147):SYS 34647:POKE 646,14:SP=5
   :Z=10:GOSUB 1000                        <174>
```

```
145 PRINT"POLARE KURVEN MITTELS TRA":Z=12
   :GOSUB 1000:PRINT"HIER:VIERBLAETTRIGER KLEE"
                                        <075>
150 PAU,2:HFL,1,2:DEF FN B(X)=-A*SIN(2*X)/2
                                        <120>
155 A=5:TRS,-4,4,-4,4:FOR X=0 TO 2*πSTEP.1
   :TRA,0,0,FN B(X),FN B(X),X:NEXT X    <051>
160 LTK,0,0,.25,.25,2*π:TKR,0,0,4,4,2*π
   :FOR I=0 TO 2*πSTEP.1:LTV,0,0,.2,.2,I:NEXT I
                                        <099>
165 REC,0,0,319,199:PAU,3:HOF           <101>
170 Z=20:GOSUB 1000:PRINT"DIESMAL EINE HARDCOPY
   (J/N)?":GOSUB 2000                   <017>
175 IF A$="J"THEN GOSUB 3000            <214>
178 REM                                 <065>
179 REM +++++++ LTP,LTR,LTB ++++++++++++ <143>
180 PRINT CHR$(147):SYS 34647:POKE 646,14:SP=5
   :Z=8:GOSUB 1000                      <173>
185 PRINT"GEWOEHNLICHE ZYKLOIDE MITTELS LTP"
   :PAU,2                               <003>
190 DEF FN C(T)=3*(T-SIN(T))
   :DEF FN D(T)=3*(1-COS(T)):TRS,-3,40,-3,50
                                        <013>
195 HFL,14,6:TBK,-1,10,38,-2:LTR,0,9,37,0 <191>
200 FOR T=0 TO 2*π+3 STEP.1:LTP,FN C(T),FN D(T)
   :NEXT T:LTB,30,8,36,.5               <197>
300 PAU,3:HOF:Z=10:GOSUB 1000
   :PRINT"ABSCHLIESSEND NOCH:":Z=12:GOSUB 1000
                                        <026>
305 PRINT"3D-DARSTELLUNG MITTELS":Z=13
   :GOSUB 1000:PRINT"TRS UND FUNKT":PAU,3 <074>
310 DEF FN E(X)=EXP(X/Z)*SIN(X):HAN
   :REC,0,0,319,199                     <008>
315 TRS,-3,5,-5,9:TLN,-2,0,4,0:TLN,0,-1,0,8
   :TLN,-1,-1,2,2:D=1/20                <010>
320 FOR I=0 TO 20:Z=5-4*I*D:TRS,-3-I*D,5-I*D,
   -5-I*D,9-I*D:FUNKT,E,-.5,3.31:NEXT I <052>
325 PAU,4:HOF:SP=3:Z=18:GOSUB 1000
   :PRINT"SOLLS JETZT EINE HARDCOPY SEIN(J/N)?"
                                        <210>
330 GOSUB 2000:IF A$="J"THEN GOSUB 3000 <251>
398 REM                                 <030>
399 REM +++++++ ENDE ++++++++++++++++++ <153>
400 PRINT CHR$(147):SYS 34647:POKE 646,14:SP=10
   :Z=12:GOSUB 1000:PRINT"DAS WARS !"   <046>
405 PAU,5:PRINT CHR$(147):AUS:END       <065>
996 REM                                 <118>
997 REM ++++++ UNTERPROGRAMME ++++++++++ <095>
998 REM                                 <120>
999 REM ++++++ CURSOR SETZEN ++++++++++ <011>
1000 POKE 211,SP:POKE 214,Z:SYS 58640:RETURN
                                        <122>
1998 REM                                <100>
1999 REM +++++++ GET-ABFRAGE ++++++++++ <096>
2000 GET A$:IF A$=""THEN 2000           <102>
2100 RETURN                             <202>
2998 REM                                <080>
2999 REM ++++++ HARDCOPY ++++++++++++++ <050>
3000 OPEN 1,4,10:PRINT#1:CLOSE 1:HAN:SYS 34865
   :HOF:RETURN                          <096>
```

```
1 REM ********************************    <250>
2 REM *                              *    <229>
3 REM *       GRAFIK - DEMO          *    <236>
4 REM *            ZUM               *    <227>
5 REM *  TESTEN DES ERSTEN TEILS VON  *    <220>
6 REM *                              *    <233>
7 REM *        H I R E S - 3         *    <197>
8 REM *                              *    <235>
9 REM *     (H.PONNATH HH 1984)      *    <045>
10 REM********************************    <003>
20 POKE 52,128:POKE 56,128:SYS 37498
   :REM EINSCHALTEN DER NEUEN BASIC-BEFEHLE
                                        <138>
25 REM ---------- DER LIN-BEFEHL -------    <054>
30 DEF FN A(X)=(Y2-Y1)*(X-X1)/(X2-X1)+Y1
   :DEF FN B(Z)=(Y4-Y3)*(Z-X3)/(X4-X3)+Y3  <043>
35 DEF FN C(I)=X1+I*(X2-X1)/10
   :DEF FN D(I)=X3+I*(X4-X3)/10             <130>
40 DEF FN E(I)=X4-I*(X4-X3)/10              <065>
45 X1=10:X2=100:X3=190:X4=310:Y1=10:Y2=170
   :Y3=195:Y4=15                            <201>
50 HFL,1,6                                  <203>
55 FOR I=0 TO 10 STEP.5:X=FN C(I):Z=FN D(I)
   :LIN,X,FN A(X),Z,FN B(Z):NEXT            <223>
60 LIN,X1,Y1,X2,Y2:LIN,X3,Y3,X4,Y4         <244>
65 PAU,5                                    <136>
70 HOF:PRINT CHR$(147)"DAS WAR DER LIN-BEFEHL"
   :PRINT:PRINT"PAU,HFL U. HOF FUNKTIONIEREN
    AUCH                                    <087>
75 PAU,5                                    <146>
80 REM ------------ DER REC-BEFEHL ------    <100>
85 LOE:HAN                                  <070>
90 DEF FN Z(X)=-3.39098E-3*X↑2+1.053668*X+8.9497
   :X1=200:Y1=198:X2=310:Y2=5               <117>
95 X3=2:X4=309:E=60                         <121>
100 FOR I=0 TO E:XU=X2-I*(X2-X1)/E:YU=FN A(XU)
   :XO=X4-I*(X4-X3)/E:YO=FN Z(XO)           <138>
105 REC,XO,YO,XU,YU:NEXT I:PAU,3            <033>
110 FAR,6,1:PAU,2:FAR,0,7:PAU,1:FAR,2,8:PAU,1
   :LOE:PAU,1:HOF:PRINT                     <166>
115 PRINT"DAS WAREN DIE BEFEHLE REC,FAR UND
    LOE":PAU,5                              <137>
120 REM -- PKT,BLO,CIR,LBK UND LRE -----    <019>
125 HAN:BLO,40,100,160,150:BLO,180,100,200,150
   :BLO,200,120,250,150                     <111>
130 BLO,240,100,250,120:LBK,44,102,68,120
   :LBK,72,102,98,120:LBK,102,102,128,120   <249>
135 LBK,132,102,156,120:LBK,182,102,198,148
   :LRE,44,122,68,148                       <109>
140 LRE,132,122,156,148:LIN,160,145,180,145
   :CIR,45,155,5,5,2*π                      <243>
145 CIR,55,155,5,5,2*π:CIR,155,155,5,5,2*π
   :CIR,145,155,5,5,2*π                     <120>
150 CIR,185,155,5,5,2*π:CIR,195,155,5,5,2*π
   :CIR,245,155,5,5,2*π                     <182>
155 LIN,0,160,319,160:LIN,0,163,319,163     <247>
160 FOR K=1 TO 10:FOR J=1 TO 235            <122>
165 V=INT(RND(1)*20)+235-J:W=INT(RND(1)*100-J/3)
   :IF W>0 AND V>0 THEN:PKT,V,W:NEXT        <067>
170 NEXT K:PAU,5:HOF                        <015>
175 PRINT:PRINT"DIESER ZUG FUHR MIT
   :PKT,BLO,CIR,LBK,LRE":PAU,5              <001>
180 REM ------ RAD UND LRA -------------    <055>
185 HFL,0,6:FOR L=0 TO 3*πSTEPπ/30
   :RAD,160,100,INT(10*L),INT(10*L),L       <147>
190 NEXT L:PAU,4:FOR L=0 TO 3*πSTEPπ/15
   :LRA,160,100,INT(7*L),INT(7*L),L:NEXT L  <013>
195 PAU,4:HOF:PRINT:PRINT"SOWOHL RAD ALS AUCH
    LRA FUNKTIONIEREN":PAU,4                <254>
200 REM ------ LPK,LLN,LKR -------------    <188>
205 POKE 53280,0                            <194>
210 HFL,6,14:BLO,20,10,300,190:X(1)=60:X(2)=120
   :X(3)=170:X(4)=200:X(5)=201              <244>
215 X(6)=240:X(7)=260:Y(1)=65:Y(2)=42:Y(3)=70
   :Y(4)=100:Y(5)=135:Y(6)=150              <019>
220 Y(7)=117:FOR I=1 TO 7:LKR,X(I),Y(I),2,2,2*π
   :LKR,X(I),Y(I),1,1,2*π:NEXT I            <023>
225 FOR I=2 TO 7:LLN,X(I-1),Y(I-1),X(I),Y(I)
   :NEXT I:LLN,X(7),Y(7),X(4),Y(4)          <013>
230 FOR I=0 TO 500:V=INT(RND(1)*280)+20
   :W=INT(RND(1)*180)+10:LPK,V,W:NEXT I:PAU,3
                                        <034>
235 POKE 53280,14:HOF:PRINT:PRINT"DER GROSSE
    WAGEN WURDE GEBILDET MIT"               <231>
240 PRINT"DEN BEFEHLEN:LLN,LPK UND LKR":PRINT
   :PRINT:POKE 646,1:PRINT"ALLES O.K!"      <197>
245 POKE 646,14:END                         <119>
```

TEIL II

70 Farben für den Commodore 64

Einleitung

Wahrscheinlich werden viele, die den Titel lesen, behaupten, daß 70 Farben auf dem Commodore 64 nicht darstellbar sind. Liest man das Handbuch des C 64, erfährt man, daß der Video-Chip nur 16 Farben auf den Schirm bringen kann.

Heuer habe ich ein Atari-Spiel gesehen, das den Eindruck erweckte, dreidimensional zu sein, weil es unterschiedliche Farbabstufungen verwendete. In diesem Augenblick entschied ich mich, den gleichen Effekt auf einen C 64 zu bringen. Die Frage war nur, wie!

Wie kann ein Drucker oder eine Farbröhre fast alle Farben des Regenbogens darstellen? Sie wissen sicher schon, daß das mit nur drei Grundfarben geschieht. Das menschliche Auge kann drei verschiedene Farben, die ganz nah beieinander sind, nicht auflösen. Es sieht einen Farbfleck, dessen Farbe anders ist als die drei Einzelfarben. Die drei Grundfarben des Fernsehgeräts sind rot, grün und blau (RGB-System). Stellen Sie sich ganz nah vor ein Fernsehgerät, und Sie sehen, daß die einzelnen Farbflecke aus drei verschiedenen Farbpunkten zusammengesetzt sind.

Diese drei Farben mischt der Video-Chip des C 64 und erzeugt damit die 16 Farben. Was passiert nun, wenn man versucht, diese 16 Farben noch einmal untereinander zu vermischen?

Im nächsten Abschnitt zeige ich Ihnen, wie das geht.

1

Das Mischen von Farben

1 Das Mischen von Farben

1.1 Grundlagen

Sie wissen sicher, daß Fernsehröhren 525 oder 625 Zeilen am Bildschirm erzeugen, mit denen sie ein Bild aufbauen. Alle 25- oder 30mal pro Sekunde wird eine Bildschirmhälfte geschrieben. Zuerst werden die ungeraden Zeilen in der ersten Hälfte und danach die geraden in der zweiten Hälfte gebildet. Beim C 64 sind beide Bildschirmhälften identisch.

Die mögliche Bildschirmauflösung des C 64 besteht aus 320 mal 200 Punkten. Überlegen wir uns, wie man in dieser sehr hoch auflösenden Grafik Farben mischen kann.

Zweifellos kann ein Punkt nicht gleichzeitig aus zwei Farben bestehen. Aber was passiert, wenn die Farben des Punktes sehr schnell geändert werden? Ich habe mich hingesetzt und experimentiert. Alle Experimente führten zur gleichen Schlußfolgerung. Fernsehgeräte erzeugen C 64-Bilder mit einer Frequenz von 50 bis 60 pro Sekunde, so daß es keinen Zweck hat, die Farben mit einer höheren Geschwindigkeit zu ändern. Dieser Vorgang verbraucht bereits 90% der Prozessor-Zeit. Trotz dieser Einschränkung bestand die einzige Möglichkeit, das Ziel zu erreichen, darin, die Farbpunkte zu verändern.

1.2 Senkrechte Linien

In einigen Fällen geben Farbkombinationen am Bildschirm keinen deutlichen Kontrast (zum Beispiel brauner Text - Farbcode 9 - auf einem blauen Schirm - Farbcode 6). Die meisten C 64-Programmierer verwünschen diesen Effekt. Aber was passiert, wenn wir diese oder andere Farbpaare mit diesem schlechten Kontrast einsetzen? Wir bauen ein Bild auf, bei dem unterschiedliche Farbpunkte sehr schnell nebeneinander geändert werden. Am Bildschirm sind vertikale Linien mit jeweils einer Punktreihe zu sehen.

Das erste Programm (FARB01 1 im Anhang) ist so geschrieben. Als nächstes gebe ich eine kurze Beschreibung dieses Programms.

In Zeile 10 wird der hochauflösende Bildschirmbereich mit dem Bit-Muster: 01010101bin = 85dez gefüllt. Der hochauflösende Bildschirmbereich beginnt bei Speicherstelle 8192dez = 2000hex und endet bei Speicherstelle 16191dez = 3F40hex. Mit

der Funktion in Zeile 20 wird der Video-Chip in den hochauflösenden Grafik-Modus gesetzt.

In Zeile 30 wird der Startwert der Variablen A (in diesem Fall 0) gewählt. Diese Variable enthält die momentane Farbkombination.

Zeile 40 ruft ein ganz einfaches Unterprogramm auf, das ab Zeile 100 beginnt und den Bildschirmbereich (das ist der Farbspeicher im hochauflösenden Grafikmodus) mit dem momentanen Wert von A auffüllt. Sie wissen sicher, daß in diesem Modus die Farbe eines 8 x 8 Punktes (Größe eines Zeichens) im Bildschirmspeicher und ebenfalls im Farbspeicher (ab Speicherstelle 55296dez = D800hex) abgelegt sind. Das ist deshalb so, weil eine Speicherstelle im Farbspeicher nur vier Bits enthalten kann. Bei der hochauflösenden Grafik werden acht Bits verlangt, damit zwei Farben beschrieben werden können, nämlich die Vordergrund- und Hintergrundfarben jedes 8 x 8 Punktes. Die Variable A nimmt alle möglichen Farbkombinationen ein, wenn sie von 0 bis 255 inkrementiert wird. Das geschieht in Zeile 60. Mit Zeile 50 erhalten Sie die Möglichkeit, durch Drücken der Leertaste die Farbkombinationen "weiterzuschalten".

Nach dem Starten des Programms wird der Bildschirm mit verschiedenen Farbkombinationen gefüllt.

Geben Sie RUN ein und sehen Sie sich an, was passiert. Nach einiger Zeit (die FOR/NEXT-Schleife in Zeile 10 ist sehr lang) wird der Bildschirm schwarz. Das ist die erste Kombination: schwarz und schwarz gemischt.

Steht eine Farbe als Bildschirmfarbe fest, kann sie mit 16 weiteren Farben kombiniert werden. Dabei sind einmal beide gemischten Farben gleich. Bei allen anderen Kombinationen erhält man verschiedene vertikale Farblinien. Warum erhalten wir aber immer das gleiche Ergebnis anstelle neuer Farben? Wahrscheinlich liegt das an der hohen Arbeitsfrequenz des Video-Chips und des Fernsehgeräts.

An diesem Punkt wurde ich ganz mutlos. Es sah so aus, als ob es keine Möglichkeit gäbe, mehr Farben zu erzeugen. Sie werden sich fragen, warum ich nicht auf die Idee gekommen bin, horizontale Linien zu verwenden. Ich glaubte nicht, daß auf diese Weise der gewünschte Effekt eintreten könnte. So stellte ich meine Experimente vorerst ein. Als ich Monate später ein C 64-Spiel entwickelte, war der Wunsch wieder da, verschiedene Farbschattierungen zu verwenden. Nachdem ich den alten Versuch noch einmal durchgespielt hatte, entschied ich mich, den Weg mit den horizontalen Linien zu beschreiten.

1.3 Horizontale Linien

Betrachten wir das Programm 2 (FARB02) im Anhang. Dieses Programm gleicht dem ersten Programm mit Ausnahme der Zeile 10. Das Programm schreibt in den vorher beschriebenen hochauflösenden Bildschirmbereich abwechselnd 0 und 255 hinein. Die binären Werte lauten: 11111111bin und 00000000bin. Diese Bit-Muster erscheinen als waagrechte Linien am Bildschirm. Lassen wir nun dieses Programm laufen.

Zunächst erhielt ich das gleiche Ergebnis wie beim ersten Programm. Nach einigen neuen Kombinationen, die durch das Drücken der Leertaste erschienen, sah ich die verschiedensten Farbzusammenstellungen. Nach diesen Kombinationen habe ich gesucht. Sie können sich vorstellen, daß ich sehr glücklich war. Nach vielen Versuchen hatte ich endlich mein Ziel erreicht. Jetzt mußte ich nur noch die richtigen Kombinationen finden, um neue Farben zu erzeugen.

1.4 Tabelle der ausgewählten Kombinationen

Geben Sie nach dem letzten geladenen Programm nicht NEW ein. Dieses Programm hilft Ihnen bei der Auswahl der verschiedenen Farbkombinationen. Lassen Sie das Programm wieder laufen und betrachten Sie folgende Kombinationen:

1.	Schwarz	mit	Schwarz
2.	Weiß	mit	Weiß
3.	Rot	mit	Rot
4.			Blau
5.			Braun
6.			Grau1
7.	Türkis	mit	Türkis
8.			Gelb
9.			Hellgrün
10.			Grau3

11. Violett	mit	Violett
12.		Grün
13.		Orange
14.		Hellrot
15.		Grau2
16.		Hellblau
17. Grün	mit	Violett
18.		Grün
19.		Orange
20.		Hellrot
21.		Grau2
22.		Hellblau
23. Blau	mit	Rot
24.		Blau
25.		Braun
26.		Grau1
27. Gelb	mit	Türkis
28.		Gelb
29.		Hellgrün
30.		Grau3
31. Orange	mit	Violett
32.		Grün
33.		Orange
34.		Hellrot
35.		Grau2
36.		Hellblau
37. Braun	mit	Rot
38.		Blau
39.		Braun
40.		Grau1
41. Hellrot	mit	Violett
42.		Grün
43.		Orange
44.		Hellrot
45.		Grau2
46.		Hellblau

47. Grau1	mit	Rot
48.		Blau
49.		Braun
50.		Grau1
51. Grau2	mit	Violett
52.		Grün
53.		Orange
54.		Hellrot
55.		Grau2
56.		Hellblau
57. Hellgrün	mit	Türkis
58.		Gelb
59.		Hellgrün
60.		Grau3
61. Hellblau	mit	Violett
62.		Grün
63.		Orange
64.		Hellrot
65.		Grau2
66.		Hellblau
67. Grau3	mit	Türkis
68.		Gelb
69.		Hellgrün
70.		Grau3

Insgesamt ergibt das 70 Kombinationen.

Ich habe diese lange Tabelle nicht deshalb erstellt, um meinem Verleger möglichst viele Seiten zu verkaufen. Ich möchte vielmehr, daß Sie nachvollziehen können, was ich feststellte, als ich diese Tabelle zu Papier brachte. Sie sollen selbst erkennen, daß es fünf Farbgruppen gibt und daß nur Farben innerhalb der gleichen Gruppe gemischt werden können. Folgende Gruppen gibt es:

1. Schwarz
2. Rot, Blau, Braun, Grau1
3. Violett, Grün, Orange, Hellrot, Grau2, Hellblau
4. Türkis, Gelb, Hellgrün, Grau3
5. Weiß

Betrachten Sie Programm 3 (FARBBAS) im Anhang. Dieses sehr einfache Programm zeigt die Farben der 5 oben genannten Gruppen. Können Sie Ihr Fernsehgerät in Schwarzweiß-Darstellung einstellen, so tun Sie das jetzt oder nehmen Sie die Farbsteuerung so zurück, daß Sie einen Schwarzweiß-Bildschirm erhalten. Sie sehen dann, daß die Leuchtstärke aller Farben der gleichen Gruppen gleich ist. Das hat folgende Bedeutung: Einmal kann der C 64-Video-Chip nur fünf unterschiedliche Helligkeitsstufen erzeugen. Zum anderen können nur die Farben mit gleicher Leuchtstärke untereinander kombiniert werden.

Wieviele gibt es nun davon? Gruppe 1 hat nur eine Kombination: Schwarz und Schwarz. Das gleiche gilt auch für Gruppe 5. Gruppe 2 (genauso wie Gruppe 4) enthält vier Farben mit insgesamt 4 x 4 = 16 Kombinationen. Schließlich können die 6 Farben von Gruppe 3 mit 6 x 6 = 36 unterschiedlichen Arten vermischt werden. Insgesamt ergibt das 1 + 16 + 36 + 16 + 1 = 70 Kombinationen.

Unterscheiden sich diese Kombinationen wirklich untereinander? Das kann man nur herausfinden, wenn man mit diesen 70 Farben ein Bild erstellt. Das geschieht in Programm 4 (FARBDEM1 im Anhang).

In den Zeilen 10 bis 40 werden der gewünschte hochauflösende Grafikmodus, der Bildschirm- und der Farbspeicher, Vorder- und Hintergrundfarbe angewählt. Mit den Zeilen 50 bis 80 wird der hochauflösende Bildschirmbereich mit waagrechten Zeilen gefüllt. In den Zeilen 90 bis 110 wird der Farbspeicher mit Grau2 als Vordergrund- und Hintergrundfarbe besetzt.

Ich habe schon erklärt, daß ein Byte im Farbspeicher für zwei Farben steht: die Vorder- und Hintergrundfarbe eines 8 x 8 Punktfelds. Deshalb bedeutet 158dez = 10011110bin an Speicherstelle 1024, daß die obere linke Ecke des Bildschirms mit der Farbe braun (Farbcode 9) als Vordergrund und hellblau (Farbcode 14) als Hintergrundfarbe gefüllt wird. Hexadezimal kann 158dez mit 9Ehex angegeben werden. Sie sehen, daß die zwei Ziffern der Hexadezimal-Darstellung einer Zahl angeben, welche Farbe Vordergrund (9hex = 9dez) und Hintergrund (Ehex= 14dez) haben sollen.

Deshalb sind die Data-Anweisungen, die in Zeile 10000 beginnen, in Hexadezimal-Darstellung angegeben. Insgesamt gibt es 70 Zahlen, welche die 70 möglichen Farbkombinationen darstellen.

In Zeile 120 baue ich einen Bildschirm mit sieben Zeilen auf, wobei jede Zeile zehn Felder erhält. Alle Felder haben unterschiedliche Farben. Das Unterprogramm ab 230 (von Zeile 140 aus aufgerufen) liest die hexadezimale Darstellung einer Farbe ein und wandelt sie in einen Dezimalwert um (Variable C). Diese Variable wird in Zeile 170 mit POKE in die angegebenen Farbspeicherstelle eingegeben.

Zeile 220 sorgt dafür, daß das Programm nicht mit STOP oder END abgebrochen wird, da eine READY-Meldung den Farbspeicherinhalt zerstört.

Nachdem das Programm mit RUN gestartet wurde, erzeugt es ein Bild mit 70 verschiedenen Farben. Sie sehen selbst, daß alle Farben unterschiedlich sind. Wenn Sie es nicht glauben, ändern Sie die Farbcodes in den Data-Anweisungen am Ende des Programms. Im Anhang finden Sie drei weitere Möglichkeiten. Die zweite davon in sehr interessant. Es stehen jeweils zwei Kombinationsfarben nebeneinander, also 26 oder 62. Was ist der Unterschied zwischen dem Mischen von Rot und Blau oder dem Mischen von Blau und Rot? Ich weiß es nicht. Aber alle Mischungen erzeugen unterschiedliche Farben.

Es gibt noch eine weitere Frage, die ich nicht beantworten kann. Warum ergeben der Einsatz von waagrechten Linien und nicht der von senkrechten am Bildschirm den gewünschten Effekt? Das ist eine Besonderheit des C 64-Video-Chips und nicht eine des Farbmonitors oder des Fernsehgeräts. Ich habe dieses Programm mit allen möglichen Fernsehgeräten mit verschiedenen Farbsystemen (PAL und auch NTSC) laufen lassen.

Wahrscheinlich liegt es doch am Video-Chip. Das kann für einen Hardware-Ingenieur eine harte Prüfung sein, auch wenn er die Schaltung des Chips kennt.

Dieses Programm dient nicht nur dazu, die 70 Farben zu zeigen. Man kann damit auch die Farbkombinationen für andere Programme auswählen.

Man muß wissen, daß die Farbmischung, die auf diese Art erzeugt wird, die vertikale Auflösung vermindert. Die Auflösung in verschiedenen Modi sieht folgendermaßen aus:

Normaler Zeichenmodus:
Erweiterter Zeichenmodus:
Hochauflösender Grafik-Modus: 320 x 100 Punkte

Vielfarben-Zeichenmodus:
Vielfarben-Grafik-Modus: 160 x 100 Punkte

2

Die Verwendung von Mischfarben in der Praxis

2 Die Verwendung von Mischfarben in der Praxis

2.1 Mischfarben im HIRES-Modus

Wie man Mischfarben im hochauflösenden Grafikmodus einsetzt, haben Sie in einigen Beispielen gesehen. Aber Sie wissen noch nicht, wie Sie das in Ihre eigenen Programme einbinden können. Deshalb werden wir jetzt ein paar praktische Übungen machen.

Immer dann, wenn Sie den hochauflösenden Grafik-Modus verwenden, müssen Sie zuerst den Video-Chip in diesen Modus setzen und zwei Speichergebiete auswählen. Einen für den 8 KByte großen Grafik-Bildschirm, den anderen für den 1 KByte Farbspeicher. Dafür kann man zwei Speichergebiete verwenden.

Wie Sie sicherlich wissen, kann der Video-Chip nur mit 16 KByte RAM auf einmal arbeiten. Zuerst müssen Sie den Teil des Speichers, den Sie für die Grafik verwenden wollen, auswählen. Das können Sie mit dem CIA 2 (dem zweiten "zentralen Interface-Adapter"-Chip des C 64) machen. Der Video-Chip selbst erzeugt ein 14 Bit-Adreß-Signal. Das 15. und 14. Bit, das dieser Adresse hinzugefügt wird, sind der Kehrwert der unteren zwei Bits der Speicherstellen DD00hex = 56576dez (Port A des CIA 2). Von nun an verwende ich immer die unteren 16 KByte. Dieser Teil wird automatisch angewählt, wenn der Computer eingeschaltet ist.

Danach müssen Sie sich entscheiden, welche Hälfte dieser 16 KByte den Grafikspeicher und welches KByte als Farbspeicher arbeiten sollen. Das machen Sie, indem Sie einen Wert an die Speicherstelle D018hex = 53272dez mit POKE eingeben. Die oberen vier Bits dieses Werts wählen den Farbspeicher (0 bis 15), die unteren drei Bits haben keine Bedeutung, das restliche Bit (Bit 3) wählt den Grafikspeicher. In meinem Programm stellen Sie fest, daß ich immer 31dez (= 1Fhex) mit POKE in dieser Speicherstelle eingebe. Die oberen vier Bits dieses Werts sind: 0001bin = 1dez. Das bedeutet, daß der Farbspeicher bei 1024dez beginnt und mit 2023dez endet. Die unteren vier Bits wählen den Grafikspeicher bei 8192dez bis 16191dez.

Danach müssen Sie den Video-Chip in den hochauflösenden Grafikmodus setzen, indem Sie in die Speicherstelle D011hex = 53265dez den Wert 59 mit POKE hineingeben. Erfolgt ein Fehler, wenn Sie mit RUN Ihr Programm laufen lassen, können Sie den Video-Chip in den normalen Zustand zurücksetzen, wenn Sie die Tasten ‹RUN/STOP› und ‹RESTORE› gleichzeitig drücken. Möchten Sie das auf diese Weise nicht machen (weil Sie vielleicht den nicht maskierbaren Interrupt für andere Zwecke verwenden wollen), können Sie folgende Anweisung in Ihr Programm einbauen:

```
POKE 53265,27: POKE 53272,21
```

Diese Anweisung kann unmittelbar im direkten Modus eingegeben werden, wenn das Programm durch einen Fehler angehalten hat. So können Sie auch die Fehlermeldung sehen.

Weitere Informationen über den hochauflösenden Grafikmodus finden Sie im Commodore 64-Handbuch oder im Commodore 64-Programmierführer.

Programm 5 (FARBSET) im Anhang ist eine verbesserte Version des Programms 4. Innerhalb dieses Programms erscheinen zwei "Cursor", die aus Bytes erzeugt worden sind, am Bildschirm. Mit ihnen können Sie die Farbfelder anwählen und ändern.

Zuerst müssen Sie einen aktiven Cursor wählen, indem Sie F1 für Cursor 1 oder F3 für Cursor 2 drücken. Diese Cursor kann man mit den Cursor-Steuertasten bewegen. Drükken Sie die Taste C, werden die Farbfelder unter dem jeweiligen Cursor ausgetauscht. Der aktive Cursor ist immer weiß, der andere ist schwarz.

Der erste Teil des Programms funktioniert wie Programm 4. Die Zeilen 150 bis 190 im Unterprogramm sind geändert. Sie werden aufgerufen, wenn sich zwei Felder ändern.

Der zweite Teil des Programms, der mit Zeile 1000 beginnt, erzeugt zuerst die beiden Sprites. Die Daten für diese Sprites finden Sie in Zeile 11000 bei den DATA-Anweisungen.

In Zeile 1090 wird die Cursor-Position initialisiert (Zeile und Spalte). Zeile 1100 wählt die momentane Cursor-Farbe. In den Zeilen 1110 bis 1160 wird die Position des Cursors vertauscht und in die Register des Video-Chips mit POKE eingegeben.

In den Zeilen 1170 bis 1380 wird mit GET die gedrückte Taste aufgenommen und der Befehl ausgeführt (Cursor-Anwahl, Cursor-Bewegung und Ändern der Farbe). Nach dem Listing des Programms finden Sie eine Liste und die Bedeutung der wichtigsten Variablen.

Ich hoffe, daß dieses Programm Ihnen hilft, Ihre eigenen Bilder zu erstellen.

Programm 6 (FARBHIMMEL) zeigt eine mögliche Anwendung von Zusatzfarben. In diesem Programm wird versucht, die Weite des Himmels nachzuempfinden. Es kann Teil eines Video-Spiels sein, das zusammen mit anderen dreidimensionalen Effekten funktioniert.

In den Zeilen 10 bis 70 wird wieder der hochauflösende Bildschirm erzeugt. Die Zeilen 80 bis 100 füllen die erste Zeile des Farbspeichers mit dem Code für Dunkelblau. Diese Farbe wurde mit Rot (Farbcode 2) und Blau (Farbcode 6) erzeugt.

Die Zeilen 110 bis 130 füllen die zweiten und dritten Farbspeicherzeilen mit der nächsten Blaukombination (Code 6 für Blau und Code 11 für Grau1). Die Zeilen 140 bis 160 füllen die nächsten drei Farbspeicherzeilen mit dem Gegencode (Grau1 und Blau).

Wenn Sie den Bildschirm betrachten, stellen Sie fest, daß diese beiden Kombinationen sich von den anderen etwas unterscheiden. Die zweite Kombination ist etwas heller, obwohl die gleichen Farben miteinander gemischt worden sind.

Die nächsten vier Zeilen des Farbspeichers werden mit dem Originalblau des C 64 gefüllt. Die nächsten fünf Zeilen sind dann hellblau. Die restlichen Zeilen des Bildschirms sind grün und stellen den Boden dar.

Mit dem nächsten Programm (Nummer 7) habe ich versucht, einen Spiegeleffekt an einer nächtlichen Wasseroberfläche zu erzeugen. In diesem Programm - wie auch in den anderen - habe ich als Grafik-Effekt nur die zusätzlichen Farben eingesetzt.

Nachdem Sie das Programm mit RUN gestartet haben, sehen Sie zwei unterschiedliche große Mauern vor einer Wasseroberfläche und Ihr Spiegelbild darin. Die Farben des Spiegelbilds sind etwas anders als die des Originals. Dieser Effekt kann auch mit anderen dreidimensionalen Darstellungsformen verwendet werden.

Der erste Teil des Programms ist mit dem des vorhergehenden gleich. Ab Zeile 190 habe ich die Prozedur viermal wiederholt, damit die vier Felder am Bildschirm mit den Farben aufgefüllt werden. Dieser Kombinations-Farbeffekt kann auch im Vielfarben-Grafikmodus eingesetzt werden. Sie wissen sicher schon, daß dieser Modus angewählt wird, indem man mit POKE den Wert 216 in die Speicherstelle D0116hex = 350270dez eingibt.

Wird der Vielfarben-Modus angewählt, kann jede Zeichengröße (8 x 8 Punkte) drei einzelne Farben haben, außerdem noch eine Hintergrundfarbe, die für den ganzen Bildschirm einheitlich ist. Denken Sie daran, daß im Vielfarben-Modus die horizontale Auflösung und beim Zusatzfarben-Modus die vertikale Auflösung vermindert wird. In diesem Fall liegt die Auflösung bei 160 x 100 Punkten. Das ist nicht allzu viel, wenn Sie Wert auf schöne Bilder legen.

2.2 Gemischte Farben im Zeichen-Modus

Der Standard-Zeichen-Modus kann für Mischfarben nicht eingesetzt werden. Jedes Zeichen hat seine eigene individuelle Vordergrundfarbe und die Hintergrundfarbe ist für alle 1000 Zeichen des Bildschirms gleich. Es ist nicht möglich, zwei Farben für ein Einzelzeichen zu mischen.

Das gilt auch für den erweiterten Zeichen-Modus. In diesem Modus können Sie eine der drei vordefinierten Hintergrundfarben für jedes Zeichen anwählen. Deshalb ist es sehr schwierig, Farbmischungen bei einer solchen Begrenzung einzusetzen. Der einzige Zei-

chen-Modus, der jetzt noch übrigbleibt, ist der Vielfarben-Zeichen-Modus. In diesem Fall können nur acht Farben aus den 16 Einzelfarben verwendet werden. Die Farbmischung ist in diesem Fall noch mehr eingeschränkt.

Ich sehe außer einigen besonderen Anwendungen auch keine großen Notwendigkeiten, Mischfarben im Zeichen-Modus einzusetzen.

2.3 Weitere Konsequenzen

Ich habe bis jetzt noch kein Wort darüber verloren, wie man Mischfarben bei Sprites einsetzen kann. Die Sprites-Anwendung hängt so sehr vom Zweck des Programms ab, daß es keine allgemeinen Regeln gibt, außer denen, die ich vorher schon erklärt habe.

Die Anwendung von Mischfarben hat ihre eigene Begrenzung. Wenn Sie mit RUN Programm 4 erneut laufen lassen und Ihr Fernsehgerät in den Schwarzweiß-Modus bringen, sehen Sie, daß alle 70 Farben schließlich nur aus 5 verschiedenen Leuchtstärken bestehen.

Farben mit der gleiche Helligkeitsstufe erzeugen ein Schattenbild, wenn sie nebeneinander stehen (dieser Effekt kann in einigen Fällen ganz nützlich sein - siehe Programm 6). Die Verwendung von zusätzlichen Farben vermindert die vertikale Auflösung. Sie müssen Ihre eigenen Kompromisse zwischen Farben und Auflösung treffen.

Mit all diesen Einschränkungen können Sie sich jetzt selbst entscheiden, ob Sie bei bestimmten Anwendungen die Farbmischung verwenden wollen oder nicht.

Die Software-Gruppen, die trotz der Beschränkungen die zusätzlichen Farben einsetzen, sind die Entwickler von Spiel- und Grafikprodukten. Bei einigen Spielprogrammen wird das Bild deutlich verbessert und man kann dreidimensionale Darstellungen entwerfen, die mehr Spaß machen. Die Farben können für Schatten- und Spiegelbilder eingesetzt werden.

Anhang

Bemerkung:

Alle Programme, die in diesem Anhang aufgezeigt werden, verwenden die Speicherstellen 1024 bis 2023 für den Farbspeicher und die Speicherstellen 8192 bis 16384 für den hochauflösenden Grafikspeicher im HIRES-Modus.

Wird der Video-Chip durch ein Programm in den Grafik-Modus gesetzt, kann durch das gleichzeitige Drücken der Tasten RUN/STOP und RESTORE in die normale Bildschirmdarstellung zurückgekehrt werden.

Eventuelle Fehler im Programm oder in den Tabellen bitte ich zu entschuldigen. Ich hoffe, sie so fehlerfrei wie möglich wiedergegeben zu haben.

Die hier abgedruckten Programme befinden sich auch auf der Demonstrations-Diskette.

Programm 1 (FARB01)

```
10 FOR I=8192 TO 16191:POKE I,85:NEXT
20 POKE 53272,31:POKE 53265,59
30 A=0
40 GOSUB 100
50 GETS$:IF S$<>" " THEN 50
60 A=A+1:IF A=256 THEN A=0
70 GOTO 40
100 FOR I=1024 TO 2023:POKE I,A:NEXT
110 RETURN
```

Programm 2 (FARB02)

```
10 A=0:FOR I=8192 TO 16191:POKE I,A:A=255-A:NEXT
20 POKE 53272,31:POKE 53265,59
30 A=0
40 GOSUB 100
50 GET S$:IF S$<>" " THEN 50
60 A=A+1:IF A=256 THEN A=0
70 GOTO 40
100 FOR I=1024 TO 2023:POKE I,A:NEXT
110 RETURN
```

Programm 3 (FARBBAS)

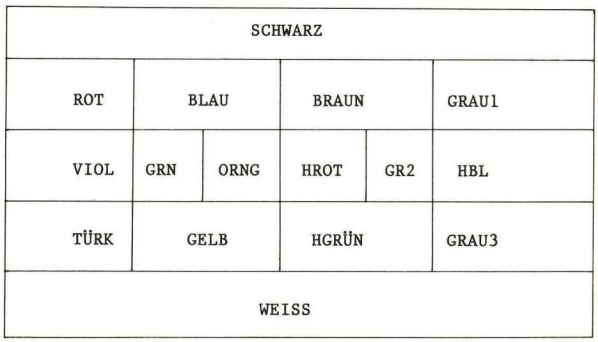
```
10 PRINT CHR$(147);:POKE53280,12:POKE53281,12
20 PRINT CHR$(18) CHR$(144);"
                    ";
30 PRINT CHR$(18) CHR$(144);"
                    ";
40 PRINT CHR$(18) CHR$(28)"          "CHR$(31)"
       "CHR$(149)"          ";
45 PRINT CHR$(151)"          ";
50 PRINT CHR$(18) CHR$(28)"          "CHR$(31)"
       "CHR$(149)"          ";
55 PRINT CHR$(151)"          ";
60 PRINT CHR$(18) CHR$(156)"       "CHR$(30)"      "
    CHR$(129)"        ";
65 PRINT CHR$(150)"       "CHR$(152)"      "CHR$(154
    )"       ";
70 PRINT CHR$(18) CHR$(156)"       "CHR$(30)"      "
    CHR$(129)"       ";
75 PRINT CHR$(150)"       "CHR$(152)"      "CHR$(154
    )"       ";
80 PRINT CHR$(18) CHR$(159)"          "CHR$(158)"
          ";
85 PRINT CHR$(153)"       "CHR$(155)"          ";
90 PRINT CHR$(18) CHR$(159)"          "CHR$(158)"
          ";
95 PRINT CHR$(153)"          "CHR$(155)"          ";
100 PRINT CHR$(18) CHR$(5)"
                    ";
110 PRINT CHR$(18) CHR$(5)"
                    ";
```

Damit beim Abtippen der Leerzeichen nicht allzuviel gerätselt werden muß, gebe ich nachfolgend geordnet nach Zeilennummern in der entsprechenden Reihenfolge ihre Länge an:

Zeile

10	0 Leerzeichen
20	40 Leerzeichen
30	40 Leerzeichen
40	10 Leerzeichen, 10 Leerzeichen, 10 Leerzeichen
45	10 Leerzeichen,
50	10 Leerzeichen, 10 Leerzeichen, 10 Leerzeichen
55	10 Leerzeichen
60	7 Leerzeichen, 6 Leerzeichen, 7 Leerzeichen
65	7 Leerzeichen, 6 Leerzeichen, 7 Leerzeichen
70	7 Leerzeichen, 6 Leerzeichen, 7 Leerzeichen
75	7 Leerzeichen, 6 Leerzeichen, 7 Leerzeichen
80	10 Leerzeichen, 10 Leerzeichen
85	10 Leerzeichen, 10 Leerzeichen
90	10 Leerzeichen, 10 Leerzeichen
95	10 Leerzeichen, 10 Leerzeichen
100	40 Leerzeichen
110	40 Leerzeichen

So sieht das Bild aus, wenn das Programm gelaufen ist:



Programm 4 (FARBDEM1)

```
10 POKE 53265,59
20 POKE 53272,31
30 POKE 53280,12
40 POKE 53281,12
50 FOR I=8192 TO 16191
60 POKE I,A
70 LET A=255-A
80 NEXT I
90 FOR I=1024 TO 2023
100 POKE I,204
110 NEXT
120 FOR J=0 TO 6
130 FOR I=0 TO 9
140 GOSUB 230
150 FOR Y=J*3 TO J*3+2
160 FOR X=I*4 TO I*4+3
170 POKE 1024+40*Y+X,C
180 NEXT X
190 NEXT Y
200 NEXT I
210 NEXT J
220 GOTO 220
230 READ C$
240 LET C1=ASC(LEFT$(C$,1))
250 GOSUB 310
260 LET C=C1
270 LET C1=ASC(RIGHT$(C$,1))
280 GOSUB 310
290 LET C=C*16+C1
300 RETURN
310 IF ((48<=C1) AND (C1<=57)) THEN LET C1=C1+7
320 LET C1=C1-55
330 RETURN
10000 DATA 00,11,22,26,29,2B,33,37,3D,3F
10010 DATA 44,45,48,4A,4C,4E,54,55,58,5A
10020 DATA 5C,5E,62,66,69,6B,73,77,7D,7F
10030 DATA 84,85,88,8A,8C,8E,92,96,99,9B
10040 DATA A4,A5,A8,AA,AC,AE,B2,B6,B9,BB
10050 DATA C4,C5,C8,CA,CC,CE,D3,D7,DD,DF
10060 DATA E4,E5,E8,EA.EC.EE.F3.F7.FD.FF
```

Unterschiedliche DATA-Anweisungen für Programm 4:

(FARBDEM2)

```
10000 DATA 44,00,45,11,48,22,4A,26,4C,4E
10010 DATA 29,54,2B,55,62,58,66,5A,69,5C
10020 DATA 5E,6B,84,92,85,96,88,99,8A,9B
10030 DATA B2,8C,B6,8E,B9,A4,BB,A5,33,A8
10040 DATA AA,37,AC,3D,AE,3F,C4,73,C5,77
10050 DATA 7D,C8,7F,CA,D3,CC,D7,CE,DD,E4
10060 DATA E5,DF,E8,F3,EA,F7,EC,FD,EE,FF
```

Bei diesen DATA-Anweisungen werden Farbpaare (XX und YY) ausgetauscht:

(FARBDEM3)

```
10000 DATA 00,11,22,33,44,55,66,77,88,99
10010 DATA AA,BB,CC,DD,EE,FF,26,62,29,92
10020 DATA 2B,B2,69,96,6B,B6,9B,B9,37,73
10030 DATA 3D,D3,3F,F3,7D,D7,7F,F7,DF,FD
10040 DATA 45,54,48,84,4A,A4,4C,C4,4E,E4
10050 DATA 58,85,5A,A5,5C,C5,5E,E5,8A,A8
10060 DATA 8C,C8,8E,E8,AC,CA,AE,EA,CE,EC
```

(FARBDEM4)

```
10000 DATA 00,22,26,29,2B,62,66,69,6B,92
10010 DATA 96,99,9B,B2,B6,B9,BB,44,45,48
10020 DATA 4A,4C,4E,54,55,58,5A,5C,5E,84
10030 DATA 85,88,8A,8C,8E,A4,A5,A8,AA,AC
10040 DATA AE,C4,C5,C8,CA,CC,CE,E4,E5,E8
10050 DATA EA,EC,EE,33,37,3D,3F,73,77,7D
10060 DATA 7F,D3,D7,DD,DF,F3,F7,FD,FF,11
```

2-D-Tabelle für gemischte Farben

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 SCHWARZ	X															
1 WEISS		X														
2 ROT			X				X			X		X				
3 TÜRKIS				X				X						X		X
4 VIOLETT					X	X			X		X		X		X	
5 GRÜN					X	X			X		X		X		X	
6 BLAU			X				X			X		X				
7 GELB				X				X						X		X
8 ORANGE					X	X			X		X		X		X	
9 BRAUN			X				X			X		X				
A HROT					X	X			X		X		X		X	
B GRAU1			X				X			X		X				
C GRAU2					X	X			X		X		X		X	
D HGRÜN				X				X						X		X
E HBLAU					X	X			X		X		X		X	
F GRAU3				X				X						X		X

X bezeichnet die möglichen Kombinationen

C 64 Farbcode-Tabelle

Code (dezimal)	Code (hexadezimal)	Farbe
0	0	Schwarz
1	1	Weiß
2	2	Rot
3	3	Türkis
4	4	Violett
5	5	Grün
6	6	Blau
7	7	Gelb
8	8	Orange
9	9	Braun
10	A	Hellrot
11	B	Graul
12	C	Grau2
13	D	Hellgrün
14	E	Hellblau
15	F	Grau3

Tabelle der gemischten Farben

Code dezimal	Code Hexadezimal	Code Binär	Gemischte Farben
0	00	00000000	Schwarz + Schwarz
17	11	00010001	Weiß + Weiß
34	22	00100010	Rot + Rot
38	26	00100110	Rot + Blau
41	29	00101001	Rot + Braun
43	2B	00101011	Rot + Graul
51	33	00110011	Türkis + Türkis
55	37	00110111	Türkis + Gelb
61	3D	00111101	Türkis + Hellgrün
63	3F	00111111	Türkis + Grau3
68	44	01000100	Violett + Violett
69	45	01000101	Violett + Grün
72	48	01001000	Violett + Orange
74	4A	01001010	Violett + Hellrot
76	4C	01001100	violett + Grau2
78	4E	01001110	Violett + Hellblau
84	54	01010100	Grün + Violett
85	55	01010101	Grün + Grün
88	58	01011000	Grün + Orange
90	5A	01011010	Grün + Hellrot
92	5C	01011100	Grün + Grau2
94	5E	01011110	Grün + Hellblau
98	62	01100010	Blau + Rot
102	66	01100110	Blau + Blau
105	69	01101001	Blau + Braun
107	6B	01101011	Blau + Graul
115	73	01110011	Gelb + Türkis
119	77	01110111	Gelb + Gelb
125	7D	01111101	Gelb + Hellgrün
127	7F	01111111	Gelb + Grau3
132	84	10000100	Orange + Violett
133	85	10000101	Orange + Grün
136	88	10001000	Orange + Orange
138	8A	10001010	Orange + Hellrot
140	8C	10001100	Orange + Grau2
142	8E	10001110	Orange + Hellblau
146	92	10010010	Braun + Rot

Code dezimal	Code Hexadezimal	Code Binär	Gemischte Farben
150	96	10010110	Braun + Blau
153	99	10011001	Braun + Braun
155	9B	10011011	Braun + Graul
164	A4	10100100	Hellrot + Violett
165	A5	10100101	Hellrot + Grün
168	A8	10101000	Hellrot + Orange
170	AA	10101010	Hellrot + Hellrot
172	AC	10101100	Hellrot + Grau2
174	AE	10101110	Hellrot + Hellblau
178	B2	10110010	Graul + Rot
182	B6	10110110	Graul + Blau
185	B9	10111001	Graul + Braun
187	BB	10111011	Graul + Graul
196	C4	11000100	Grau2 + Violett
197	C5	11000101	Grau2 + Grün
200	C8	11001000	Grau2 + Orange
202	CA	11001010	Grau2 + Hellrot
204	CC	11001100	Grau2 + Grau2
206	CE	11001110	Grau2 + Hellblau
211	D3	11010011	Hellgrün + Türkis
215	D7	11010111	Hellgrün + Gelb
221	DD	11011101	Hellgrün + Hellgrün
223	DF	11011111	Hellgrün + Grau 3
228	E4	11100100	Hellblau + Violett
229	E5	11100101	Hellblau + Grün
232	E8	11101000	Hellblau + Orange
234	EA	11101010	Hellblau + Hellrot
243	F3	11110011	Grau3 + Türkis
247	F7	11110111	Grau3 + Gelb
253	FD	11111101	Grau3 + Hellgrün
255	FF	11111111	Grau3 + Grau3

Programm 5 (FARBSET)

```
10 POKE 53265,59
20 POKE 53272,31
30 POKE 53280,12
40 POKE 53281,12
50 FOR I=8192 TO 16191
60 POKE I,A
70 LET A=255-A
80 NEXT I
90 FOR I=1024 TO 2023
100 POKE I,204
110 NEXT
120 FOR J=0 TO 6
130 FOR I=0 TO 9
140 GOSUB 230
145 GOSUB 150 : GOTO 200
150 FOR Y=J*3 TO J*3+2
160 FOR X=I*4 TO I*4+3
170 POKE 1024+40*Y+X,C
180 NEXT X
190 NEXT Y
195 RETURN
200 NEXT I
210 NEXT J
220 GOTO 1000
230 READ C$
240 LET C1=ASC(LEFT$(C$,1))
250 GOSUB 310
260 LET C=C1
270 LET C1=ASC(RIGHT$(C$,1))
280 GOSUB 310
290 LET C=C*16+C1 : LET C(I,J)=C
300 RETURN
310 IF ((48<=C1) AND (C1<=57)) THEN LET C1=C1+7
320 LET C1=C1-55
330 RETURN
1000 POKE 2040,254 : POKE 2041,255
1010 FOR I=0 TO 62
1020 READ A : POKE 16256+I,A
1030 NEXT I
1040 FOR I=0 TO 62
1050 READ A : POKE 16320+I,A
1060 NEXT I
```

```
1070 LET CS=0
1080 POKE 53269,3
1090 LET X(0)=0 : LET X(1)=0 : LET Y(0)=0 : LET Y(1)=0
1100 POKE 53287,1-CS : POKE 53288,CS
1110 FOR I=0 TO 1
1120 POKE 53248+2*I,(X(I)*32+28)AND255
1130 POKE 53249+2*I,Y(I)*24+52
1140 IF X(I)*32+28 > 255 THEN POKE 53264,PEEK(5
3264)OR(2^I) : GOTO 1160
1150 POKE 53264,PEEK(53264) AND (255-2^I)
1160 NEXT I
1170 GET A$
1180 IF A$=CHR$(133) THEN LET CS=0 : GOTO 1100
1190 IF A$=CHR$(134) THEN LET CS=1 : GOTO 1100
1200 IF A$=CHR$(29) THEN LET X(CS)=X(CS)+1 : IF
 X(CS)=10 THEN LET X(CS)=9
1210 IF A$=CHR$(29) THEN 1100
1220 IF A$=CHR$(157) THEN LET X(CS)=X(CS)-1 : I
F X(CS)=-1 THEN LET X(CS)=0
1230 IF A$=CHR$(157) THEN 1100
1240 IF A$=CHR$(17) THEN LET Y(CS)=Y(CS)+1 : IF
 Y(CS)=7 THEN LET Y(CS)=6
1250 IF A$CHR$(17) THEN 1100
1260 IF A$CHR$(145) THEN LET Y(CS)=Y(CS)-1 : IF
 Y(CS)=-1 THEN LET Y(CS)=0
1270 IF A$=CHR$(145) THEN 1100
1280 IF A$="C" THEN 1300
1290 GOTO 1170
1300 LET I=X(0) : LET J=Y(0)
1310 LET C=C(X(1),Y(1))
1320 GOSUB 150
1330 LET I=X(1) : LET J=Y(1)
1340 LET C=C(X(0),Y(0))
1350 GOSUB 150
1360 LET C(X(0),Y(0))=C(X(1),Y(1))
1370 LET C(X(1),Y(1))=C
1380 GOTO 1170
10000 DATA 00,11,22,26,29,2B,33,37,3D,3F
10010 DATA 44,45,48,4A,4C,4E,54,55,58,5A
10020 DATA 5C,5E,62,66,69,6B,73,77,7D,7F
10030 DATA 84,85,88,8A,8C,8E,92,96,99,9B
10040 DATA A4,A5,A8,AA,AC,AE,B2,B6,B9,BB
10050 DATA C4,C5,C8,CA,CC,CE,D3,D7,DD,DF
10060 DATA E4,E5,E8,EA,EC,EE,F3,F7,FD,FF
11000 DATA 255,255,255,255,255,255
11010 DATA 192,0,3,192,0,3
```

Stichwortverzeichnis

2
2D-Grafik 126

3
3D-Programm 128
3D-Grafik 148

A
Abschalten
- Bildschirm 121
- Interrupt 149
Abspeichern/Laden 169
AND 45
AND-Funktion 104
AUTONUMBER-Befehl 161

B
Basic-Interpreter 95
Basic-Programm 40
Basic-RAM 18
Basic-ROM 17, 80
Basic-Unterprogramme 85
Basic-Warmstartadresse 38
Betriebssystem 16, 38
Betriebssystemroutine 122, 176
Bildpunkt 113
Bildschirmcode 20, 29, 61
Bildschirmfarbspeicher 59
Bildschirmkoordinaten 69
Bildschirmspeicher 20, 23, 150
Bildverschiebung 122
Binärausdruck 23
Binärsystem 33
Binärzahl 101
Bit-Map 81, 150, 153, 174
Bit-Map-Löschen 88
Bit-Map-Mehrfarben-Modus 83
Bit-Map-Modus 64, 84
Bit-Mapping 63
Bit-Numerierung 37
Bit-Raster 101

C
CIA 22, 209
COMMODORE-Taste 57, 60, 62
Complex Interface Adapter 6526 22
CTRL-Taste 57, 60
Cursor 27, 40, 122, 210
Cursorsteuerzeichen 122

D
Datenrichtungsregister 77
Dezimalzahl 33, 37
Direktmodus 40, 57
Dollarzeichen 35

E
Eigene Zeichen definieren 48
Ein- und Ausgabebausteine. 16
Elektronenstrahl 125, 153
Ellipse
- löschen 92
- zeichnen 91
Entstehung eines Fernsehbilds 125
ERROR 107

F
Farb-RAM 22
Farbinformation 150
Farbkennzahl 27
Fehlermeldung 23, 89
Fehlerquelle 121
Fließkomma-Arithmetik 95
Fluchtpunktperspektive 134
Frei wählbare Koordinatensysteme 169
FRETOP 43
Funktionstastenbelegung 160

G
Geisterbild 28, 65
Gleitkomma-Arithmetik 170
Gleitkommazahl 170
Grafik-Befehle 163
Grafik-Software 13
Grafik-Unterprogramm-Bibliothek 85, 92
Grafik-Unterstützung 159
Grafikmöglichkeiten 14

H
Hardcopy-Routine 174
Hex-Zahl 36
Hexadezimalsystem 35
Hilfsfunktionen 162
Hintergrundfarbe 28, 62, 204
Hintergrundfarbregister 60
Hintergrundregister 61
Hinterschneidungsproblem 145
HIRES an 88
HIRES aus 89
HIRES-3 159
HIRES-Modus 66, 209
Hochauflösende Grafik 63, 75
Hochauflösender Grafikmodus 163
Hochauflösungs-Modus 65, 150

I
IF...THEN...-Verzweigung 45
Indizierte Variable 40
Interpreter-Routinen 176, 178
Interrupt 152
Interrupt-Maske 152
Interrupt-System 49
Interrupt-Vektor 154

K
Kassettenpuffer 151
Kavaliers-Perspektive 134
Kernal-Sprungadresse 180
Kernal-ROM 16
Kollisionsregister 116
Koordinatensystem 105, 130
Kopieren 49

L
Least significant Bit 35
Least significant Byte 38
Linien
- senkrechte 199
- waagrechte 201
LIST-Kommando 152
Logische Operatoren 45
LSB 38
Lsb 35
Löschen im Bildschirmsystem 168
Löschen
- ausgefülltes Rechteck 116, 173
- Ellipse 92, 168 173
- Funktion 173
- Linie 168, 173
- Punkt 168, 173
- Radius 168, 173
- Rechteck 168, 173

M
Maschinenprogramm 161
Maschinensprache 95, 154
Maske 46
Matrix 29
Mehrfarben-Zeichen 58
Mehrfarben-Modus 83, 110, 112
Mehrfarben-Zeichen-Modus 57, 62
MEMSIZ 43
MERGE-Befehl 161
MOB 99
Most significant Bit 35
Most significant Byte 38
MSB 38
Msb 35

N
Netzgrafik 149
Nibble 67
Normalmodus 94
Normalzustand 123, 124

O
OLD-Befehl 161
OR 45
OR-Funktion 104

P
Pentagramm 101, 111
Pixel 113
Potenz 33
PRINT FRE(0) 42
Prioritäten von Sprites 115
Prozessorstapelspeicher 18
Punkt löschen 90
Punkt setzen 69, 89
Punktmuster 29

R
Rahmen 29
Rasterinterrupt 152
Rasterzeile 125, 153
Rasterzeilen-Interrupt 155
Rasterzeilenregister 125
Rechteck ausfüllen 166, 172
Registerübersicht 24
RENUMBER-Befehl 160
RESTORE-Taste 65, 125, 178, 209
RUN/STOP-Taste 65, 94, 125, 150, 178, 209

S
Schrittweite 140
Schutz-POKE 43, 75, 83, 159
Schützen des RAMs 49
Scrollen 124
Sechzehnfingerlinge 36
Selbstdefiniertes System 171
SHIFT-Taste 62
SID 22
Sinus-Funktion 126
Sinuskurve 127
Sound Interface Device 6581 22
Speicherkapazität 15
Speicherorganisation 15, 40
Speichertest 19, 29, 52
Speicherzelle 71
Sprite 99
Sprite-Bildpunkt 116
Sprite-Daten-Verschiebung 114
Sprite-Editor 102
Sprite-Eigenheit 114
Sprite-Farben-Register 110
Sprite-Kontrollregister 107
Sprite-Zeiger 104
Sprite/Hintergrund-Kollision 153
Sprite/Sprite-Kollision 153
Spritedefinitionsfeld 107
Sprungtabelle 88
Stack 18
Strecke löschen 91
Strecke zeichnen 90
SYNTAX ERROR 80, 160, 164
SYS-Kommando 122

T
Taktfrequenz 95
Tastaturpuffer 130
Transformationsgleichung 130
Trickfilm 155

V
Variable 40
Verdeckte Kanten 145
VIC-II-Chip 22, 58, 64, 75, 100, 104, 115, 154, 209

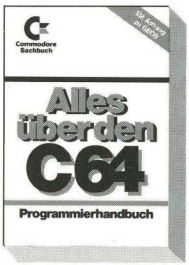
Video-Interface-Controller 6567 22
Video-RAM 84

X
X-Y-Ebene 135

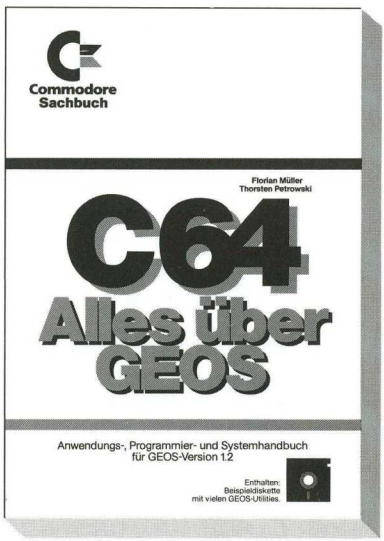
Z
Zahlencode 101
Zehnfingerlinge 33
Zeichen-ROM 17, 28, 48, 83, 151
Zeichengenerator-ROM 29
Zeichentrickfilm 114
Zeichnen
- ausgefülltes Rechteck 166, 172
- Ellipse 91, 166, 172
- Linie 165, 172
- Punkt, 164, 171
- Radius 166, 172
- Rechteck 166, 172
Zeropage 18, 40, 65
Zeropage-Adressen 175, 176
Zweifingerlinge 33

EXKLUSIV bei Markt&Technik

Commodore-Sachbücher

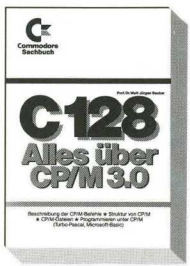


Commodore Sachbuchreihe
Alles über den C64
2. Auflage 1986, 514 Seiten
Dieses umfangreiche Grundlagenbuch zum C64 enthält neben einem BASIC-Lexikon alle Informationen und Tips, die der Spezialist zur Grafik- und Musikprogrammierung benötigt. Ein Kapitel beschäftigt sich mit der Programmierung in Maschinensprache und der Einbindung von Maschinensprache-Routinen in BASIC-Programme. In diesem Zusammenhang erfahren Sie auch alles über einen wichtigen Bestandteil des Betriebssystems aller Commodore-Computer, das »Kernal«.
Bestell-Nr. 90379
ISBN 3-89090-379-7
DM 59,–
(sFr 54,30/öS 460,20)



F. Müller/T. Petrowski
Alles über GEOS: Anwendungs-, Programmier- und Systemhandbuch
1987,
461 Seiten,
inklusive Diskette

Das umfassende Buch über Anwendung und Programmierung der grafischen Benutzeroberfläche GEOS.
Best.-Nr. 90461,
ISBN 3-89090-461-0
DM 49,–
(sFr 45,10/öS 382,20)



Prof. Dr. W.-J. Becker
C128 – Alles über CP/M 3.0
1986, 299 Seiten
Eine fundierte Einführung in die Anwendung des Betriebssystems CP/M 3.0 bzw. CP/M Plus auf dem Commodore 128.
Bestell-Nr. 90370
ISBN 3-89090-370-3
DM 52,–
(sFr 47,80/öS 405,60)

Dr. Ruprecht
C128-ROM-Listing
1986, 456 Seiten
Dieses kommentierte ROM-Listing umfaßt das Betriebssystem des C128, den Monitor des C128 sowie das BASIC 7.0 von Microsoft.
Best.-Nr. 90212
ISBN 3-89090-212-X
DM 58,–
(sFr 53,40/öS 452,40)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

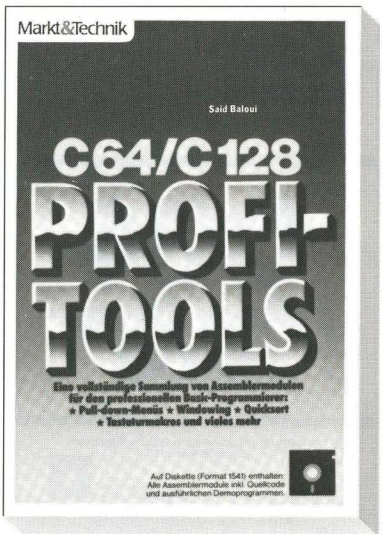
Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

7053S6

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bücher zum Commodore 64/128

Prof. F. Nestle/D. Pohlmann
C64/C128 Comal 80 Programmierpraxis
1987, 192 Seiten, inkl. Disk.
Wenn Sie die Einfachheit von Basic mit dem Komfort von Logo oder Pascal verbinden wollen, treffen Sie mit Comal eine gute Wahl. Comal ist durch seine Spracheigenschaften besonders für die Schule geeignet und wird in großem Umfang statt Basic eingesetzt. Das Buch führt Sie problemorientiert mit Beispielen und Strukturprogrammen in das moderne Prozedurkonzept von Comal ein. Besonders wird auf die praktischen Möglichkeiten der Sprache eingegangen. Viele instruktive Beispiele ergänzen die Theorie.
Bestell-Nr. 90511
ISBN 3-89090-511-0
DM 49,–
(sFr 45,10/öS 382,20)



S. Baloui
64'er Profi-Tools
1988, 156 Seiten, inkl. Disk.
Eine vollständige Sammlung von Assembler-Routinen für professionelle Basic-Programmierer. Aus dem Inhalt: Pull-down-Menüs, Windowing, Quicksort, Tastatur-Makros und vieles mehr. Auf Diskette sind alle Assemblermodule inkl. Quellcode und ausführlichen Demoprogrammen enthalten.
Bestell-Nr. 90617
ISBN 3-89090-617-6
DM 49,–*
(sFr 45,10*/öS 417,00*)

S. Vilsmeier
C64/C128 Objekt-Bibliotheken zu Giga-CAD Plus
1988, 64 Seiten,
inkl. zwei Disketten
Eine Sammlung von neuen Objekten, Zeichensätzen und Utilities für das bekannte Konstruktionsprogramm Giga-CAD Plus. Dieses Buch beschreibt eine Reihe nützlicher Utilities und Erweiterungen wie die Filmroutine »Title Wizard« und den »Film-Converter«. Die mitgelieferten Construction-Sets sind auf zwei doppelseitig bespielten Disketten enthalten.
Bestell-Nr. 90581
ISBN 3-89090-581-1
DM 39,–*
(sFr 35,90*/öS 331,90*)
* Unverbindliche Preisempfehlung

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

403/804

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bücher zum Commodore 64/128

S. Vilsmeier
3D-Konstruktion mit GIGA-CAD Plus auf dem C64/C128
1986, 370 Seiten, inkl. 2 Disk.
Mit GIGA-CAD können Computergrafiken von besonderer Räumlichkeit und Faszination geschaffen werden. GIGA-CAD Plus ist schneller und einfacher zu bedienen, die Benutzeroberfläche wurde verbessert und der Befehlssatz erweitert. Die Eingabe erfolgt in erster Linie über den Joystick. Hardware-Anforderung: C64 mit Floppy 1541 oder C128 (im 64'er-Modus), Fernseher oder Monitor, Joystick und Commodore- oder Epson-kompatibler Drucker.
● Das verbesserte GIGA-CAD-Programm mit neuen Features wie erweitertem Befehlssatz und bis zu 10mal schneller liegt dem Buch im Floppy-1541-Format bei.
Best.-Nr. 90409
ISBN 3-89090-409-2
DM 49,–
(sFr 45,10/öS 382,20)

H. Haberl
Mini-CAD mit Hi-Eddi plus auf dem C64/C128
1986, 230 Seiten, inkl. Diskette
Auf der beiliegenden Diskette findet der Leser das vollständige Zeichenprogramm »Hi-Eddi«, mit dem das komfortable Erstellen von technischen Zeichnungen, Plänen oder Diagrammen ebenso möglich ist wie das Malen von farbigen Bildern, Entwurf und Ausdruck von Glückwunschkarten, Schildern, ja sogar von bewegten Sequenzen (kleine Trickfilme, Schaufenster-Werbung).
● Wer sagt, daß CAD auf dem C64 nicht möglich ist?!
Best.-Nr. 90136
ISBN 3-89090-136-0
DM 48,–
(sFr 44,20/öS 374,40)

B. Bornemann-Jeske
Vizawrite-Buch für den C64/C128
1987, 228 Seiten
Mit dem »Vizawrite-Buch« liegt erstmals ein vollständiges und detailliertes Arbeitsbuch für den Anfänger und den professionellen Anwender zur Textverarbeitung auf dem C64/C128 vor. Die Grundlagenkapitel führen Sie anhand kurzer Übungsaufgaben in die elementaren Funktionen des Systems ein. Das Kapitel für Fortgeschrittene zeigt Ihnen jede Programmfunktion im Detail. Zahlreiche praktische Tips aus verschiedenen Anwendungsbereichen ermöglichen Ihnen die optimale Nutzung Ihres Textverarbeitungssystems.
Best.-Nr. 90231
ISBN 3-89090-231-6
DM 49,–
(sFr 45,10/öS 382,20)

O. Hartwig
Experimente zur Künstlichen Intelligenz mit C64/C128
1987, 248 Seiten
Sind Maschinen intelligent? Können Computer denken? Erschließen Sie sich eines der interessantesten Gebiete der modernen Computerforschung! Anhand zahlreicher Programme erfahren Sie hier die Möglichkeiten der Künstlichen Intelligenz, speziell auf dem C64 und dem C128. Der Schwerpunkt des Buches liegt auf der Praxis. Alle KI-Techniken werden durch anschauliche Programme vorgestellt, die sofort nachvollziehbar sind. Zusätzlich erhalten Sie jede Menge Anregungen zu eigenen Experimenten. Die KI-Programme können ohne weiteres in eigene Programme integriert werden.
Best.-Nr. 90472
ISBN 3-89090-472-6
DM 49,–
(sFr 45,10/öS 382,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

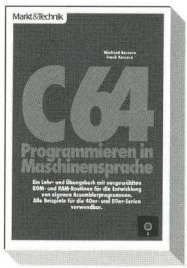
706358

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

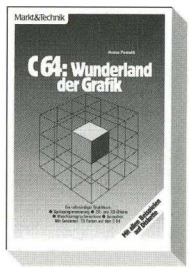
Bücher zum Commodore 64/128

M. Hegenbarth/R. Trierscheid
BASIC-Grundkurs mit dem C64
1985, 377 Seiten
Kein rein theoretisch ausgelegter BASIC-Kurs, sondern praxisnah auf den C64 zugeschnitten. Auch der Computerneuling kann mit diesem Buch lernen, mit seinem C64 in BASIC zu arbeiten, und wird auf die Besonderheiten seines Computers hingewiesen. Der leichtverständliche, lockere Stil und die gute logische Gliederung der Kapitel unterstützen dies. Erwähnenswert ist ein Kapitel, das die Kommunikation zweier C64 beschreibt, der Anhang, in dem eine Liste nützlicher PEEKs, POKEs und SYS und noch vieles mehr enthalten ist.
● Für den Lesertyp, der beim Lernen auch noch Spaß haben möchte.
Best.-Nr. 90361
ISBN 3-89090-361-4
DM 44,-
(sFr 40,50/öS 343,20)

F. Matthes
Pascal mit dem C64
1986, 215 Seiten, inkl. Diskette
Buch und Compiler ermöglichen jedem Besitzer eines C64 den Einstieg in die moderne Programmiersprache Pascal. Der Compiler akzeptiert den gesamten Sprachumfang mit einigen Erweiterungen. Er bildet mit einem sehr komfortablen Full-Screen-Editor eine schnelle Einheit, so daß der Programmentwicklungsaufwand minimal ist. Übersetzte Programme laufen ohne weitere Hilfsprogramme auf jedem C64, nutzen den gesamten Programmspeicher des C64 und sind 3-4mal schneller als vergleichbare Programme in BASIC. Dem Buch liegt ein leistungsfähiges Pascal-System mit einigen Pascal-Programmen auf Diskette bei.
Best.-Nr. 90222
ISBN 3-89090-222-7
DM 52,-
(sFr 47,80/öS 405,60)



W. Kassera/F. Kassera
C64-Programmieren in Maschinensprache
Der Aufschwung im Programmieren stellt sich ein, wenn Sie die betriebssysteminternen ROM-Routinen kennen, über ihre Funktionsweise und ihr Zusammenspiel informiert sind. Und Sie müssen die Maschinensprache Ihres C64 beherrschen. Beides ermöglicht Ihnen dieses Buch. Es zeigt, wie Sie bewegte Bildschirmobjekte programmieren, die Interrupt-Routine des Systems erweitern, die Arithmetik-Routinen im ROM und deren Datentypen beherrschen, und alles, was Sie über Ein-/Ausgabe, BASIC-Variable und andere wichtige Themen wissen müssen.
Best.-Nr. 90168
ISBN 3-89090-168-9
DM 52,-
(sFr 47,80/öS 405,60)



H. Ponnath
C64: Wunderland der Grafik
1985, 232 Seiten, inkl. Diskette
Der Autor legt beim Leser ein solides Fundament an Wissen, und er tut dies auf so unterhaltsame Art, daß Sie bestens gerüstet sind, um so interessante Aufgaben wie die Programmierung hochauflösender zwei- und dreidimensionaler Grafiken anzugehen. Mit Sprites zu jonglieren ist für Sie bald kein Problem mehr, aber auch das vertrackte Verdeckungsproblem bei dreidimensionaler Grafik kriegen Sie jetzt endlich in den Griff. Finden Sie heraus, was wirklich im Grafik-Chip Ihres C64 steckt!
● Eine lesenswerte und kenntnisreiche Einführung in dieses hochinteressante Thema von einem sachkundigen Autoren; mit allen Beispielen auf beigefügter Diskette.
Best.-Nr. 90363
ISBN 3-89090-363-0
DM 49,-
(sFr 45,10/öS 382,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

706359

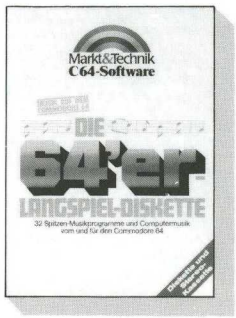
Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Die 64'er-Langspiel-Diskette

ACHTUNG!
Computer-Freaks aufgepaßt:
32 Spitzen-Musikprogramme aus dem 64'er-Musik-Programmierwettbewerb auf einer Diskette mit komfortablem Lademenü. Von Pop bis Klassik ist für jeden Musikgeschmack etwas dabei: Shades, This is not America, Invention Nr. 13, Mondscheinsonate, You can win if you want, Der Clou, Für Elise, The pink Panther und viele mehr.

Hardware-Anforderungen:
Commodore 64 oder Commodore 128 im C-64-Modus, Floppy-Station 1541, 1570 oder 1571

Ein »Muß«
für jeden 64'er-Fan!



Best.-Nr. 39630
DM 39,90*
(sFr 34,90*/öS 399,-*)
* Unverbindliche Preisempfehlung

Einmalig in
der Computergeschichte:
- Alle Musikstücke werden in Stereoqualität auf einer hochwertigen Kassette mit Rauschunterdrückung mitgeliefert!
- Eineinhalb Stunden erstklassige Computermusik!
- Klang umwerfend!

Lieferumfang:
1 Diskette beidseitig bespielt mit 32 Musikstücken
1 Kassette mit allen Musikstücken in Stereoqualität für handelsübliche Kassettenrecorder oder Stereoanlagen

Markt&Technik-Softwareprodukte erhalten Sie in den Fachabteilungen der Kaufhäuser, in Computershops oder im Buchhandel.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

706229-2

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

64'er C= Spielesammlung

Lassen Sie sich in eine abenteuerliche Spielewelt entführen!

64'er
64'er-Spielesammlung Band 1
1987, 115 Seiten, inkl. Disk.
Mit den 15 spannenden Spielen und der ausführlichen Anleitung ist Ihnen ein fantastisches Spielvergnügen gewiß: Balliard: Einfallswinkel = Ausfallswinkel - Wer das nicht befolgt, hat es schwer mit dieser Mischung aus Tennis und Billard. The Way: Zu verschlungenen Pfaden gesellen sich Geldsäcke und böse Geister, die es zu bekämpfen gilt. Fireburg: Hoffentlich fängt Ihr Joystick nicht Feuer, wenn es heißt, die wertvollen Koffer aus dem brennenden Haus des Professors zu erwischen. Pirat: Taktik, Timing und gute Navigationskenntnisse sind Voraussetzung für ein langes Piratenleben. Und viele weitere spannende Spiele.
Bestell-Nr. 90429
ISBN 3-89090-429-7
DM 39,-*
(sFr 35,90*/304,20*)

64'er
64'er-Spielesammlung Band 2
1987, 98 Seiten, inkl. Disk.
Der zweite Band der Spielesammlung mit 14 aufregenden Spielen entführt Sie wieder in eine fantastische Action-Welt.
Bestell-Nr. 90428
ISBN 3-89090-428-9
DM 39,-*
(sFr 35,90*/öS 304,20)

64'er
C64-Spielesammlung Band 3
1988, 103 Seiten, inkl. Disk.
Auch mit der dritten Spielesammlung können Sie wieder in eine zauberhafte Spielewelt eintreten. Mannigfaltige Gefahren erwarten Sie. Angefangen bei einem Großwesir, über Zauberer, Agenten, bis hin zu gemeingefährlichen Verbrechern - für gute Unterhaltung und außergewöhnlichen Spielspaß mit den 12 Spielen ist also gesorgt! Und diesmal dürfen Sie bei fast allen Ihren Reisen sogar einen Partner mitnehmen!
Bestell-Nr. 90596
ISBN 3-89090-596-X
DM 39,-*
(sFr 35,90*/öS 304,20)

*** Unverbindliche Preisempfehlung**

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

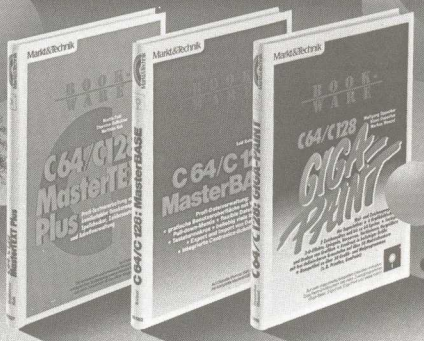
804325

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

BOOK-WARE

Haben Sie schon mal Profi-Software zum Buchpreis gekauft?

»Bookware« – das sind professionelle Programme zum Preis eines Buches!



M. Pahl, T. Rullkötter, M. Kuk
C64/C128 MasterText Plus
1988, 201 Seiten, inkl. Diskette
MasterText Plus – die leistungsfähige Textverarbeitung: 40-Zeichen- und 80-Zeichen-Ausgabe – Suchen und Ersetzen – Silbentrennung – Blockoperationen – Formularverwaltung – integrierte Centronics-Schnittstelle – jetzt mit Rechtschreibkorrektur und Adreßverwaltung – Komprimieren von Texten – individuelle Farbgebung und Druckeranpassung – freie Tastenbelegung – Zeichensatz-Editor – komfortable Druckeranpassung: Druckertreiber für MPS 801, MPS 802, Epson-Drucker und Kompatible.
Bestell-Nr. 90527, ISBN 3-89090-527-7
DM 59,–* (sFr 54,30*/öS 502,–*)

S. Baloui
C64/C128 MasterBase
1988, 155 Seiten, inkl. Diskette
Die professionelle Dateiverwaltung für den C64/C128. Besondere Leistungsmerkmale: integrierte Centronics-Schnittstelle – Export und Import von Daten – nachträgliche Veränderung der Struktur einer bereits bestehenden Datei – Tastatur-Makros – einfache Bedienung über Windows und Pull-down-Menüs – als einzige Dateiverwaltung für den C64 erlaubt Ihnen MasterBase, beliebig viele Indexfelder zu verwenden (extrem schnelle Suche nach bestimmten Daten; selbst größte Dateien werden in Nullzeit umsortiert).
Bestell-Nr. 90583, ISBN 3-89090-583-8
DM 59,–* (sFr 54,30*/öS 502,–*)

W. Oppacher, K. Oppacher, M. Wenzel
C64/C128 Giga Paint
1988, ca. 200 Seiten, inkl. 2 Disketten
Ein professionelles Mal- und Zeichenprogramm: stufenloses Verkleinern, Vergrößern und Verzerren – Zeichnen von Kurven durch beliebige Punkte und 3-D-Operationen unter Verwendung aller 16 Farben – Kompatibilität zu über 30 Grafikprogrammen – universelle Druckroutine für fast jeden grafikfähigen Drucker – Ausdruck beliebiger Bildausschnitte – frei definierbare Graustufen – Basic-Erweiterung – beliebige Positionierung von Bildschirmausschnitten – Programmierung flimmerfreier Rasterinterrupts und vieles mehr.
Bestell-Nr. 90619, ISBN 3-89090-619-2
DM 59,–* (sFr 54,30*/öS 502,–*)

* Unverbindliche Preisempfehlung

Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computerfachgeschäften oder bei Ihrem Buchhändler.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie bei Ihrem Buchhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0.
SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656.
ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 587 1393-0.
Rudolf Lechner & Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 67 7526

HEIMO PONNATH,
geboren 1944 in Hamburg, ist Physikochemiker. Seine erste Berührung mit Informatik fand 1966 statt. Seitdem fasziniert ihn der scheinbare Gegensatz zwischen den strengen Regeln der Mathematik und den vielfältigen Möglichkeiten der Grafik. Der Heimcomputer als Brücke dazwischen und als Mittel zur Kreativität für jeden verführte ihn zum Schreiben. Heute ist Ponnath als freier Journalist tätig.

C64 - Wunderland der Grafik

Grafik und der C64 – ein schier unerschöpfliches Thema! Dieses Buch zeigt eine Vielzahl sehr interessanter Lösungen, um die grafischen Möglichkeiten des Commodore 64 optimal zu nutzen.
Als Krönung enthält es ein zuschaltbares Assemblerprogramm, das umfangreiche grafische und einige neue BASIC-Befehle anbietet. Im zweiten Teil des Buches wird eine Möglichkeit gezeigt, wie man bis zu 70 verschiedene Farben erzeugen kann.
Viele Beispielprogramme begleiten die Reise durch das Wunderland der Grafik. Dieses Buch ist eine Fundgrube für alle ambitionierten C64-Benutzer, die wirklich das Letzte aus ihrem Rechner herausholen wollen.

Hardware-Anforderung:
Commodore 64 oder Commodore 128 im 64er-Modus, Diskettenlaufwerk Floppy 1541, 1570 oder 1571.
Zum Thema »Grafik auf dem C128« ist vom selben Autor das Buch »Grafik-Programmierung C128«, Bestell-Nr. MT 90202, im Markt&Technik-Verlag erschienen.

ISB N 3-89090-363-0

Markt&Technik

4 001057 903630

DM 49,-
sFr 45,10
öS 382,20