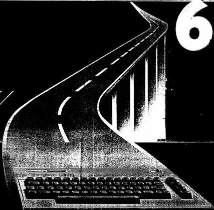
Brücke zum Commodore 64



Diejenigen Bezeichnungen von im Buch genannten Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Es kann also aus dem Fehlen der Markierung ® nicht geschlossen werden, daß die Bezeichnung ein freier Warenname ist. Ebensowenig ist zu entnehmen, ob Patente oder Gebrauchsmusterschutz vorliegen.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Thoma, Manfred Walter:
Brücke zum Commodore 64 : erw. Handbuch / Manfred Walter Thoma. – Heidelberg : Hüthig, 1985.
ISBN 3-7785-1095-9

© 1985 Dr. Alfred Hüthig Verlag GmbH, Heidelberg
Printed in Germany
Satz und Druck: Druckerei Bitsch GmbH, Birkenau

Einleitung

Vielen Anwendern des COMMODORE 64 ist es bis heute nicht gelungen, alle Möglichkeiten des Computers auszuschöpfen. Spezielle Techniken, die nicht mit „normalen" BASIC-Befehlen durchzuführen sind, blieben rätselhaft. Durch das überaus sparsame COMMODORE-BASIC V2.0 sind die wesentlichen Stärken des COMMODORE 64 wie,

- hochauflösende Grafik mit 320 * 200 Punkten,
- 8 freibewegliche Kleingrafiken (SPRITE),
- 3-stimmiger Synthesizer

nur mit vielen und teilweise sehr unverständlichen POKE's und PEEK's möglich. Das von COMMODORE mitgelieferte Handbuch bietet hier kaum eine Hilfe.

Das vorliegende Buch bietet Ihnen eine Zusammenfassung aller wesentlichen Programmiertechniken in den Bereichen

- Grafik
- Musik
- Diskette
- Cassette
- Joystick, Paddle, Lightpen, USER-Port etc.
- Speicherverwaltung des COMMODORE 64

Es ist ausschließlich in BASIC gehalten, so daß außer grundlegenden Kenntnissen der Programmiersprache BASIC keine weiteren Kenntnisse vorausge-

setzt werden. Für alle diejenigen von Ihnen, denen der Umgang mit binären und hexadezimalen Zahlen, sowie der logischen Arithmetik, nicht geläufig ist, findet in den Anhängen eine kurze Erläuterung der Begriffe und Zusammenhänge statt.

Das Buch kann als Arbeitsbuch im eigentlichen Sinne genutzt werden, d.h., Sie können es Stück für Stück durcharbeiten. Sie können es jedoch auch wie ein Handbuch nutzen, da alle Abschnitte weitestgehend in sich abgeschlossen sind.

Sie werden dem Inhaltsverzeichnis entnommen haben, daß hier ein sehr breiter Bereich bearbeitet worden ist. Sicherlich ist es möglich, über jeden einzelnen Abschnitt ein Buch oder eine Bücherreihe zu verfassen, wie weit es allerdings für einen Hobby- oder Freizeitprogrammierer sinnvoll ist, bleibt dahingestellt.

Ich hoffe, daß Sie mit Hilfe dieses Buches einen Einstieg in die Tiefen des COMMODORE 64 erhalten.

Es werden im gesamten Buch eine Menge von Adressen und Registern benutzt, dabei ist es leicht möglich, daß der Überblick verloren geht. Nehmen Sie dann die Register-Tabellen im Anhang zur Hilfe.

Inhaltsverzeichnis

Seite

(A) Speicheraufbau

A1.0	Mehr als nur ein „schwarzer Kasten"	15
A2.0	64 kByte RAM (Random Access Memory)	16
A2.1	ROM (Read Only Memory)	16
A3.0	Die ersten 2 kByte	19
A3.1	Die Zero-Page	19
A4.0	Das BASIC-RAM	20
A4.1	Speichern von Variablen	23
A4.2	Speichern von Arrays	26
A4.3	Löschen eines BASIC-Programms	28
A4.4	MERGE - Aneinanderhängen von Programmen	28
A4.5	Schützen eines Speicherbereichs	29
A5.0	BASIC- und KERNAL-ROM	30
A6.0	Ein-/Ausgabe-Bausteine und das Zeichen-ROM	32
A6.1	Der VIC II - Chip	34
A6.2	Verschieben des Bildschirmspeichers	35
A6.3	Arbeiten mit 8 Bildschirmen	36
A7.0	CIA - Complex Interface Adapter	37

(B) Grafik

B1.0	Wunderland Grafik	43
B2.0	Bildschirm- und Farb-RAM	44
B3.0	ASCII- und Bildschirmcode	47
B4.0	Neue Zeichen erstellen	51
B4.1	Zeichengenerator	53
B4.2	Arbeiten mit neuem Zeichensatz	57
B5.0	Zeichen noch bunter	58
B5.1	Mehrfarbmodus	58
B5.2	Erweiterte Hintergrundfarben	60
B6.0	SPRITE - Kleingrafiken	62
B6.1	Definition eines Sprites	62
B6.2	Wohin Sprite-Daten legen?	63
B6.3	Wie sehen die Sprite-Daten aus?	64
B6.4	Wo liegen die Daten für welchen Sprite?	66
B6.5	Einschalter der Sprites	67
B6.6	Farbe für jeden Sprite	68
B6.7	Wo stehen die Sprites	69
B6.8	Dehnen in X- und Y-Richtung	71
B6.9	Sprite - Sprite Kollision	74
B6.10	Sprite - Hintergrund Kollisi	75
B6.11	Prioritäten	77
B7.0	Multi-Color-Sprite	77
B8.0	Hochauflösende Grafik	80
B8.1	Vorbereitungen für Hires	81
B8.2	Verschieben der Bit-Map	83
B9.0	Punkt setzen	85
B10.0	Li ie zeichnen	89
B11.0	Die Ellipse	94

B12.0 Zeichen in hochauflösender Grafik ... 97

B13.0 Das Grafikpaket ... 102

B14.0 Punkt löschen ... 106

(C) Musik

C1.0 Drei Stimmen und vieles mehr ... 107

C2.0 Die Hüllkurve ... 107
C2.1 Attakt - das Anschwellen ... 108
C2.2 Decay - die Lautstärkespitze ... 108
C2.3 Sustain - die Lautstärke ... 108
C2.4 Release - Ausklingen ... 108
C2.5 Die ADSR-Kurve ... 109
C2.6 Programmieren der ADSR-Kurve ... 109
C2.7 Werte sind Zeitspannen ... 110
C2.8 Volumen und Sustain ... 111

C3.0 Die Frequenz ... 112

C4.0 Die Wellenform ... 113
C4.1 Pulseweite ... 115
C4.2 Gate-Bit ... 115

C5.0 Der erste Ton ... 116
C5.1 Probieren geht über studieren ... 117

C6.0 Mehrstimmigkeit ... 118
C6.1 Der Akkord ... 118
C6.2 Prinzip der Polyphonie ... 119
C6.3 Verschlüsselung von Noten ... 124
C6.4 Bessere Verschlüsselung ... 126
C6.5 Codieren und decodieren ... 131

C7.0 Verfremdung von Tönen ... 133
C7.1 Hochpass-Filter ... 135
C7.2 Tiefpass-Filter ... 135

C7.3	Bandpass-Filter	136
C7.4	Kombination der Filtertypen	136
C7.5	Ringmodulation	138
C7.6	Synchronisation	139
C8.0	Besonderheiten der Stimme 3	140
C8.1	„Ausschalten" der Stimme 3	141
C9.0	Musik - ein Buch mit sieben Siegeln ?	142

(D) Diskette

D1.0	Was ist eine Disk ?	143
D1.1	Weg vom Computer zur Floppy	144
D2.0	Formatieren	145
D2.1	Fehlerkanal	146
D3.0	SAVE	147
D3.1	LOAD	147
D3.2	VERIFY	147
D3.3	SAVE '@:...' - überschreiben	148
D4.0	Maschinenprogramme laden	148
D5.0	Laden der Directory	149
D5.1	Der Joker	149
D6.0	Scratch - löschen	150
D6.1	Rename - umbenennen	151
D6.2	Validate - aufräumen	151
D6.3	Copy - kopieren auf der Disk	152
D6.4	Initialize - initialisieren	152
D7.0	Sequentielle Datei	152
D7.1	Eröffnen eine sequentiellen Datei	153
D7.2	Beschreiben einer Datei	154

D7.3 Lesen aus einer Datei 155
D7.4 Anhängen von Daten 156
D7.5 Statusvariable (ST) 156
D7.6 Verändern von Dateien 157

D8.0 Datensätze 158

D9.0 Telefon-Datei 161

D10.0 Datensätze über 88 Zeichen 164

D11.0 USER - File 164

D12.0 CLOSE 165

D13.0 Zusammenfassung sequentielle Datei 166

D14.0 Disketten Monitor 167
D14.1 R - Lesen eines Blockes 170
D14.2 L - Listen des Blockes 170
D14.3 A - Ändern von Daten 171
D14.4 W - Zurückschreiben 171
D14.5 X - Programmende 171

D15.0 Directory 172
D15.1 Die BAM 173
D15.2 Ändern des Diskettennamens 175

D16.0 Relative Datei 176
D16.1 Aufbau 177
D16.2 Vorbereiten einef relativen Datei 178
D16.3 Eintrag in der Directory 180
D16.4 Die Side-Sektor-Blöcke 181

D17.0 Beschreiben eines Records 183
D17.1 Lesen eines Records 184
D17.2 Wie sieht ein beschriebener Record aus? 184

D18.0 Record zerlegen in Datenfelder 185
D18.1 Wieviele Datenfelder? 185

D18.2 Datenfelder - Datensatz 186
D18.3 Datensatz - Datenfelder 189

D19.0 Index-Dateien 189
D19.1 Index-Datei mit zwei Feldern 190
D19.2 Index-Datei mit einem Feld 191
D19.3 Index-Nummer gleich Recordnummer 191

D20.0 Adress-Datei 192

D21.0 Sortieren und Suchen in einer sortierten Datei 202

D22.0 Direkter Zugriff auf die Diskette 207
D22.1 Datenspeicher 207
D22.2 Block in Datenspeicher laden 208
D22.3 Positionieren im Datenspeicher 209
D22.4 Datenspeicher zurückschreiben 210
D22.5 Block frei oder belegt kennzeichnen 214
D22.6 Einlesen der Directory in ein Programm 215

D23.0 Memory- und User-Befehle 216
D23.1 Ändern der Gerätenummer 217

(E) Kassette

E1.0 Ein billiger Massenspeicher 219

E2.0 Bits werden zur Frequenz 219

E3.0 Laden und Speichern 220
E3.1 Der Programmkopf 221

E4.0 Motorsteuerung 223

E5.0 Sequentielle Datei 226
E5.1 Eröffnen einer Datei 227
E5.2 Lesen einer Datei 228

(F) Ein- und Ausgabe

F1.0	Die Tastatur	229
F1.1	Matrix	229
F1.2	Tastaturabfrage	230
F1.3	RESTORE und SHIFT/LOCK	230
F1.4	Tastaturpuffer	230
F2.0	Joystick	231
F2.1	Aktivieren der Control-Ports	231
F2.2	JOY1 und JOY2	232
F2.3	Werte der Register 56320 und 56321	232
F2.4	Kombinationen	233
F2.5	Joystick im BASIC-Programm	233
F3.0	Paddle	235
F3.1	Abfragen der Tasten	236
F3.2	Anwendung der AD-Wandler	237
F4.0	Lightpen	237
F4.1	Funktion	238
F4.2	Justieren	238
F4.3	Ein Zeichen auslesen	238
F4.4	Übergabe von Lightpendaten	239
F4.5	Lightpen in der hochauflösenden Grafik	240
F5.0	Serieller Bus	240
F5.1	Gerätenummer	241
F5.2	Der Drucker	241
F5.3	Centronics-Schnittstelle	242
F6.0	USER-Port	243
F6.1	User-Port und CIA	244
F6.2	Datenrichtungsregister	245
F6.3	Datenregister	246
F6.4	Lauflicht	246

(G) Anhang

G1.0	Dezimalsystem	251
G1.1	Binärsystem	252
G1.2	Logische Arithmetik	254
G1.3	Zerlegen in High- und Lowbyte	257
G1.4	Hexadezimalsystem	258
G2.0	Sprite-Definitionsblatt	260
G2.1	Bildschirm- und Farbspeicher	261
G2.2	ASCII- und Bildschirmcode	262
G2.3	Notenwerte	264
G2.4	Diskettenaufbau	267
G3.0	Tips und Tricks	270
G4.0	Register-Tabellen	275

SPEICHERAUFBAU

A1.0 Mehr als nur ein „schwarzer Kasten"

Durch die technische Revolution ist es heute möglich, auf kleinstem Raum einen leistungsfähigen Computer zu bauen. Auf einer Platine findet ein kompletter Computer Platz, der vor Jahren noch einen kleinen Schrank füllte. Auch der Preis für einen Computer ist derart gefallen, daß sich viele entschlossen haben, einen Homecomputer anzuschaffen. Der COMMODORE 64 (C=64), einer der leistungsfähigsten Home-Computer, eröffnet dem Freizeit- und Hobbyprogrammierer eine Welt schier unendlicher Vielfalt. Doch leider erlaubt das sehr sparsame BASIC (V2) von COMMODORE keine komfortable und einfache Bedienung seiner wesentlichen Stärken. Hierzu zählen vor allem:

a) Hochauflösende Grafik (HIRES)
b) Sprite
c) Musik

Schon um einen Ton aus dem Lautsprecher erklingen zu lassen, ist es notwendig, in irgendwelchen ominösen Speicherstellen irgendwas hineinzulegen (POKEn). Warum und weshalb gerade hier und nicht irgendwo anders, blieb vielen bis heute verborgen. Erraten werden konnte nur, daß bestimmte Speicherbereiche für bestimmte Funktionen zuständig sind. Für die Programmierung von Grafik, Musik etc. ist es aber nötig, genau zu wissen, wo was liegt.

A2.0 64 kByte RAM (Random Access Memory)

Das Herz des C=64 bildet der Mikroprozessor 6510. Er ist Nachfolger des bekannten 6502 Mikroprozessors und besitzt einen 8-Bit Datenbus und einen 16-Bit Adressbus. Ein „bus" ist der Weg für die elektronischen Signale, um von einem Baustein zu einem anderen zu gelangen. Über den Datenbus werden Informationen, die eine Größe von einem Byte (8 Bits) besitzen, übertragen. Der Adressbus teilt dem Speicher mit, welche Speicherstelle als nächstes beschrieben oder gelesen wird. Durch die 16-Bit-Struktur lassen sich 65536 Speicherstellen (2^16 = 65536 = 64 kByte) ansprechen. Aus diesem Grunde beträgt die maximale Speicherkapazität 64 kByte (mehr Speicherstellen können nicht angesprochen werden).
Der gesamte adressierbare Speicherbereich des C=64 ist mit einem RAM (Random Access Memory) belegt. Ein RAM ist ein Speicher für den beliebigen Zugriff. Er kann gelesen und beschrieben werden (PEEK und POKE). Das bedeutet, daß von Adresse 0 bis 65535 je ein RAM-Speicherplatz zur Verfügung steht.

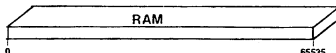


Abb. 1 64 kByte RAM

Der Schreib/Lese-Speicher (RAM) ist in acht Chips untergebracht. Jeder Chip nimmt 64 kBit auf (Typ 4164). Der Inhalt von RAM-Bausteinen kann gelöscht werden. Wird der Computer ausgeschaltet, gehen die Daten des gesamten RAM-Bereichs verloren.

A2.1 ROM (Read Only Memory)

Nach dem Einschalten befindet sich im gesamten RAM-Bereich „nichts". Sie könnten mit dem Computer in diesem Zustand nichts anfangen, denn irgendetwas muß vorhanden sein, was eine Kommunikation mit dem Computer erlaubt. Und weiter, es muß über den 16-Bit-Adressbus erreichbar sein. Doch

wie kann der Computer das machen, wenn schon alle 65536 Adressen mit einem RAM-Speicherplatz belegt worden sind? Es wird einfach „obendrauf" gelegt. Einer dieser „obendrauf" gelegten Koordinatoren ist das Betriebssystem, auch KERNAL genannt.

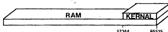


Abb. 2 64 kByte RAM + Kernal

Das Betriebssystem belegt den Adressbereich 57344 - 65535 und liegt im „1.Stock" über dem RAM. Das Betriebssystem ist der wichtigste Koordinator im C=64. Ohne ihn läuft nichts. Das Betriebssystem ist ein (Maschinen-) Programm, das immer zur Verfügung stehen muß. Aus diesem Grund kann sich dieses Programm nicht in einem RAM-Baustein befinden. Nach einem Aus- und Einschalten des Rechners wäre es auf Nimmerwiedersehen verschwunden. Ein spezieller Baustein kann Daten für immer speichern, unabhängig ob der Computer an- oder ausgeschaltet ist. Er nennt sich ROM (Read Only Memory). Wie der Name schon besagt, kann aus ihm nur gelesen werden (PEEKen). Um mit dem Computer in Verbindung zu treten, benötigt er noch einen Ein- und Ausgabebaustein. Auch er befindet sich im „1.Stock" über den Adressen 53248 -57343.

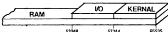


Abb. 3 64 kByte RAM + Kernal + I/O

Er liegt also direkt vor dem Betriebssystem und ist ebenfalls in einem ROM untergebracht. Der Ein- und Ausgabebaustein (input/output=I/O) ist zuständig für die Kommunikation zwischen der Außenwelt und dem Computer, sei es nun die Tastatur, die Joysticks, der Bildschirm oder aber die Diskettenstation oder Datasette.

Damit der C=64 auch unsere Sprache (BASIC) versteht, befindet sich im Bereich 40960 bis 49151 ein „Übersetzer". Dieses BASIC-Interpreter-ROM liegt natürlich auch in der ersten Etage.

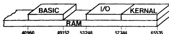


Abb. 4 64 kByte RAM + Kernal + I/O + BASIC

Als letzten wichtigen Baustein benötigt der Computer noch ein Zeichen-ROM (Character-ROM). Werden Buchstaben, Zeichen und Zahlen auf dem Bildschirm dargestellt, so haben sie eine bestimmte Form, die durch gesetzte oder nicht gesetzte Punkte in einer 8*8-Matrix sichtbar gemacht werden. Die Form der Zeichen ist im Zeichen-ROM festgelegt. Eine Besonderheit ist, daß das Zeichen-ROM in der „2. Etage" über dem Bereich 53248 bis 57343 liegt. Im „1. Stock" liegt ja der Ein- und Ausgabebaustein!

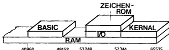


Abb. 5 64 kByte RAM + Kernal + I/O + BASIC + Zeichen

Wie verwaltet nun der Rechner den Speicherbereich, wo doch scheinbar einige Speicheradressen mehrfach benutzt werden? Er schaltet immer von einer Etage in die andere. Und zwar immer dann, wenn er von einem bestimmten Bereich etwas braucht. So z.B. zwischen dem Zeichen-ROM und dem I/O-Baustein. Den gesamten RAM-Bereich unter dem Kernal-, I/O- und BASIC-ROM betrachtet er als nicht vorhanden, obwohl er aber tatsächlich vorhanden ist.

Der Bereich 49152 - 53247 ist nicht überlagert und ist 4 kByte reiner RAM-Speicher. Doch kann dieser Speicherbereich nicht für BASIC-Programme genutzt werden, da der BASIC-Interpreter nur den Bereich von Adresse 0 bis

40959 verwalten kann. Es stehen also zunächst 40960 Speicherstellen a 1 Byte für BASIC-Programme zur Verfügung. Doch von den 40 kByte geht noch mehr verloren.

A3.0 Die ersten 2 kByte

Auch das Betriebssystem benötigt irgendwo Platz, um Zwischenergebnisse oder sich ständig ändernde Daten abzulegen. Benutzt wird hierfür der Bereich von Adresse 0 bis 1023. Besonders wichtig sind die ersten 256 Bytes (0-255) in diesem Bereich. Man spricht von der sogeannten ZERO-PAGE. Eine Page (Seite) ist ein Bereich von einem Viertel-kByte, also 256 Bytes. Teilt man den gesamten Speicherbereich (65536) in Seiten a 256 Bytes auf, so erhält man 256 Seiten. Die Zero-Page, die nullte Seite, ist die erste im Speicherbereich.

A3.1 Die Zero-Page

In ihr befinden sich alle wichtigen Informationen, die das Betriebssystem benötigt. Es empfiehlt sich nicht, wahllos hinein zu POKEn, denn es werden wichtige Daten verändert und der Computer wird wahrscheinlich „abstürzen". Viele „Zeiger" sind zu finden („Zeiger" sind Werte, die auf bestimmte Adressen weisen). Sie zeigen u.a. auf die Adressen, wo die BASIC-Programme anfangen bzw. wo sie aufhören (bitte entnehmen Sie den genauen Inhalt der einzelnen Adressen aus dem Anhang). Die nächste Page (1.Seite = 256 bis 511) dient vor allem dem Prozessor-Stack. Die Adressen 512 bis 1023 werden wieder vom Betriebssystem genutzt. In diesem Bereich befindet sich von Adresse 828 bis 1019 der Bandpuffer. Ein Zwischenspeicher für Daten, die zwischen dem Computer und der Datasette ausgetauscht werden.
Ab der Adresse 1024 folgt der Bildschirmspeicher. Alle Zeichen, die auf dem Bildschirm sichtbar sind, liegen in diesem Speicher. Die Größe ist bedingt durch die maximale Anzahl der darzustellenden Zeichen auf dem Bildschirm.

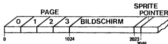


Abb. 6 Adressbereich 0 - 2048

In 25 Zeilen und 40 Spalten sind 1000 Zeichen möglich. Der Bildschirmspeicher hat eine Größe von 1000 Byte und belegt den Bereich bis 2023. Abschließend liegen von 2040 bis 2047 noch die Spritezeiger.

A4.0 Der BASIC - RAM

Der gesamte RAM-Bereich von 2048 bis 40959 (38 kByte) steht für Programme frei zur Verfügung. Hier wird das Programm mit seinen zugehörigen Variablen untergebracht. Wie, das sollten Sie sich genauer ansehen. Damit der Inhalt einzelner Adressen besser betrachtet werden kann, ist nachfolgend ein kleiner Monitor aufgelistet. Nach Eingabe der Startadresse zeigt er Ihnen den Inhalt der Adresse in Dezimal-, Binär- und Hexadezimalwerten, sowie das entsprechende ASCII-Zeichen an (eine Hilfe, um Zeichen im Speicher zu finden).

```
100 rem****************** mini - monitor ****************
110 :
120 rem************** variablen bestimmung **************
130 bl$="       ":h$="0123456789abcdef"
140 t$=" adresse  dezi  hexa   binaer   ascii   "
150 rem**********************************************
160 :
170 rem***************** hauptprogramm ******************
180 poke 53280,0
190 print chr$(147)
200 print " --- mini monitor ---"
210 print
220 input " * adresse (start) :";v
230 if v<0 then print chr$(147):end
240 if v> 65535 then print chr$(145);: goto 220
250 print
260 print t$:x=0
270 rem***************** einlese-schleife ****************
280 for z = v to 65535:x=x+1
290 w=peek(z): l1=len(str$(w)):l2=len(str$(z))
300 print left$(bl$,7-l2);z;
310 print left$(bl$,5-l1); ;"   ";
320 gosub 470: print he$;"   ";
330 gosub 520: print bi$;"   ";
```

```
340 if w <32 or (w>127 and w<160) then print chr$(46):
    goto 360
350 print chr$(w)
360 if x<16 then 410
370 print " *weiter (space)* *ende (return)"*
380 get a$: if a$ = "" then 380
390 if a$ = chr$(13) then 190
400 x=0:print chr$(147):print:print:print t$
410 next
420 print " *ende*"
430 get a$: if a$= "" then 430
440 goto 190
450 rem*************************************************
460 :
470 rem************* dezimal - hexa ********************
480 he$="":d=int(w/16):he$=mid$(h$,d+1,1):d=w-d*16
490 he$=he$+mid$(h$,d+1,1):return
500 rem*************************************************
510 :
520 rem************* dezimal - binaer ******************
530 bi$="":d=w
540 d=d/2:d$="0":if d<>int(d) then d$="1"
550 d=int(d):bi$=d$+bi$:if d>0 then 540
560 if len(bi$)<8 then bi$="0"+bi$: goto 560
570 return
580 rem*************************************************
```

Nachdem Sie den Mini-Monitor eingegeben oder geladen haben, liegt er natürlich als Programm im BASIC-Speicherbereich. Mit dem Monitorprogramm können Sie nun den Monitor im BASIC-RAM untersuchen (die Beschreibung bezieht sich genau auf das obige LISTing). Geben Sie als Startadresse 2049 ein (BASIC-Start). Auf dem Bildschirm sehen Sie die Adresse, den Wert in Dezimal, Hexadezimal und Binär, sowie das entsprechende ASCII-Zeichen.

```
2049   66   42   0100 0010   B
2050    8   08   0000 1000   .
2051  100   64   0110 0100   -
2052    0   00   0000 0000   .
2053  143   8F   1000 1111   .
2054   42   2A   0010 1010   *
2055   42   2A   0010 1010   *
```

Sehen Sie sich dazu einmal die Zeile 100 im Programm an. Es ist eine Remark-Zeile mit vielen Sternchen (*). Ab der Adresse 2054 sind die Sterne auch zu finden. Die Zeilennummer (100) ist in den Adressen 2051 und 2052 zu finden. Sie liegen hier zerlegt als Lowbyte und Highbyte (0*256+100=100). Aus dem Listing ist zu entnehmen, daß hinter der Zeilennummer der Befehl REM steht. Doch hier finden Sie zwischen der Zeilennummer und dem Text (*) nur den Wert 143! 143 ist die Kennzahl für den Befehl REM. Jeder BASIC-Befehl besitzt eine solche „Kennzahl", die TOKEN genannt wird. Ein großer Vorteil ist dabei, daß jeder BASIC-Befehl jeweils nur 1 Byte belegt.

```
128  END       129  FOR       130  NEXT
131  DATA      132  INPUTØ    133  INPUT
134  DIM       135  READ      136  LET
137  GOTO      138  RUN       139  IF
140  RESTROE   141  GOSUB     142  RETURN
143  REM       144  STOP      145  ON
146  WAIT      147  LOAD      148  SAVE
149  VERIFY    150  DEF       151  POKE
152  PRINTØ    153  PRINT     154  CONT
155  LIST      156  CLR       157  CMD
158  SYS       159  OPEN      160  CLOSE
161  GET       162  NEW       163  TAB
164  TO        165  FN        166  SPC(
167  THEN      168  NOT       169  STEP
170  +         171  -         172  *
173  /         174  ^         175  AND
176  OR        177  größer    178  =
179  kleiner   180  SGN       181  INT
182  ABS       183  USR       184  FRE
185  POS       186  SQR       187  RND
188  LOG       189  EXP       190  COS
191  SIN       192  TAN       193  ATN
194  PEEK      195  LEN       196  STR$
197  VAL       198  ASC       199  CHR$
200  LEFT$     201  RIGHT$    202  MID$
203  GO
255  PI
```

Aus dieser Liste können Sie die Beziehung zwischen Token und Befehl entnehmen. 143 entspricht dem Befehl REM. Außer den Befehlen ist auch allen Funktionen und Operatoren ein Token zugeordnet.

Vor der Zeilennummer (2051/2052) steht die Adresse, wo die nächste Zeile des Programms im Speicher beginnt. Die Adresse ist wieder in Low- und Highbyte angegeben (2049 = Lowbyte, 2050 = Highbyte). Rechnen Sie die Adresse aus, so erhalten Sie die Adresse 2114 (8*256+66=2114).
Blättern Sie mit dem Monitor weiter (SPACE), ist der weitere Text der REM-Zeile zu sehen, bis in der Adresse 2113 auf eine „Null" gestoßen wird. Die Null markiert das Ende dieser Zeile und eine neue Zeile beginnt mit dem folgendem Byte. Der Aufbau der folgenden Zeilen ist identisch:

72 Lowbyte Adresse nächste Zeile
8 Highbyte Adresse nächste Zeile
110 Lowbyte Zeilennummer
0 Highbyte Zeilennummer

Es folgen dann die Befehle in Token und die Zeichen in ASCII, am Ende der Zeile dann das Nullbyte als Trennzeichen. In der Zeile 110 ist nur das Trennzeichen vorhanden (:). Das gesamte Programm ist so aufgebaut und kann mit Hilfe des Monitors verfolgt werden.

Schauen Sie sich abschließend noch das Programmende ab Adresse 3580 an. Die letzten Sterne (*) der Remarkzeile 580 sind zu sehen und hinter dem Trennbyte (0) folgen zwei weitere Nullbytes, die das Ende kennzeichnen. Immer wenn die nächste Adresse 0 ist, weiß der Rechner, daß hier das Programm endet. Wie Sie sehen, folgen zwar noch Daten, aber die gehören nicht zum Programm, sondern es handelt sich um Variablen.

A4.1 Speichern von Variablen

Wagen Sie nun einen Blick in die Zero-Page ab der Adresse 43. Irgendwo muß für das Betriebssystem stehen, wo ein BASIC-Programm beginnt und wo es endet. In 43/44 ist der Programm-Anfang und in 45/46 die Endadresse des Programms zu finden.

43 (1) Lowbyte Programm-Anfang
44 (8) Highbyte Programm-Anfang
45 (x) Lowbyte Programm-Ende
46 (x) Highbyte Programm-Ende

Der Programm-Anfang liegt normal immer bei 2049 (8*256+1). In den Adressen 45/46 wird auf das Programm-Ende gezeigt (in diesem Fall 14*256+5=3589!) und gleichzeitig zeigt die Adresse 45/46 auf den Beginn der Variablen. Es ist ja klar, daß sich der Computer Inhalte von Variablen irgendwo merken muß. Sehen Sie sich das Ende des BASIC-Programmms genauer an.

Ab der Programm-Endadresse, die aus den Adressen 45/46 zu entnehmen ist, liegen die z.Z. vom Computer benutzten Variablen. Pro Variable werden zunächst 7 Bytes belegt. Damit der Rechner unterscheiden kann, um welchen Variablentyp es sich handelt (string, real, integer), bedient er sich einer einfachen Regel.

1.) Variablennamen dürfen nur aus ASCII-Zeichen bestehen. Aus diesem Grunde wird das Bit 7 nicht benutzt.

2.) Variablennamen können nur in einer Länge von zwei Zeichen unterschieden werden.

Jeweils das Bit 7 der beiden Bytes des Variablennamens wird zur Kennzeichnung des Variablentyps benutzt.

REAL:	1.Byte	0xxx xxxx
	2.Byte	0xxx xxxx

Der Variablenname (2 Zeichen!) steht in den ersten beiden Bytes jeweils in den Bits 0 bis 6. Je nach Kombination von gesetzten oder gelöschten Bit 7 kann nach Variablentyp unterschieden werden. Sind beide Bits „0“, so handelt es sich um eine reale Variable.

STRING	1.Byte	0xxx xxxx
	2.Byte	1xxx xxxx
INTEGER	1.Byte	1xxx xxxx
	2.Byte	1xxx xxxx
FN	1.Byte	1xxx xxxx
	2.Byte	0xxx xxxx

Aus den folgenden 5 Byte ist dann entweder der Inhalt der Variablen zu entnehmen oder aber der Ort, wo der Inhalt im Speicher zu finden ist (Strings).

TYP	BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6
real	0 Name	0 Name	Expon.	-- Wert in 4 Bytes zerlegt --			
$	0 Name	1 Name	Länge	Adr.lb	Adr.hb	0	0
%	1 Name	1 Name	Highb.	Lowb.	0	0	0
FN	1 Name	0 Name	Adr.lb	Adr.hb	Adr.lb	Adr.hb	0

Bei REAL- und INTEGER-Variablen steht der Inhalt direkt hinter dem Namen. Reale Variablen benötigen 5 Byte, Integervariablen hingegen nur 2 Byte für die Darstellung. Bei Stringvariablen ist nach dem Namen die Länge der Variablen zu finden, anschließend der Ort, wo der Inhalt liegt.

Sehen Sie sich den Bereich hinter dem BASIC-Programm einmal an. Als erstes finden Sie die Stringvariable BL (das „L" entsteht aus 204-128), gefolgt von der Länge (6) und dem Ort, wo der Inhalt zu finden ist (8*256+146=2194). Die Adresse 2194 ist genau der Punkt, wo im Programm der Leerstring definiert worden ist. Bei der Variablen BL$ handelt es sich nicht um eine Variable im eigentlichen Sinn, denn der Inhalt (6 Leerzeichen) wird im gesamten Programm niemals verändert. Es handelt sich mehr um eine Konstante. Ändert sich der Inhalt einer Stringvariablen, wird der geänderte Ausdruck immer an das Ende des BASIC-RAM gelegt! Das Ende des BASIC-RAM liegt (normalerweise) bei 40959. Von hier aus werden die Inhalte der Stringvariablen abwärts (!) abgelegt. Neue oder sich ändernde Stringvariablen werden immer wieder „vornean" gelegt, bis auf den Speicherbereich getroffen wird, wo die Arrays abgelegt sind. Indizierte Variablen werden direkt hinter den normalen Variablen abgelegt, so daß der Speicheraufbau folgende Struktur besitzt:

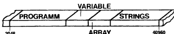


Abb. 7 BASIC-RAM

Nach dem Programm folgen die normalen Variablen. Aus den Adressen 45/46 ist die Adresse zu entnehmen. Nach den normalen Variablen liegen die Arrays. Stringvariablen werden abschließend vom BASIC-RAM-Ende abwärts bis an die Arrays abgelegt (49/50). Wo der nächste Stringinhalt abgelegt wird, steht übrigens in den Adressen 51/52. Erreichen die Stringvariablen das Ende der Arrays bzw. der normalen Variablen, wenn keine Arrays vorhanden sind, wird der gesamte String-Bereich aufgeräumt. Nur noch die aktuellen Stringinhalte bleiben erhalten und werden neu an das Ende des BASIC-RAM gelegt. Dieses „Entmüllen" (GARBAGE COLLECT) ist eine sehr umfassende und langwierige Routine, die mehrere Minuten (!) dauern kann. Allerdings kommt es sehr selten vor, daß so viele Strings vorhanden sind, daß aufgeräumt werden muß. Durch die Funktion FRE(0) wird ein Aufräumen erzwungen. Der tatsächliche freie Speicherplatz kann natürlich nur dann bestimmt werden, wenn alle „alten", nicht mehr benötigten Stringinhalte gelöscht sind. Geben Sie einmal das kleine Programm ein und fragen Sie dann mit PRINT FRE(0) nach dem freien Speicherplatz:

```
10 dim a$(200):print chr$(147)
20 for i= 1 to 200
30 for j=1 to 10
40 x=int(rnd(0)*26)+65
50 a$(i)=a$(i)+chr$(x)
60 next j
70 print chr$(19);a$(i),i
80 next i
```

Wie sie sehen, dauert es schon bei nur 200 Stringvariablen einige Sekunden, bis die „free-Meldung" erscheint.

A4.2 Speichern von Arrays

Eine Besonderheit bildet die Ablage von indizierten Variablen. Sie liegen zwischen den normalen Variablen und den Stringinhalten.
Ein Array kann in mehren Dimensionen definiert werden. Man spricht entsprechend von 1-, 2-, bzw. n-dimensionalen Feldern.

A(100)	1-dimensional
A(20,60)	2-dimensional
A(2,4,5,6,7)	5-dimensional

Im Gegensatz zu normalen Variablen, die pro Variable 7 Bytes belegen (ggf. mit Nullen aufgefüllt), wird bei den Arrays nur der tatsächliche Speicherbedarf benutzt. Reale Variablen werden in 5 Byte, Ort und Länge der Stringvariablen in 3 Byte und Integervariablen in 2 Byte untergebracht. Der für ein Array benötigte Speicherplatz errechnet sich durch multiplizieren der Feldparameter + 1 (Feld 0 !!) und der Länge des Eintrags (real=5, string=3, integer=2). Das Array A(20,5,7) belegt:

(20+1)*(5+1)*(7*1)*5 = 5040 Byte !

Es ist mit 5 multipliziert worden, weil A(...) eine reale Variable ist.
Bevor die eigentlichen Variablen abgelegt sind, steht in einem Arraykopf, um welches Array es sich handelt, wie groß es ist und wie viele Dimensionen es besitzt.

Byte 0	Name des Array und Type (Bit7 !)
Byte 1	Name des Array und Type (Bit7 !)
Byte 2	Feldlänge (LB)
Byte 3	Feldlänge (HB)
Byte 4	Anzahl der Dimensionen
Byte 5	Größe der Dimension (LB)
Byte 6	Größe der Dimension (HB)

Je nach Anzahl der Dimensionen wird die Größe der Dimensionen ab Byte 5 abgelegt. Bei einem 2-dimensionalen Feld folgt in Byte 7 und Byte 8 die Größe der nächsten Dimension usw. (je nach Anzahl der Dimensionen). Nun endlich folgen die Variableninhalte bzw. der Ort, wo sie liegen (strings).
Ein konkretes Zahlenbeispiel:

A$(2,3,4)

Byte 0	65	Name/Typ 0 1000001
Byte 1	128	Name/Typ 1 0000000 (hier nur Typ)
Byte 2	155	Feldlänge lb
Byte 3	0	Feldlänge hb zur Feldlänge (2+1)*(3+1)*(4+1) zählen noch die belegten Bytes des Arraykopfes zu.
Byte 4	3	Anzahl der Dimensionen
Byte 5	5	Dimensiongröße lb (Feld 3)
Byte 6	0	Dimensiongröße hb (Feld 3)

Byte 7	4	Dimensionsgröße lb (Feld 2)
Byte 8	0	Dimensionsgröße hb (Feld 2)
Byte 9	3	Dimensionsgröße lb (Feld 1)
Byte 10	0	Dimensionsgröße hb (Feld 1)
Byte 11		Stringlänge A$(0,0,0)
Byte 12		Adresse lb von A$(0,0,0)
Byte 13		Adresse hb von A$(0,0,0)
Byte 14		Stringlänge A$(1,0,0)
.....		....

Alle anderen Stringvariablen folgen ab Byte 15 im selben Aufbau (Länge, LB/HB).

A4.3 Löschen eines BASIC-Programms

Durch Eingabe des Befehls NEW wird scheinbar das gespeicherte Programm gelöscht. Tatsächlich befindet sich aber das Programm (fast) unverändert im Speicher. Gelöscht werden nur die „Zeiger", die auf das Programm zeigen.

1.) Das BASIC-Ende (45/46) wird auf Basic-Anfang gelegt.

2.) Die ersten beiden Bytes nach dem Basic-Anfang werden auf Null gesetzt (wenn auf zwei Null-Bytes gestoßen wird, ist das Programm beendet).

3.) Die Variablenzeiger werden wieder in den ursprünglichen Zustand gesetzt.

Ist versehentlich ein NEW eingegeben worden, läßt sich das Programm retten, indem die Zeiger wieder auf das Programm gestellt werden.

A4.4 MERGE - Aneinanderhängen von Programmen

Der Aufbau einer Programmbibliothek ist immer dann sehr sinnvoll, wenn Programme oder Programmteile oft verwendet werden (Grafik, Bilder etc). Leider ist das aneinanderhängen von Programmen ohne weiteres nicht möglich. Doch überlegen Sie einmal, wie normalerweise BASIC-Programme im

Speicher abgelegt werden. Ein BASIC-Programm wird immer ab der Adresse abgelegt, die in 43/44 (BASIC-Anfang) bestimmt ist. Befindet sich ein Programm im Computer und es wird ein zweites geladen, so wird dieses wieder ab der in 43/44 bezeichneten Adresse abgelegt. Hier liegt aber schon das erste Programm und es wird ganz einfach überschrieben. Ein weiteres Programm muß also an das Ende des ersten Programms gelegt werden. Um das zu erreichen, wird der BASIC-Anfang (43/44) auf BASIC-Ende vom ersten Programm gelegt (45/46).

```
Z=PEEK(45)+PEEK(46)*256     BASIC-Ende!
Z=Z-2
```

Von der Endadresse des BASIC-Programms werden 2 abgezogen (die beiden Nullbyte für Programmende!!). Jetzt werden die Adressen 43/44 (BASIC-Anfang) mit dem Wert Z (HB/LB) geladen.

```
POKE 43, Z AND 255: POKE 44, Z/256
```

Das Programm, das nun geladen wird, wird direkt hinter das erste Programm gelegt.

```
LOAD "PROG.2",8 (1)
```

Abschließend ist der BASIC-Anfang wieder auf den ursprünglichen Wert zu setzen.

```
POKE 43,1: POKE 44,8        Adresse 2049
```

Beachtet werden muß noch folgendes: die Zeilennummern des nachgeladenen Programms dürfen nicht kleiner oder gleich der sich im vorhandenen Programm befindlichen Zeilennummern sei !

A4.5 Schützen eines Speicherbereichs

Der BASIC-Speicherbereich liegt normal von Adresse 2049 bis 40959. Damit innerhalb dieses Bereiches ein geschützter Raum entsteht (z.B. für Maschinenprogramme, Bildschirme) muß der BASIC-RAM, also der für Programme und Variablen reservierte Bereich, beschnitten werden. Die gängigste Methode ist

das Verlegen des BASIC-RAM-Endes. Der Zeiger in der Adresse 55/56 bestimmt das BASIC-RAM-Ende. Es zeigt auf 40960 (Highbyte = 160). Um das BASIC-RAM-Ende weiter nach vorne zu legen, z.B. nach 32768, wird der Zeiger entsprechend geladen. Highbyte (32768/256) gleich 128, Lowbyte gleich 0.

```
POKE 55, 0: POKE 56, 128
```

Bedenken Sie bitte, daß die Strings immer an das Ende des BASIC-RAM gelegt werden. Doch die Strings wissen nicht, daß der RAM verkleinert worden ist. Der Zeiger 51/52 bestimmt den Ort, ab wo die Strings abgelegt werden dürfen. Auch dieser muß auf die Adresse 32768 zeigen (normal 40960). Er muß ebenfalls mit

```
POKE 51, 0: POKE 52, 128
```

geladen werden. Jetzt ist der Bereich 32768 bis 40960 geschützt und kann ohne Bedenken genutzt werden. Eine FREE-Meldung gibt auch nur noch 30720 Bytes aus.

A5.0 BASIC- und KERNAL-ROM

Das BASIC-ROM überlagert den RAM-Adressbereich 40960 bis 49151 (siehe A2.1). Aber woher weiß der Rechner, daß er in das überlagerte ROM und nicht in den RAM greifen soll? Probieren Sie einmal folgendes aus:

```
POKE 41000,0: PRINT PEEK(41000)
```

Obwohl in die Adresse 41000 der Wert „0“ gelegt worden ist, erscheint beim Auslesen der Wert 209. Was ist geschehen? Der eingegebene Wert „0“ liegt tatsächlich in Adresse 41000, aber im RAM!! Sowie versucht wird, die Adresse auszulesen, lesen Sie das darüber liegenden ROM (BASIC-ROM) aus!

POKEn ins RAM! PEEKen aus dem ROM!

Genauso verhält es sich im Bereich des Betriebssystems (Kernal).

```
POKE 61000,0: PRINT PEEK(61000)
```

Sie erhalten den Wert 160. Auch hier wird in den RAM geschrieben und aus dem ROM gelesen. Aus diesem Grund ist es sehr einfach, den BASIC-Interpreter in einen anderen Bereich (RAM) zu kopieren.

```
10 for s=40960 to 49151
20 poke s,peek(s)
30 next s
```

Was etwas komisch aussieht (poke s,peek (s)), ist doch ganz logisch. Lesen (PEEK) aus dem ROM und schreiben (POKE) ins RAM. Nun befindet sich der BASIC-Interpreter als Kopie auch im RAM-Bereich 40960 bis 49151. Weiterhin arbeitet das BASIC aber im ROM.

Das Datenregister des Prozessors (Adresse 1) legt u.a fest, ob aus dem ROM- oder RAM-Bereich gelesen werden soll. Zuständig für diese Organisation ist Bit 0 und Bit 1. Normal befindet sich in Adresse 1 der Wert

```
55   = 0011 0111
```

Je nach Stellung der ersten beiden Bits entstehen die Kombinati

```
xxxx xx11 = „BASIC-ROM" / „KERNAL-ROM"
xxxx xx10 = „BASIC-RAM" / „KERNAL-ROM"
xxxx xx01 = „BASIC-RAM" / „KERNAL-RAM" (mit I/O)
xxxx xx00 = es sind 64 kByte RAM frei !!
```

Mit der Kombination xxxx xx00 ist der gesamte Speicher ein RAM, nur leider ist damit nichts anzufangen.

Nach dem Kopieren des BASIC-Interpreters wird dem Rechner durch Löschen des Bits 0 mitgeteilt, daß er sein BASIC aus dem RAM-Bereich holen soll.

```
50 poke1,peek(1) and 254
```

Damit Sie sich überzeugen können, daß das BASIC nun aus dem RAM geholt wird, ändern Sie das BASIC ein bißchen. Aus der READY-Meldung wird nun eine FERTIG-Meldung.

```
60 poke 41848, 70:rem f
65 poke 41849, 69:rem e
```

```
70 poke 41850, 82:rem r
75 poke 41851, 84:rem t
85 poke 41852, 73:rem i
90 poke 41853, 71:rem g
99 stop
```

Auch das Betriebssystem soll ins darunterliegende RAM kopiert werden.

```
100 for s = 57344 to 65535
110 poke s, peek(s)
120 next s
```

Damit das Betriebssystem ebenfalls aus dem RAM geholt wird, muß dies i Adresse 1 mitgeteilt werden. Bit 0 muß gesetzt und Bit 1 gelöscht werden.*

```
130 poke 1, peek (1) and 252 or 1
```

Geben Sie einmal POKE 59574,20 ein und LISTen Sie das Programm. Der gesamte Bildschirmaufbau ist zwar zerstört, aber Sie haben festgestellt, daß das Betriebssystem aus dem kopierten Bereich geholt wird. Normalisieren Sie das Betriebssystem wieder mit POKE 59574,39.

Damit das BASIC- und das Betriebssystem wieder aus dem ROM gelesen wird, brauchen Sie nur das Bit 0 und Bit 1 in Adresse 1 zu setzen (POKE 1,55) oder RUN/STOP+RESTORE auslösen.

A6.0 Ein-/Ausgabe-Bausteine und der Zeichen-ROM

Bisher sind die I/O-Bausteine (Ein-/Ausgabe=input/output) global als I/O-ROM bezeichnet worden. Allerdings ist dieser Bereich wesentlich komplexer. Einige Bausteine sind Zwittertypen zwischen RAM und ROM. So lassen sich beim SID-Chip (Sound Interface Divice) einige Register zwar lesen aber nicht beschreiben, oder sie lassen sich beschreiben aber nicht lesen. Der I/O-Bereich setzt sich aus mehreren Baugruppen zusammen.

* Erläuterungen zu den logischen Operatoren AND und OR finden Sie übrigens i

Name	Adresse:	
VIC	53248 - 54271	DØØØ - D3FF
SID	54272 - 55295	D4ØØ - D7FF
Farb-RAM	55296 - 56319	D8ØØ - DBFF
CIA 1	56320 - 56575	DCØØ - DCFF
CIA 2	56576 - 56831	DDØØ - DDFF
unbenutzt	56832 - 57344	DEØØ - EØØØ

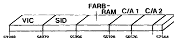


Abb. 8 I/O

Und eine weitere Besonderheit taucht auf: „über" dem I/O-Bereich liegt außerdem noch das Zeichen-ROM. Da der Rechner auf beide Einheiten Zugriff haben muß, wird zwischen dem I/O-Bereich und dem Zeichen-ROM ständig (alle 17 ms) hin und her geschaltet. Um den Zeichensatz aus dem Zeichen-ROM in einen RAM-Bereich zu kopieren, muß zum einen auf das Zeichen-ROM „gezeigt" werden und zum anderen muß das Umschalten zwischen I/O und Zeichen-ROM ausgeschaltet werden. In der schon bekannten Adresse 1 dient das Bit 2 als Zeiger auf I/O oder Zeichen-ROM.

xxxx x1xx = bedient den I/O-Bereich. Zeichen können nur vom VIC gelesen werden.

xxxx x0xx = bedient das Zeichen-ROM (kann gelesen werden).

An dieser Stelle soll nicht weiter auf das Kopieren des Zeichensatzes eingegangen werden. In einem Abschnitt zur Grafik wird dieser Vorgang ausführlich beschrieben.

A6.1 Der VIC II - Chip

Der Video-Controler (VIC) ist vor allem für die Erzeugung eines Bildes zuständig. Alles, was auf dem Bildschirm erscheint, ist vom VIC gesteuert. Sprites, Farben, Zeichen, hochauflösende Grafik, eben alles. In seiner internen Struktur besitzt er 14 Adressleitungen und kann aus diesem Grund nur auf 16 kByte des gesamten Speicherbereichs zugreifen ($2^{14}=16384$). Im Einschaltzustand werden die ersten 16 kByte des Speicherbereichs verwaltet (0-16383). Doch hier tauchen auch schon die ersten Besonderheiten auf. Wie kann der Video-Controler auf den Zeichensatz zugreifen, der außerhalb (ab 53248) der ersten 16 kByte liegt? Und weiter, der Farb-RAM (55296-56319) liegt ebenfalls außerhalb des adressierbaren 16 kByte-Bereiches! Doch zuerst zum Zeichensatz. Der Video-Controler greift immer auf den RAM-Bereich zu. Innerhalb der ersten 16 kByte befindet sich ein 4 kByte Speicherbereich (4096-8191), der mitten im BASIC-Speicher liegt. Greift der Video-Controler auf diesen Bereich zu, so wird er automatisch auf das Zeichen-ROM umgeschaltet. So kann der VIC II - Chip auch im ersten 16 kByte Bereich auf den Zeichensatz zugreifen. Im selben Prinzip arbeitet der Video-Controler im Adressbereich 36864 bis 40959. Greift der VIC II -Chip auf diesen RAM-Bereich zu, wird er wieder auf das Zeichen-ROM „umgeleitet". Der Farb-RAM liegt fest innerhalb der Adressen 55296 bis 56319. Es ist ein separater Baustein und er ist fest mit dem VIC auf diese Adressen verschaltet (nicht zu verwechseln mit dem „darunterliegenden" RAM!). Mit diesen Tricks befinden sich alle für den Video-Controler wichtigen Bereiche innerhalb der ersten 16 kByte.

Adresse 1024-2023 Bildschirm-Speicher
Adresse 2040-2047 Sprite-Zeiger
Adresse 4096-8191 scheinbar der Zeichensatz

Teilt man den 64 kByte Speicher in Bereiche a 16 kByte ein, so erhalten Sie vier 16 kByte Bereiche. Der adressierbare Bereich des Video-Controler kann auf einen der 4 Bereiche eingestellt werden. Die ersten beiden Bits der Adresse 56576 bestimmen diesen Ort.

Binär	DEZ	Adressbereich
xxxx xx11	3	0 - 16383 (Einschaltzustand)
xxxx xx10	2	16384 - 32767
xxxx xx01	1	32768 - 49151
xxxx xx00	0	49152 - 65535

Gesetzt werden die Bits mit folgender Zeile (den Wert „DEZ" entnehmen Sie der obigen Tabelle):

```
POKE 56576, PEEK(56576) AND 252 OR DEZ
```

Wird der adressierbare Bereich des Video-Controlers mit

```
POKE 56576, PEEK(56576) AND 252 OR 1
```

in den Bereich 32768 bis 49151 verschoben, muß der Bildschirmspeicher ebenfalls in diesem Adressbereich liegen. Dem Betriebssystem muß dann mitgeteilt werden, wo der „neue" Bildschirmspeicher zu finden ist.

A6.2 Verschieben des Bildschirmspeichers

Innerhalb eines 16 kByte Bereiches könnte der Bildschirmspeicher theoretisch an 16 verschiedenen Orten liegen (16* 1 kByte = 16 kByte). Mit den 4 hochwertigen Bits des Registers 53272 wird der Ort des Bildschirmspeichers festgelegt.

Binär	DEZ	Adressbereich	
0000 xxxx	0	0 - 1023	
0001 xxxx	16	1024 - 2047	Einschaltzustand
0010 xxxx	32	2048 - 3071	
0011 xxxx	48	3072 - 4095	
0100 xxxx	64	4096 - 5119	
0101 xxxx	80	5120 - 6143	
0110 xxxx	96	6144 - 7167	
0111 xxxx	112	7168 - 8191	
1000 xxxx	128	8192 - 9215	
1001 xxxx	144	9216 - 10239	
1010 xxxx	160	10240 - 11263	
1011 xxxx	176	11264 - 12287	
1100 xxxx	192	12288 - 13311	
1101 xxxx	208	13312 - 14335	
1110 xxxx	224	14336 - 15359	
1111 xxxx	240	15360 - 16383	

* Erläuterungen zu den logischen Operationen finden Sie übrigens im

Gesetzt werden die entsprechenden Bits mit:

POKE 53272, PEEK(53272) AND 15 OR DEZ

Der Dezimalwert (DEZ) ist der Tabelle zu entnehmen. Der oben angegebene Adressbereich bezieht sich immer auf die ersten 16 kByte. Ist der adressierbare Adressbereich des Video-Controlers z.B. auf den Bereich 32768-49151 verlegt worden, so liegt der Bildschirmspeicher um 32768 Byte verschoben (32768 + 1024 = 33792!!).
Abschließend wird dem Betriebssystem noch mitgeteilt, wo ab sofort der Bildschirmspeicher zu finden ist. In der Adresse 648 wird das Highbyte der Bildschirmstartadresse abgelegt. Normal finden wir hier eine 4, woraus sich ergibt, daß der Bildschirmspeicher bei 1024 (4*256) beginnt.

POKE 648, Adresse/256
POKE 648, 33792/256 (=132)

Achten Sie darauf, daß Sie den Bildschirmspeicher nicht an Orte legen, die für die Funktionsfähigkeit des Rechners notwendig sind (Zero-Page etc). Ebenso sollten Textbildschirme nicht in den Bereich 4096-8191 oder 36864-40959 gelegt werden (siehe A6.1). Der Bildschirm wird durch den Zugriff auf den Zeichensatz gestört. Und selbstverständlich auch nicht in den Bereich, wo das Programm liegt, denn dann würde es unweigerlich zerstört.

A6.3 Arbeiten mit 8 Bildschirmen

An einem kleinen Demonstrationsbeispiel soll Ihnen das Arbeiten mit mehreren Bildschirmen verdeutlicht werden. Das kurze Programm ermöglicht Eingaben auf 8 Bildschirmen. Beachten Sie bitte das Setzen des Farb-RAM in Zeile 80. Dies ist notwendig, da bei einem Löschen des Bildschirmes (CHR$(147)) auch der Farb-RAM auf die Hintergrundfarbe gesetzt wird (bei neueren Betriebssystemen ist dies nicht der Fall). Im zweitem Teil des Programms werden alle 8 Bildschirme noch einmal gezeigt.

```
10 for x=8 to 15
20 poke 53272, peek(53272) and 15 or 16*x
30 poke 648,x*1024/256
```

```
40 print chr$(147)
50 print "bildschirm ";x-7
60 print:input "text:";t$
70 next
80 for f=55296 to 56295:poke f,1:next
90 for x=8 to 15
100 poke 53272, peek(53272) and 15 or 16*x
110 poke 648,x*1024/256
120 get a$:if a$= "" then 120
130 next
140 poke 53272, peek(53272) and 15 or 16
150 poke 648,4
160 print chr$(147):end
```

A7.0 CIA - Complex Interface Adapter

Die CIA-Bausteine sind die letzte Baugruppe innerhalb des Ein-/Ausgabebereichs. Die CIA's kontrollieren vor allem das zeitliche Zusammenspiel bei der Datenein- und ausgabe. Im allgemeinen besteht der CIA aus folgenden Teilen:

a) 16 programmierbare Ein- und Ausgabeleitungen
b) Zwei 16-Bit-Intervalltimer
c) Uhr (24-Stunden)
d) 8-Bit Schieberegister (serielle Ausgabe)
e) Handshake für Ein- oder Ausgabe

Die programmierbaren Ein- und Ausgabeleitungen liegen an dem USER-Port und können Daten von außen aufnehmen oder sie vom Computer zu einem externen Gerät senden (siehe F5.0). Im Abschnitt zum USER-Port (siehe F6.0), wird dieses ausführlich behandelt.

Die Startadresse vom CIA 1 wird folgend mit CI bezeichnet, um das Arbeiten mit den Registern zu vereinfachen.

CI = 56320

Sogenannte „Handshakeleitungen" kontrollieren das zeitliche Zusammenspiel bei der Datenübermittlung. Werden z. B. Daten zu einem Drucker geschickt, so arbeitet der Drucker nur dann, wenn Daten für ihn am Ausgang zur

Verfügung stehen. Erst wenn er die Daten erhalten und verarbeitet hat, kann er neue Daten annehmen. Es wird dem Computer über die Handshakeleitungen mitgeteilt, wann neue Daten verarbeitet werden können. Damit wird sichergestellt, daß die Übertragungsgeschwindigkeit immer vom „langsamsten" Gerät gesteuert wird und somit keine Daten verloren gehen.

Ein Schieberegister ist ein Bauelement, welches Daten bitweise über eine Leitung ausgibt. So nimmt es ein Byte (8 Bit) in sich auf und gibt dieses Byte Bit für Bit über eine Leitung wieder aus. Genutzt wird es unter anderem zur Datenübertragung beim seriellen Ausgang (siehe F6.0).

Interessant sind die im CIA vorhandenen Timer (Zähler). Da sind zum einen zwei 16-Bit Timer (Timer A und Timer B). Sie zählen von einem bestimmten Wert abwärts gegen Null. Die Timer werden vom Betriebsystem mit einem Wert von 16421 geladen (HB = 64, LB = 37).

CI+4 56324 Lowbyte Timer A
CI+5 56325 Highbyte Timer A
CI+6 56326 Lowbyte Timer B
CI+7 56327 Highbyte Timer B

Die beiden Timer (A und B) dienen vor allem der Interrupt-Behandlung. Immer wenn der Timer A den Wert Null erreicht, wird das Bit 0 im Interrupt Control Register (CI + 13) gesetzt. Und wenn Bit 0 gesetzt ist, springt das Betriebssystem z.B. in eine Routine, um die Tastatur abzufragen. Ist keine Taste betätigt worden, führt es das Programm an der Stelle weiter, wo es „unterbrochen" worden ist. Ohne eine Interrupt-Behandlung könnte z.B. ein laufendes Programm nie mehr mit der RUN/STOP-Taste gestoppt werden. Der Zeitabstand zwischen den Interrupts wird von den Timern A und B festgelegt. Das Zeitintervall kann sehr einfach verändert werden. Geben Sie bitte einmal

POKE 56325, 200

ein. Sie werden sofort bemerken, daß der Cursor wesentlich langsamer blinkt. Die Interruptzeit ist verlängert worden. Durch Eingabe von

POKE 56325, 8

wird das Interruptintervall verkürzt und der Cursor blinkt sehr schnell.

Der Timer B arbeitet identisch, setzt aber beim Erreichen von Null das Bit 1 im Register CI+13 (56333) und löst damit einen Interrupt aus. Nach dem Erreichen der Null fangen beide Timer wieder bei ihrem ursprünglichen Wert an zu zählen. Die beiden Timer sind vom Speicherraum prinzipiell doppelt belegt. Der Wert, von wo aus die Timer zählen sollen, sind in einem Zwischenspeicher abgelegt. Dieser Speicher wird immer dann angesprochen, wenn der Timer beschrieben (POKEn) wird. Beim Auslesen (PEEKen) wird der momentane Zustand des Timers angegeben. Beim Erreichen der Null setzt sich der Timer automatisch auf den Startwert.

Stoppt man einen dieser Timer, so kann kein Interrupt mehr stattfinden, da der Wert Null nicht mehr erreicht wird und im Interrupt Control Register die Bits 0 oder 1 nicht mehr gesetzt werden. Zwei Control Register, wobei jedes für einen Timer zuständig ist, können die Timer beeinflussen.

CI+14	56334	Control Register Timer A
CI+15	56335	Control Register Timer B

Um einen Timer zu stoppen, wird das Bit 0 gelöscht ('0'). Dies ist z.B. beim Kopieren des Zeichen-ROMs notwendig. Es kann auch bestimmt werden, ob die Timer nur einmal bis Null zählen und dann stoppen (Bit3 = „1"). Die Zählfrequenz für alle Timer werden von der Netzfrequenz entnommen. Mit Bit 7 wird festgelegt, ob es sich um eine Frequenz von 50 Hertz oder 60 Hertz (Amerika) handelt (Bit7 gesetzt „1" = 50 Hz).

Eine wirklich sehr interessante Besonderheit ist die im CIA enthaltene Echtzeituhr. Diese 24-Stunden Uhr arbeitet mit einer Genauigkeit von 1/10 Sekunde. Sie läuft im Gegensatz zur „TIS-Uhr" sehr genau. Die Echtzeituhr besteht aus vier Registern.

CI+8	56328	Zehntelsekunden in BCD
CI+9	56329	Sekunden in BCD
CI+10	56330	Minuten in BCD
CI+11	56331	Stunden in BCD

In den jeweiligen Registern liegen mehrere Informationen.

CI+8	0000 zzzz	Die Bits 0-3 geben die Zehntelsekunden an (z). Bit 4-7 sind immer Null.

CI+9	0SSS ssss	Die Bits 0-3 geben die Einersekunden wieder (s). Bit 4-6 die Zehnersekunden (S).Bit7 immer Null.
CI+10	0MMM mmmm	Wie CI+9 für Minuten.
CI+11	PooH hhhh	Die Einer- und Zehnerstunden (H,h). Bit7 kennzeichnet AM/PM (0= vormittag / 1= nachmittag).

Durch die Möglichkeit, diese Register zu laden und auszulesen, läßt sich eine Echtzeituhr aufbauen. Im dem „Uhrenprogramm" wird die Uhrzeit umgerechnet und in die Register gelegt. Mehrere Dinge müssen beachtet werden.

a) Die Zählfrequenz muß auf 50 Hertz gestellt werden. Im Normalzustand ist sie auf 60 Hertz gesetzt (Zeile 60120).

b) Damit die Uhr gestellt werden kann, ist Bit 7 im Register CI+15 zu löschen (Zeile 60130).

Um die Uhrzeit richtig setzen und lesen zu können, muß eine bestimmte Reihenfolge eingehalten werden. Beim Setzen der Zeit werden die Werte in einem Zwischenspeicher abgelegt und erst endgültig beim Setzen der Zehntelsekunden abgespeichert. Ebenso beim Lesen der Zeit. Erst nach dem Auslesen der Zehntelsekunden werden alle anderen Werte übergeben.

```
10 uh=0:gosub 60000
20 uh=1:gosub 60000:goto 20
30 :
60000 rem****************** uhr **********************
60010 ci=56320
60020 hz=ci+14
60030 se=ci+15
60040 hh=ci+11
60050 mm=ci+10
60060 ss=ci+9
60070 zs=ci+8
60080 h1=0:m1=0:s1=0:pm=0
```

```
60090 if uh=1 then 61000
60100 :
60110 rem*************** uhr stellen *********************
60120 poke hz, peek(hz) or 128: rem auf 50 hertz
60130 poke se, peek(se) and 127: rem uhrzeit setzen
60140 print chr$(147)
60150 input " * zeit (hhmmss) :";tm$
60160 if len(tm$) <> 6 then print chr$(145);: goto 60150
60170 h=val(mid$(tm$,1,2))
60180 m=val(mid$(tm$,3,2))
60190 s=val(mid$(tm$,5,2))
60200 if h>23 or m>59 or s>59 then print chr$(145);:
      goto 60150
60210 if h>11 then h1=128: h=h-12
60220 if h<10 then h1=h1+h: goto 60240
60230 h1=h1+16+(h-10)
60240 poke hh, h1
60250 m1=int(m/10)*16+m-int(m/10)*10
60260 poke mm, m1
60270 s1=int(s/10)*16+s-int(s/10)*10
60280 poke ss, s1
60290 poke zs, 0
60300 return
60310 rem*****************************************
60320 :
61000 rem********** uhr auslesen/anzeigen ****************
61010 h=peek(hh)
61020 m=peek(mm)
61040 s=peek(ss)
61050 z=peek(zs)
61060 if h>127 then h=h-128: h1=12
61070 h=int(h/16)*10+h-int(h/16)*16
61080 if h1=12 then h1=h1+h: goto 61120
61100 h1=h
61110 if h1=12 then h1=0
61120 if h1=24 then h1=12
61130 m1=int(m/16)*10+m-int(m/16)*16
61140 s1=int(s/16)*10+s-int(s/16)*16
61150 h$=str$(h1): if h1<10 then h$="0"+right$(h$,1)
61160 m$=str$(m1): if m1<10 then m$="0"+right$(m$,1)
```

```
61170 s$=str$(s1): if s1<10 then s$="0"+right$(s$,1)
61180 tm$=right$(h$,2)+":"+right$(m$,2)+":"+right$(s$,2)
61190 print chr$(18);chr$(19);tm$
61200 return
61210 rem***********************************************
```

Der gesamte Speicheraufbau des C=64 ist schon ein sehr verzwicktes Ding. Für die alltägliche Anwendung sollte das Behandelte mehr als ausreichend sein. Obwohl dieses Kapitel sehr theoretisch war, hoffe ich, daß Sie etwas mehr Einblick in den C=64 erhalten haben.

GRAFIK

B1.0 Wunderland GRAFIK

Ein wesentliches Leistungsmerkmal eines Computers ist heute seine Grafikfähigkeit. Unter Grafik versteht man im allgemeinen die Fähigkeit, Buchstaben, Zeichen, Linien und Farben auf den Bildschirm zu zaubern. Wird heute von Grafikfähigkeit gesprochen, ist ausschließlich die Möglichkeit von hochauflösender Grafik gemeint (HIRES = high resolution). Unter hochauflösender Grafik versteht man das programmierte Setzen oder Löschen von einzelnen Punkten auf dem Bildschirm. Aus je mehr Punkten der Bildschirm besteht, desto feiner werden die Grafiken. Der Bildschirm des C = 64 besteht aus insgesamt 64000 Punkten. 320 Punkte in der X- und 200 in der Y-Achse. Eine Auflösung von 64000 Punkten muß heute als Standard und für eine anspruchsvolle Grafik als notwendig bezeichnet werden. Auch hier hat der C = 64 einen neuen Standard in dem Bereich der Heim- und Hobbycomputer gesetzt. Ingesamt 16 verschiedene Farben stehen im C = 64 für die Darstellung von Zeichen zur Verfügung. Mit 16 Farben ist zwar nicht die gesamte Palette von möglichen Farbkombinationen abgedeckt, sie sind aber im Anwendungfall mehr als ausreichend. Einen Sonderplatz im Bereich der Grafik nehmen die sogenannten SPRITEs (Kobold) ein. Es handelt sich hierbei um Kleingrafiken, die frei über den Bildschirm bewegt werden können. Für Spiele und Animationen bieten sie unzählige Möglichkeiten.
Nicht eine der vielen Grafikmöglichkeiten ist im COMMODORE BASIC eingebunden, somit bleibt eine umständliche Programmierung nicht aus.

B2.0 Bildschirm- und Farb-RAM

Zeichen, die auf dem Bildschirm dargestellt werden, liegen in einem extra Speicher, dem Bildschirmspeicher. Da es mit dem C=64 möglich ist, insgesamt 1000 Zeichen gleichzeitig darzustellen, beträgt die Größe des Bildschirmspeichers ebenfalls 1000 Byte (siehe A3.1, A6.2). Der belegte Adressbereich beginnt bei 1024 und endet bei Adresse 2023. Jeder Speicherplatz ist einem bestimmten Ort auf dem Bildschirm zugeordnet. Die obere linke Ecke des Bildschirms entspricht der Adresse 1024, daneben 1025 usw..

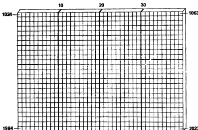


Abb. 9 Bildschirmspeicher

Wird etwas auf dem Bildschirm ausgegeben, so werden alle Zeichen in den entsprechenden Speicherplatz gelegt. Schreiben Sie bitte einmal den Buchstaben „A“ in die obere, linke Ecke des Bildschirms und lesen Sie die Adresse 1024 im Direktmodus aus (PRINT PEEK(1024)). Sie erhalten den Wert „1“. Wieso denn eine „1“? Immer wenn ein Zeichen auf dem Bildschirm dargestellt wird, holt sich der Rechner das Zeichen aus dem Zeichen-ROM. In dem Zeichen-ROM liegen alle Zeichen in einer Reihenfolge, die sich vom ASCII-Code erheblich unterscheidet. Die „1“ besagt, daß das Zeichen, welches an 1. Stelle

im Zeichen-ROM steht, auf dem Bildschirm dargestellt wird und an erster Stelle steht das große A. Auf dem umgekehrten Wege ist der Bildschirmspeicher mit einem Wert (Zeichen) zu laden. Mit

POKE 1024, 1

wird das obere linke Feld mit „1" (entspricht dem „A") geladen, das „A" müßte erscheinen. Doch es erscheint nicht! (bei bestimmten Bedingungen kann das „A" durchaus erscheinen. Bitte nicht wundern. Erklärung folgt). Fahren Sie mit dem Cursor allerdings in die obere linke Ecke, so wird das „A" sichtbar. Die Lösung ist recht einfach: das „A" steht zwar auf dem Bildschirm, aber die Farbe für das Zeichen fehlt. Jedem Bildschirmspeicherplatz ist ein Farbspeicherplatz zugeordnet, der bestimmt, in welcher Farbe das Zeichen erscheinen soll. Im Einschaltzustand und immer dann, wenn der Bildschirm gelöscht wird, wird der Farbspeicher mit der Bildschirmhintergrundfarbe geladen (bei neueren Betriebssystemen (Anfang 1984), wird der Bildschirmspeicher mit der Zeichenfarbe geladen). So ist die Farbe des Zeichens gleich der Hintergrundfarbe und daher nicht zu erkennen. Der Farbspeicher belegt den Bereich 55296 bis 56295.

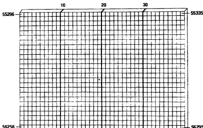


Abb. 10 Farb-RAM

Der Farbspeicherplatz 55296 ist wieder der oberen linken Ecke und damit dem Bildschirmsspeicherplatz 1024 zugeordnet. Jeder Farbspeicherplatz kann mit einer der 16 Farben belegt werden.

0	Schwarz	8	Orange
1	Weiß	9	Braun
2	Rot	10	Hellrot
3	Türkis	11	Grau 1
4	Violett	12	Grau 2
5	Grün	13	Hellgrün
6	Blau	14	Hellblau
7	Gelb	15	Grau 3

Aus der Tabelle sind die Farben mit den entsprechenden Werten zu entnehmen. Um das „A" in der Farbe Rot (2) erscheinen zu lassen, wird der Wert in die entsprechende Farbspeicherstelle gelegt.

POKE 55296, 2

Beim Farbspeicher werden nur die niederwertigen 4 Bits benutzt. Die hochwertigen Bits sind immer gesetzt. Wird ein Farbspeicherplatz ausgelesen, so erhält man immer einen Wert, der um 240 größer ist als der Farbwert.
Der Farbspeicher, der zuvor auf 2 (Rot) gesetzt worden ist, enthält den Wert

1111 0010 = 242.

Mit PEEK(55296) AND 15 kann der tatsächliche Farbwert ermittelt werden.

Zwei kleine Beispiele verdeutlichen den Zusammenhang zwischen Bildschirm- und Farbspeicher.

```
10 for i = 1024 to 1024+39
20 poke i,160:rem inv. space
30 next i
40 for j = 0 to 15
50 for i = 55296 to 55296+39
60 poke i,j
70 next i
80 next j
```

Im ersten Beispiel ist die obere Zeile des Bildschirms (1024-1063) in den Programm-Zeilen 10 bis 30 mit einem inversen SPACE gefüllt worden. Die entsprechenden Farbspeicher erhalten dann nacheinander alle 16 Farben. Es entsteht ein sich ständig farblich ändernder Farbstreifen.

```
10 print chr$(147);
20 print "*** spiele machen nicht immer spass ***"
30 for i=0 to 1000
40 fa=int(rnd(0)*16)
50 po=int(rnd(0)*40)+55296
60 poke po,fa
70 next i
```

Das zweite Beispiel zeigt, wie Buchstaben einer Überschrift ihre Farben zufällig wechseln.

B3.0 ASCII- und Bildschirmcode

Alle Buchstaben und sonstigen Zeichen müssen einem, dem Computer verständlichen, binären Code zugeordnet sein. In Amerika wurde der ASCII-Code (sprich: Aski) entwickelt. ASCII steht für „American Standart Code for Information Interchange". Der ASCII-Code arbeitet mit einer Wortbreite von 7 Bit, so daß insgesamt 128 Zeichen ansprechbar sind. Beim COMMODORE sind mehr als 128 Zeichen ansprechbar, von daher unterscheidet sich das COMMODORE-ASCII (pro Zeichensatz 256 Zeichen) vom Standard-ASCII. Der C=64 besitzt 2 verschiedene Zeichensätze a 256 Zeichen; insgesamt also 512 Zeichen. Der 1. Zeichensatz ist der sogenannte Groß/Grafik-Zeichensatz (Einschaltzustand). Der zweite der Klein/Groß-Zeichensatz. Es ist jeweils nur einer der beiden Zeichensätze gleichzeitig darstellbar. Die Umschaltung zwischen den Zeichensätzen kann durch verschiedene Maßnahmen erfolgen.

1) Umschaltung durch die Tasten C=/SHIFT. Durch das gleichzeitige Betätigen der COMMODORE- und der SHIFT-Taste wird von einem zum anderen Zeichensatz umgeschaltet.

2) Umschalten auf Klein/Groß:
   POKE 53272, 23
   PRINT CHR$(14)

3) Umschalten auf Groß/Grafik:
   POKE 53272, 21
   PRINT CHR$(142)

Alle Zeichen, also beide Zeichensätze, liegen im Zeichen-ROM (siehe A6.0).

Zeichensatz 1 : Adresse 53248 - 55295
Zeichensatz 2 : Adresse 55296 - 57343

Jeder Zeichensatz belegt 2 kByte Speicherplatz. Daraus folgt, daß pro Zeichen 8 Byte belegt werden (2048/256 = 8!). Sieht man sich ein Zeichen genauer an, so erhält man folgendes Muster:

```
Byte 0: 00011000 ( 24)
     1: 00111100 ( 60)
     2: 01100110 (102)
     3: 01100110 (102)
     4: 01111110 (126)
     5: 01100110 (102)
     6: 01100110 (102)
     7: 00000000 (  0)
```

Abb. 11 Buchstabe „A“ als Bitmuster

Ein Zeichen besteht aus 64 Punkten (8*8). Jede Zeile entspricht einem Byte. Betrachtet man die gesetzten Punkte als eine binäre „1“, so erhält man einen entsprechenden Wert. Im Zeichen-ROM selber liegen die Werte für alle Zeichen hintereinander. Nach dem Byte 7 vom Zeichen „A“ folgt das Byte 0 vom Zeichen „B“ usw.. Leider hat sich COMMODORE auch hier wieder eine Besonderheit ausgedacht. Die Reihenfolge der abgelegten Zeichen entspricht nicht dem COMMODORE-ASCII-Code! Es ist eine andere Reihenfolge gewählt worden - der Bildschirmcode. Ein kleines Programm zeigt alle Bildschirmcodes in der oberen linken Ecke an.

```
10 print chr$(147)
20 print:print:print
30 for i = 0 to 255
40 poke 1024,i
50 poke 55296,1
60 print " * bildschirmcode:";i;chr$(145)
70 get a$: if a$ = "" then 50
80 next
```

Das erste Zeichen im Zeichen-ROM ist das „@“ (Klammeraffe). Dann folgen die großen Buchstaben etc.. Ab Bildschirmcode 128 sind alle Zeichen noch einmal invers vorhanden. Das inverse „A“ hat dann den Code 128 + 1 = 129! Wird auf den zweiten Zeichensatz (C=/SHIFT) umgeschaltet, ist deutlich zu sehen, daß nun der Bildschirmcode 1 dem kleinen „a“ entspricht. Zusammenfassend ist zu sagen, daß immer dann, wenn in den Bildschirmspeicher gePOKEt wird, der Bildschirmcode benutzt werden muß. Bei PRINT-Anweisungen hingegen wird mit dem COMMODORE-ASCII gearbeitet wird.

Großes „A“ POKEn : POKE 1024,1
Großes „A“ PRINTen : PRINT CHR$(65)

Eine Tabelle des ASCII- und Bildschirmcodes finden Sie im Anhang G2.2.

Mit den vielen Grafik-Zeichen, die fest im Zeichensatz enthalten sind, lassen sich kleine, aber nützliche Grafiken erstellen. Zum Beispiel einen Rahmen um den Bildschirm.

```
60000 rem************ rahmen ***************
60010 ba=1024: fa=55296
60020 co=1:rem zeichen farbe weiss
60030 for i=0 to 39
60040 poke ba+i,64:poke fa+i,co
60050 poke ba+960+i,64: poke fa+960+i,co
60060 next i
60070 for i = 1 to 23
60080 poke ba+40*i,93: poke fa+40*i,co
60090 poke ba+40*i+39,93: poke fa+40*i+39,co
60100 next i
60110 poke ba,85: poke ba+39,73
60120 poke ba+960,74: poke ba+960+39,75
60130 rem***********************************
```

Gerade für Titelgrafiken von Programmen lassen sich recht nette Sachen anstellen. Aber auch nützliche und praktische Dinge können realisiert werden. Mit zwei Beispielen möchte ich dies unterstreichen.

Als erstes werden Textzeilen, die in TX$ abgelegt sind, von links oder von rechts in den Bildschirm geschoben. Dabei ist zu beachten, daß eine Textzeile nicht mehr als 40 Zeichen besitzt. Die eigentlichen Programme liegen ab Zeile 60000.

```
5 print chr$(147)
10 tx$="            manfred walt r thoma"
20 gosub 61000
30 tx$="                 presentiert"
40 gosub 60000
50 print:print
60 tx$="        moeglichkeiten ohne grenzen"
70 gosub 61000
80 print:print
90 tx$=""
100 gosub 60000
110 tx$=""
120 gosub 61000
130 gosub 61000
140 gosub 61000
150 tx$=""
160 gosub 60000
170 print:print
180 tx$="               mal links ......."
190 gosub 60000
200 tx$="                mal recht
210 gosub 61000
220 end
230 :
60000 rem**************** rechts - links ****************
60010 bl$="                                        "
60020 le=len(tx$): if le< 40 then tx$=tx$+left$(bl$,40-le)
60030 for i = 39 to 0 step-1
60040 printtab(i);left$(tx$,39-i):.print chr$(145);
60050 next i: print
60060 return
```

```
60070 rem*********************************************
60100 :
61000 rem**************** links - rechts ****************
61010 bl$="                                        "
61020 le=len(tx$): if le< 40 then tx$=tx$+left$(bl$,40-le)
61030 for i = 0 to 39
61040 print right$(tx$,i): print chr$(145);
61050 next i: print
61060 return
61070 rem*********************************************
```

Das zweite Beispiel invertiert alle Zahlen auf dem Bildschirm. Dieses Programm kann genutzt werden, um bestimmte Zeichen auf dem Bildschirm deutlich hervorzuheben.

```
60000 rem************ zahlen invertieren ****************
60010 for i = 1024 to 1024+999
60020 if peek(i)>47 and peek(i)<58 then poke i,peek(i)+128
60030 next
60040 rem*********************************************
```

Das Arbeiten mit den vorhandenen Zeichen hat dort seine Grenze, wo Zeichen benötigt werden, die nicht in den Zeichensätzen vorhanden sind. So z. B. die deutschen Sonderzeichen (Ä,Ö,Ü,ß).

B4.0 Neue Zeichen erstellen

Damit der vorhandene Zeichensatz (512 Zeichen) geändert werden kann, ist es notwendig, ihn aus dem ROM- in einen RAM-Bereich zu kopieren. Der RAM-Bereich für den kopierten Zeichensatz muß innerhalb des adressierbaren 16 kByte Bereichs des VIC-Chip liegen (siehe A6.1). Im Einschaltzustand ist der VIC-Chip auf die ersten 16 kByte adressiert, so daß sich der Bereich 12288 bis 16383 als Speicher für den Zeichensatz geradezu anbietet (die letzten 4 kByte im 16 kByte Bereich des Video-Controlers). Beim Kopieren des Zeichen-ROMs taucht eine weitere Schwierigkeit auf. Der Bereich 53248 bis 57343 ist „doppelt" belegt. Einmal durch den Ein-/Ausgabebaustein und weiterhin durch das Zeichen-ROM. Auf das Zeichen-ROM kann normalerweise nur der Video-Controller zugreifen, dem Benutzer ist es nicht möglich, das

Zeichen-ROM zu lesen (siehe A6.0). Beim Auslesen des Bereichs 53248 bis 57343 wird immer auf den Ein-/Ausgabebereich zugegriffen. Um von außen auf das Zeichen-ROM zuzugreifen, muß auf das Zeichen-Rom „gezeigt" werden. In der Adresse 1 dient das Bit 2 als Zeiger auf das Zeichen-ROM.

xxxx x1xx	Normalzustand: Zeichen-ROM kann nur vom VIC gelesen werden.
xxxx x0xx	Zeichen-ROM kann ausgelesen werden.

Löschen des Bit 2 mit POKE 1, PEEK(1) AND 251
Setzen des Bit 2 mit POKE 1, PEEK(1) OR 4

Doch damit nicht genug: der Video-Controler ist quasi ausgeschaltet und nicht arbeitsfähig. Sechzigmal in der Sekunde wird zwischen dem Zeichen-ROM und dem Ein-/Ausgabebereich umgeschaltet. Sowie aber jetzt auf den Video-Controler umgeschaltet wird, findet der Computer ihn nicht und „verabschiedet sich sofort". Das „Umschalten" zwischen den beiden Baugruppen muß unterbunden werden. Das Interrupt-System ist in der Adresse 56334 des CIA 1 abzuschalten, indem das Bit 0 gelöscht wird. Daraufhin ist der Zeichensatz ohne weiteres kopierbar.

```
10 poke 56334, peek(56334) and 254
20 poke 1, peek(1) and 251
30 for i = 0 to 4095
40 poke i+12288, peek(i+53248)
50 next i
60 poke 1, peek(1) or 4
70 poke 56334, peek(56334) or 1
```

Nach dem eigentlichen Kopieren (Zeile 30 bis 50) wird das Interrupt-System wieder normalisiert und der VIC wieder „eingeschaltet".
Der Zeichensatz ist zwar in den RAM-Bereich 12288 bis 16383 kopiert worden, der Video-Controler holt sich aber den Zeichensatz weiterhin aus dem Zeichen-ROM. Das schon bekannte Register 53272 (siehe A6.2) sagt dem Betriebssystem auch, wo der Zeichensatz (Startadresse) zu finden ist. Die höchwertigen 4 Bits dienen zur Bestimmung des Bildschirmspeichers (siehe A6.2), die niederwertigen 4 Bits bestimmen die Startadresse des Zeichensatzes.

Binär	DEZ	Startadresse	
xxxx 0000	0	0	
xxxx 0001	1	1024	
xxxx 0010	2	2048	
xxxx 0011	3	3072	
xxxx 0100	4	4096	Einschaltzustand
xxxx 0101	5	5120	
....	.	....	
xxxx 1100	12	12288	
xxxx 1101	13	13312	
xxxx 1110	14	14336	
xxxx 1111	15	15360	

Abschließend wird mit

POKE 53272, PEEK(53272) AND 240 OR DEZ

dem Betriebssystem mitgeteilt, wo ab sofort der Zeichensatz zu finden ist. Der Wert DEZ ist aus der Tabelle zu entnehmen (Adresse 12288 = 12).
Alle Zeichen liegen in der Reihenfolge des Bildschirmcodes, Byte für Byte im RAM-Bereich 12288-16383. Ohne weiteres können nun die Zeichen geändert werden, indem die neue Form in die entsprechenden Speicher gelegt wird. Da jedes Zeichen aus 8 Byte besteht, ist die Anfangsadresse eines Zeichens sehr leicht zu errechnen:

Adresse = 12288 + 8*CODE

CODE ist der Bildschirmcode des gesuchten Zeichens. Die folgenden 8 Byte ergeben die Form des Zeichens.

B4.1 Zeichengenerator

Um das Erstellen eines neuen, eigenen Zeichensatzes zu erleichtern, bietet das nachfolgende Programm „Zeichengenerator" eine Hilfe. Nach dem Kopieren des Zeichen-ROMs nach 12288 stehen Ihnen folgende Befehle zur Verfügung:

X	Programmende
S	SAVEn des Zeichensatzes auf Disk
L	Laden eines Zeichensatzes von Disk
Axxx	Ändern eines Zeichens xxx = 0 bis 511

```
10 rem************** zeichengenerator ******************
20 poke 53280,0
30 print chr$(147)
40 poke 52,48: poke56,48
50 print " ---- bitte warten ..... kopiere ! ---"
60 :
70 rem*********** zeichensatz kopieren *****************
80 poke 56334, peek(56334) and 254
90 poke 1, peek(1) and 251
100 for i = 0 to 4095
110 poke 12288+i, peek(53248+i)
120 next i
130 poke 1 ,peek(1) or 4
140 poke 56334, peek(56334) or 1
150 poke 53272, peek(53272) and 240 or 12
160 rem*********************************************
170 :
180 rem************** befehls-eingabe *****************
190 print chr$(147)
200 print "    z e i c h e n g e n e r a t o r"
210 print: print
220 print " * befehl :"
230 x=12:y=4:gosub 970
240 input be$
250 b$=left$(be$,1)
260 if b$="l" then 330
270 if b$="s" then 470
280 if b$="a" then 620
290 if b$="x" then end
300 goto 230
310 rem*********************************************
320 :
330 rem*************** laden von zeichensatz *************
340 x=2:y=20:gosub 970
350 input " * load mit namen :";n$
360 open 1,8,2,n$+",s,r"
370 for i = 12288 to 12288+4095
380 input#1 ,j
390 poke i,j
```

```
400 print chr$(19);chr$(18);i
410 next i
420 close 1
430 print chr$(19);"
440 goto 1020
450 rem*********************************************
460 :
470 rem*************** saven von zeichensatz ************
480 x=2:y=20:gosub 970
490 input " * save mit namen :";n$
500 n=len(n$)
510 if n<1 or n>16 then 1020
520 open 1,8,2,n$+",s,w"
530 for i = 12288 to 12288+4095
540 print#1, peek(i)
550 print chr$(19);chr$(18);i
560 next i
570 close 1
580 print chr$(19);"
590 goto 1020
600 rem*********************************************
610 :
620 rem*************** aendern von zeichen **************
630 print:print:print
640 ze=val(mid$(be$,2,3))
650 if ze<0 or ze>511 then 230
660 for i = 12288+8*ze to 12288+8*ze+7
670 de=peek(i):gosub 830
680 print "                    ";chr$(145)
690 gosub 910
700 next i:x=18:y=7
710 for i = 12288+8*ze to 12288+8*ze+7
720 y=y+1:gosub 970
730 input b1$
740 gosub 1080
750 print chr$(145);"
760 print chr$(145)
770 gosub 910
780 poke i,de
```

```
790 next i
800 goto 230
810 rem**************************************************
820 :
830 rem************** dezi in binaer *******************
840 bi$="":di=de
850 di=di/2:d$=".":if di<>int(di) then d$="1"
860 di=int(di):bi$=d$+bi$:if di>0 then850
870 if len(bi$)<8 then bi$="."+bi$:goto 870
880 return
890 rem**************************************************
900 :
910 rem*************** werte ausgeben ******************
920 printtab(7-len(str$(i)));i;
930 printtab(14-len(str$(de)));de,bi$
940 return
950 rem*************************************************
960 :
970 rem************* cursor positionieren **************
980 poke 211, x: poke 214, y:sys 58640
990 return
1000 rem*************************************************
1010 :
1020 rem************ zeile loeschen ********************
1030 x=2:y=20:gosub 970
1040 print "
1050 goto 230
1060 rem*************************************************
1070 :
1080 rem***************** binaer - dezi ****************
1090 de=0:for j=1 to 8
1100 if mid$(bi$,j,1)="1" then de=de+2^(8-j)
1110 next j
1120 return
1130 rem*************************************************
```

B4.2 Arbeiten mit neuem Zeichensatz

Ein riesiger Nachteil beim Arbeiten mit einem neuen Zeichensatz besteht darin, daß der BASIC-Speicher erheblich beschnitten wird (12288). Es bleiben rund 10 kByte Speicherplatz übrig. Nur durch Verlegen des Arbeitsbereichs vom Video-Controler kann dieses Problem umgangen werden (siehe A6.1, A6.2). Ein neuer Zeichensatz wird nach 61440 (die letzten 4 kByte unter dem Kernal) gelegt und der Arbeitsbereich des Video-Controlers auf die letzten 16 kByte verlegt (49152 bis 65535). Der Bildschirmspeicher muß selbstverständlich auch in diesem Bereich liegen und wird nach 49152 gelegt. Die neue Startadresse des Zeichensatzes wird abschließend noch dem Video-Controler mitgeteilt. Ein Programm übernimmt das Laden eines neuen Zeichensatzes und schaltet den Video-Controler auf seinen neuen Arbeitsbereich.

```
100 rem****** zeichensatz laden und verschieben *******
110 print chr$(147)
120 input " * name zeichensatz :";zn$
130 open 1,8,2,zn$+",s,r"
140 for i=61440 to 65535
150 input#1,a: poke i,a
160 print chr$(19);chr$(18);i
170 next i
180 close 1
190 rem*********************************************
200 :
210 rem*********** vic auf 49152 bis 65535 *************
220 poke 56576, peek(56576) and 252 or 0
230 rem*********************************************
240 :
250 rem*********** bildschirm nach 49152  *************
260 poke 53272, peek(53272) and 15 or 0
270 poke 648, 49152/256
280 rem*********************************************
290 :
300 rem*********** zeichensatz ab  61440  *************
310 poke 53272, peek(53272) and 240 or 12
320 rem*********************************************
330 :
```

```
340 print chr$(147);
350 print " der bildschirm liegt nun bei"
360 print "   49152 bis 50151 !!!!"
370 print " der zeichensatz liegt bei"
380 print "   61440 bis 65535"
390 rem new
400 ********************************************
```

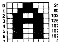
B5.0 Zeichen noch bunter

Die Farbe für Zeichen (Standardzeichensatz oder neuer Zeichensatz) wurde bisher ausschließlich mit einem gesetzten oder nicht gesetzten Bit interpretiert. Ist ein Bit gesetzt, so erscheint der Punkt in der Zeichenfarbe bzw. in der Farbe, die im entsprechenden Bildschirmspeicherplatz vorhanden ist. Ein gelöschter Punkt wird immer in der aktuellen Hintergrundfarbe dargestellt und ist damit nicht sichtbar. Zwei Möglichkeiten erlauben es, ein Zeichen noch bunter darzustellen.

B5.1 Mehrfarbmodus

Sehen Sie sich noch einmal den Buchstaben „A" genauer an. Jedes als „1" gekennzeichnete Bit wird in der aktuellen Zeichenfarbe dargestellt.

```
00 01 10 00
00 11 11 00
01 10 01 10
01 10 01 10
01 11 11 10
01 10 01 10
01 10 01 10
00 00 00 00
```



Durch das Zusammenfassen von je zwei Bits sind 4 Kombinationen möglich, die jeweils eine andere Farbe repräsentieren können (00, 01, 10, 11). Das „A" wurde hier schon in Gruppen von je zwei Bit aufgegliedert. Die Informationen für die Farben liegen neben der Hintergrundfarbe (53281) und der Zeichenfarbe (646 bzw. Farb-RAM) in zwei weiteren Farbregistern (53282 und 53283).

00	Hintergrundfarbe	53281
01	Farbregister	53282
10	Farbregister	53283
11	Aktuelle Farbe	646

Je nach Bitkombination werden die Farben aus dem entsprechenden Farbregister geholt. Dem Betriebssystem muß mitgeteilt werden, daß die Zeichen nicht mehr bitweise sondern paarweise interpretiert werden (Mehrfarbmodus).

```
Adresse 53270 : xxx0 xxxx   normal
        53270 : xxx1 xxxx   Mehrfarbmodus
```

Das Bit 4 in der Adresse 53270 stellt den Rechner auf Mehrfarbmodus ein. Durch Setzen des Bits 4 auf „1" wird auf Mehrfarbmodus geschaltet.

```
Mehrfarbmodus  an  : POKE 53270, PEEK(53270) OR 16
Mehrfarbmodus  aus : POKE 53270, PEEK(53270) AND 239
```

Im Mehrfarbmodus ist es weiterhin möglich, mehrfarbige und einfarbige Zeichen gleichzeitig darzustellen. Die Information, ob ein Zeichen im Mehrfarbmodus oder normal darzustellen ist, wird aus dem Farb-RAM bzw. der aktuellen Zeichenfarbe entnommen. Immer dann, wenn das Bit 3 im entsprechenden Farb-RAM gesetzt ist, wird das Zeichen im Mehrfarbmodus erscheinen. Bei einem gelöschten Bit 3 findet eine normale Darstellung statt. Daraus folgt, daß der Zeichenfarbwert für mehrfarbige Zeichen größer als 7 sein muß.

```
Adresse 646 :  xxxx 0 000    Schwarz   ( 0)
               xxxx 0 001    Weiß      ( 1)
               xxxx 0 010    Rot       ( 2)
               ..........
               xxxx 1 000    Orange    ( 8)
               xxxx 1 001    Braun     ( 9)
               ..........
               xxxx 1 111    Grau 3    (15)
```

Wird als Zeichenfarbe ein Wert zwischen 0 und 7 (Schwarz bis Gelb) oder aber ein Farb-RAM mit diesen Werten geladen, so werden die Zeichen nicht als Mehrfarbzeichen interpretiert. Ist der Farbwert hingegen größer als 7 (Orange bis Grau 3), findet eine mehrfarbige Darstellung statt.

```
10 rem************* mehrfarbmodus **************
20 print chr$(147)
30 poke 53280, 0:  rem rahmenfarbe          schwarz
40 poke 53281, 6:  rem hintergrund  (00)    blau
50 poke 53282, 1:  rem farbe 2      (01)    weiß
60 poke 53283, 0:  rem farbe 3      (10)    schwarz
70 poke   646,14:  rem farbe 1      (11)    hellblau
80 rem************ mehrfarbmodus an *************
90 poke 53270, peek(53270) or 16
100 for i= 0 to 10
110 print " * mehrfarbmodus !!!!"
120 next i
130 get a$: if a$ = "" then 130
140 rem*********** mehrfarbmodus aus *************
150 poke 53270, peek(53270)  and 239
160 end
```

Das obige Beispielprogramm schaltet den Mehrfarbmodus ein und PRINTet zehn Zeilen. Besitzen Sie keinen Farbfernseher oder Farbmonitor, so sehen Sie nur verschiedene Kontraste, so daß der eigentliche Buchstabe kaum noch zu erkennen ist. Wählen Sie einmal eine Zeichenfarbe (Zeile 70) kleiner als 8 und Sie werden bemerken, daß keine mehrfarbige Darstellung mehr stattfindet.

B5.2 Erweiterte Hintergrundfarben

Der Hintergrund eines Zeichens war bisher immer abhängig von der Hintergrundfarbe, die im Register 53281 abgelegt ist. Es war somit nicht möglich, die Hintergrundfarbe eines Zeiches unabhängig von der Bildschirmfarbe zu ändern. Doch auch hier bietet der C=64 eine Lösung an. Somit ist es möglich,

ein rotes Zeichen mit gelben Hintergrund auf einem blauen Bildschirm darzustellen. Es lassen sich allerdings nur 64 verschiedene Zeichen darstellen und das hat seinen Grund. Die Information, welche Farbe der Hintergrund tragen soll, wird aus dem Zeichen (Bildschirmcode) selber entnommen. Das Bit 6 und Bit 7 geben die Hintergrundfarbe an und haben somit keinen Einfluß auf das Zeichen selber. Daher sind nur noch die Bildschirmcode-Zeichen 0 bis 63 möglich (xx00 0000 bis xx11 1111). Je nach Kombination der Bit 7 und 6 wird eine der vier möglichen Hintergrundfarben gewählt (00, 01, 10, 11).

00	Adresse 53281	Hintergrundfarbe 1
01	Adresse 53282	Hintergrundfarbe 2
10	Adresse 53283	Hintergrundfarbe 3
11	Adresse 53284	Hintergrundfarbe 4

Die Register 53281 bis 53284 können jeweils mit einem zulässigen Farbwert von 0 bis 15 geladen werden.
Wird ein Zeichen mit dem Bildschirmcode 0 - 63 dargestellt, holt es sich seine Hintergrundfarbe aus dem Register 53281, da das Bit 7 und Bit 6 jeweils Null beinhaltet (00 111111 = 63). Bei einem Bildschirmcode von 65 wird die Hintergrundfarbe aus dem Register 53282 geholt (01 000001 = 65). Das eigentliche Zeichen besteht nur aus den niederwertigen 6 Bits, so daß sich wiederum das „A" ergibt (xx 000001 = 1 !!). Hier liegt auch die Erklärung, warum nur die ersten 64 Zeichen dargestellt werden können.

Normale Zeichen	0 - 63	00 111111	Farbe aus 53281
SHIFT Zeichen	64 - 127	01 111111	Farbe aus 53282
Invers Zeichen	128 - 191	10 111111	Farbe aus 53283
SHIFT/invers	192 - 255	11 111111	Farbe aus 53284

Beim folgenden Demonstrationprogramm geben Sie bitte in den Zeilen 120 und 140 den Text (auch die Leerzeichen) geSHIFTet ein. Das bedeutet gleichzeitiges Betätigen der SHIFT- und einer anderen Taste.

```
10 rem*********** erweiterter hintergrund ***********
20 print chr$(147);chr$(14)
30 poke 53280, 0:  rem rahmenfarbe          schwarz
40 poke 53281, 2:  rem hintergrund1 (00)    rot
50 poke 53282, 1:  rem hintergrund2 (01)    weiss
```

```
60 poke 53283,13:  rem hintergrund3 (10)    hellgruen
70 poke 53284, 0:  rem hintergrund4 (11)    schwarz
80 poke   646, 4:  rem zeichenfarbe         violett
90 rem********* erweiterter hintergrund an ***********
100 poke 53265, peek(53265) or 64
110 print "   normale darstellung  ":          rem 53281
120 print "   NORMALGESHIFTET      ":          rem 53282
130 print chr$(18);"   invers              ":rem 53283
140 print chr$(18);"   INVERS GESHIFTET    ":rem 53284
150 get a$: if a$ = "" then 150
160 rem******** erweiterer hintergrund aus ************
170 poke 53265, peek(53265) and 191
180 end
```

Mit den vorhandenen oder selbst definierten Zeichen lassen sich schon recht viele und bunte Grafiken erstellen, wobei das Ergebnis ausschließlich von Ihrer Kreativität abhängig ist.

B6.0 SPRITE - Kleingrafiken

SPRITEs sind freibewegliche Kleingrafiken mit einer Größe von 24 * 21 Bildschirmpunkten. Sie lassen sich „ruckfrei" bewegen und erzeugen so den Eindruck, daß sie über den Bildschirm schweben. Insgesamt lassen sich maximal 8 (verschiedene) Sprites gleichzeitig darstellen. Es ist möglich, sie in beide Achsrichtungen zu dehnen und damit jeweils um den Faktor 2 zu vergrößern. Auch in der Farbgebung stehen einfarbige oder mehrfarbige Sprites zur Verfügung (vierfarbig).

B6.1 Definition eines Sprites

Ein Sprite besteht aus einem 24 * 21 Punkte-Feld. Jeder dieser 504 Punkte kann entweder „ein- oder ausgeschaltet" sein. Die Kombination von „ein- oder ausgeschalteten" Punkten ergibt ein Muster (Figur...etc); den Sprite.

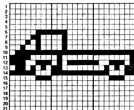


Abb. 12 Sprite ohne Daten

Oben ist ein Spritemuster dargestellt, wobei die ausgefüllten Flächen einen gesetzten Punkt repräsentieren. Die 504 Punkte entsprechen 504 Bits, dies sind iederum 63 Byte (504/8 = 63). Daraus folgt, daß für jeden Sprite irgendwo im Speicherbereich des Rechners 63 Byte reserviert und entsprechend definiert werden müssen.

B6.2 Wohin die Sprite-Daten legen ?

Sprites werden wie alle Grafikausgaben vom Video-Controler gesteuert. Auch auf die Form des Sprites muß der Video-Controler zugreifen können. Im Einschaltzustand verwaltet der Video-Controler die ersten 16 kByte, so daß die Grafikform (63 Bytes) hier zu finden sein muß (siehe A6.1). Es bieten sich mehrere günstige Plätze für Sprite-Daten an, die zum Einen den BASIC-Speicherplatz nicht beschneiden und zum Anderen vom Rechner nicht benutzt werden:

673 - 767 nicht benutzt
828 - 1019 Kassettenpuffer

Der Kassettenpuffer wird immer nur dann benutzt, wenn Daten zwischen dem Rechner und der Datasette ausgetauscht werden. Es können ohne weiteres Daten in diesem Bereich abgelegt werden. Bitte achten Sie jedoch darauf, daß diese immer zerstört werden, wenn die Datasette benutzt wird!
In dem vom Video-Controler adressierbaren 16 kByte-Bereich können Daten für einen Sprite an rund 256 verschiedenen Stellen liegen. Man teilt daher den 16 kByte-Bereich in Blöcke à 64 Byte auf 256*64 = 16384).
So liegt z. B. Block Nummer 0 im Speicherbereich 0 bis 63. Innerhalb von Block 0 dürfen natürlich keine Daten abgelegt werden, da sie wichtige Informationen auf der Zero-Page überschreiben würden. Es bieten sich also die oben bereits genannten Bereiche an.

Block	11	=	11*64	=	704	bis	767
Block	13	=	13*64	=	832	bis	895
Block	14	=	14*64	=	896	bis	959
Block	15	=	15*64	=	960	bis	1023

Die Blöcke 13 bis 15 liegen im Kassettenpuffer, Block 11 in einem nicht benutzten Bereich. Hier können die Formen für vier verschiedene Sprites abgelegt und abgerufen werden. Sollen mehr als vier Sprites definiert werden, muß entweder ein Teil des BASIC-RAM benutzt werden oder aber der Arbeitsbereich des Video-Controlers in einen anderen Bereich verschoben werden (siehe A6.1). Die Arbeitsweise ist in anderen Bereichen identisch mit der hier beschriebenen Arbeitsweise.

B6.3 Wie sehen die Sprite-Daten aus ?

Aus der Sprite-Grafik ist deutlich der Aufbau eines Sprite ersichtlich. Die 63 Bytes teilen sich wie folgt auf: die ersten 8 Punkte ab der oberen, linken Ecke bilden das erste Byte. Die folgenden 8 Punkte Byte 2, darauf Byte 3. Nun geht es in der zweiten Zeile weiter: Byte 4, Byte 5, Byte 6 usw.. Jedes ausgefüllte Feld repräsentiert ein gesetztes Bit des jeweiligen Bytes.

Zeile			
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	7	224	0
7	12	32	0
8	20	32	0
9	36	32	0
10	127	255	255
11	240	64	1
12	231	95	157
13	106	192	43
14	248	255	227
15	7	0	28
16	0	0	0
17	0	0	0
18	0	0	0
19	0	0	0
20	0	0	0
21	0	0	0

Abb. 13 Sprite mit Daten

Jedes der 63 Bytes wird in eine Dezimalzahl umgewandelt (siehe G1.1). Die Dezimalwerte stehen in den Spalten rechts neben der Sprite-Grafik. Für die Entwicklung eigener Sprites steht Ihnen im Anhang ein „Sprite-Entwurfsblatt" zur Verfügung (siehe G2.0). Nachdem der Sprite in Dezimalzahlen umgerechnet worden ist, werden alle Sprite-Daten in DATA-Zeilen eingefügt.

```
10000 rem************** daten für sprite ****************
10010 data 000,000,000,000,000,000,000,000,000,000,000,000
10020 data 000,000,000,007,224,000,012,032,000,020,032,000
10030 data 036,032,000,127,255,255,240,064,001,231,095,157
10040 data 106,192,043,248,255,227,007,000,028,000,000,000
10050 data 000,000,000,000,000,000,000,000,000,000,000,000
10060 data 000,000,000
10070 rem*********************************************
```

Damit der Video-Controler auf die Daten zugreifen kann, müssen sie in einen der Blöcke 11, 13, 14 oder 15 gelegt werden.

```
100 rem********** einlesen der Daten in block 11 *********
110 for i = 11*64 to 11*64+62: rem 704 bis 766
120 read x: poke i,x
130 next i
140 rem ***********************************************
```

Beim Arbeiten mit mehreren verschiedenen Sprites werden auch die anderen Sprites wie beschrieben in dezimale Zahlen zerlegt und dann als DATA-Zeilen in das Programm eingebunden. Die Daten für einen zweiten Sprite müssen selbstverständlich in einen anderen, noch freien Block gelegt werden (z.B 13, 14 oder 15).

B6.4 Wo liegen die Daten für welchen Sprite?

Der C=64 besitzt die Möglichkeit, insgesamt 8 Sprites gleichzeitig auf dem Bildschirm darzustellen. Jeder der 8 möglichen Sprites kann eine andere Form besitzen. Acht sogenannte „Sprite-Zeiger" zeigen auf den Speicherbereich, wo die Daten für den entsprechenden Sprite liegen. Die Sprite-Zeiger liegen direkt vor dem BASIC-RAM in den Adressen 2040 bis 2048:

2040	Zeiger für Sprite 0
2041	Zeiger für Sprite 1
2042	Zeiger für Sprite 2
2043	Zeiger für Sprite 3
2044	Zeiger für Sprite 4
2045	Zeiger für Sprite 5
2046	Zeiger für Sprite 6
2047	Zeiger für Sprite 7

Der Sprite-Zeiger zeigt auf einen der 256 möglichen Blöcke, in dem sich die Daten befinden können. Befinden sich die Daten für Sprite 2 in dem Block 15 (960-1023), so ist der Zeiger mit 15 zu laden.

POKE 2042, 15 Sprite 2 aus Block 15!

Mit Hilfe des gezeigten Programms werden die Daten, auf die der Sprite 0 zugreifen soll, in Block 11 (Zeile 110) abgelegt. Der Sprite-Zeiger 2040 ist also mit 11 zu laden.

```
200 rem************* sprite 0 aus block 11 *************
210 poke 2040, 11
220 rem*********************************************
```

Sollen mehrere Sprites die gleiche Form erhalten, so brauchen sie ihre Daten nur aus dem gleichen Block zu holen (z.B. POKE 2043, 11: POKE 2047, 11).

Anzumerken ist noch, daß bei einer Verlegung des Arbeitsbereiches des Video-Controlers in einen anderen Bereich (siehe A6.1) sich auch die Sprite-Zeiger an einer entsprechend verschobenen Stelle befinden !

B6.5 Einschalten der Sprites

Endlich ist es soweit, daß der Sprite Schritt für Schritt auf dem Bildschirm erscheinen kann. Zunächst einmal muß bestimmt werden, welcher der 8 möglichen Sprites auf dem Bildschirm erscheinen soll. Für alle weiteren Maßnahmen ist einmal mehr der Video-Controler zuständig (53248 - 54271). Alleine 33 Register dienen ausschließlich der Sprite-Behandlung. Zur Vereinfachung wird folgend die Startadresse des Video-Controlers mit „V" bezeichnet.

```
300 rem************ startadresse vic ******************
310 v = 53248
320 rem*********************************************
```

Für das Einschalten eines oder mehrerer Sprites ist das Register V+21 (53269) zuständig. Jedes Bit stellt einen „Einschalter" für ein Sprite dar, wobei Bit 0 für Sprite 0 und Bit 7 für Sprite 7 verantwortlich ist.

```
76543210 Sprite Nr.
xxxxxxxx V+21
```

Ist ein Bit gesetzt ('1'), wird der entsprechende Sprite aktiviert und erscheint damit auf dem Bildschirm. Bei einem gelöschten Bit ('0') ist der Sprite ausgeschaltet (nicht sichtbar). Befindet sich im Register V+21 eine Null,

0000 0000 = 0

so sind alle Sprites ausgeschaltet. Damit Sprite 0 aktiviert wird, muß Bit 0 des Registers V+21 gesetzt ('1') werden.

0000 0001 = 1

Der dezimale Wert (hier 1) ist in das Register zu POKEn.

```
POKE V+21, 1
```

Um zwei oder mehrere Sprites zu aktivieren, brauchen nur die entsprechenden Bits gesetzt zu werden.

0010 0001	= 17	Sprite 0 + Sprite 5
1100 1000	= 200	Sprite 3 + Sprite 6 + Sprite 7
1111 1111	= 255	alle Sprites aktiviert

```
400 rem********** einschalten sprite 0 *****************
410 poke v+21, 1
420 rem*********************************************
```

Obwohl der Sprite aktiviert und damit auf dem Bildschirm sichtbar sein sollte, ist er aus folgenden Gründen noch nicht zu sehen:

1) die Farbe des aktivierten Sprites ist noch nicht bestimmt und

2) die Ausgangsposition des Sprites ist noch nicht definiert.

B6.6 Farbe für jeden Sprite

Jeder Sprite kann eine der 16 möglichen Farben annehmen (siehe B2.0). Ein gesetzter Punkt wird in der Spritefarbe, ein nicht gesetzter in der Hintergrundfarbe des Bildschirms interpretiert (Adresse 53281). 8 Sprite-Farbregister bestimmen für jeden Sprite die Farbe.

V+39	53287	Farbe Sprite 0
V+40	53288	Farbe Sprite 1
V+41	53289	Farbe Sprite 2
V+42	53290	Farbe Sprite 3
V+43	53291	Farbe Sprite 4
V+44	53292	Farbe Sprite 5
V+45	53293	Farbe Sprite 6
V+46	53294	Farbe Sprite 7

Beim Sprite-Farbregister werden, wie auch beim normalen Farb-RAM, nur die niederwertigen 4 Bits genutzt. Die hochwertigen 4 Bits werden vom Betriebssystem immer auf „1" gesetzt (siehe B2.0).

Damit der Sprite 0 in der Farbe Weiß (1) erscheint, ist das Farbregister V+39 mit „1" zu laden.

```
500 rem********* farbe für sprite 0 (weiss) ************
510 poke v+39,  1
520 rem**********************************************
```

B6.7 Wo stehen die Sprites?

Endlich wird der letzte Schritt vollzogen, um einen Sprite sichtbar zu machen. Doch zunächst einen kleinen Rückblick auf den Bildschirmaufbau.

Der Bildschirm des C=64 besteht aus 320 Bildpunkten in der X-Achse und 200 Bildpunkten in der Y-Achse. Die Koordinate X=0; Y=0 befindet sich in der oberen, linken Ecke des Bildschirmes. Ein Sprite kann nun innerhalb dieses Koordinatensystems positioniert werden. Als Ausgangspunkt gilt die obere, linke Ecke eines Sprite. Wird der Sprite auf die Koordinaten X=160 / Y=100 positioniert, so erscheint er ungefähr in der Bildschirmmitte. Für jeden Sprite gibt es ein X- und ein Y-Positionsregister.

V	53248	X-Position Sprite 0
V+1	53249	Y-Position Sprite 0
V+2	53250	X-Position Sprite 1
V+3	53251	Y-Position Sprite 1
V+4	53252	X-Position Sprite 2
V+5	53253	Y-Position Sprite 2
...	...	...
V+12	53260	X-Position Sprite 6
V+13	53261	Y-Position Sprite 6
V+14	53262	X-Position Sprite 7
V+15	53263	Y-Position Sprite 7

```
600 rem************* sprite demo 1 *********************
610 poke v+1, 100: rem y-achse sprite0
620 for x = 255 to 0 step-1
630 poke v, x: rem x-achse sprite0
640 next x
650 rem*********************************************
900 get a$: if a$ = "" then 600
910 poke v+21, 0: rem alle sprites ausschalten
920 end
```

Der erste Sprite saust über den Bildschirm. In der FOR-NEXT-Schleife wird die X-Position von 255 nach 0 verändert und der „LKW" bewegt sich von rechts nach links. Die Y-Position wurde in der Zeile 610 mit dem Wert 100 festgelegt. Ändern Sie einmal diesen Wert, z.B. POKE V + 1, 40. Sie werden bemerken, daß der Sprite jetzt sehr weit oben von rechts nach links fährt. Ändern Sie den Wert weiter gegen Null, ist er irgentwann nicht mehr sichtbar. Der Sprite fährt außerhalb des Bildschirmes. Derselbe Effekt tritt auf, wenn die Y-Position größer als 230 wird. Der Bereich, wo ein Sprite noch voll sichtbar ist, unterscheidet sich von der tatsächlichen Bildschirmgröße. Hier sollten Sie ein bißchen probieren.

Sicherlich wird es Ihnen sofort ins Auge gefallen sein, daß der „LKW" nicht von ganz rechts gestartet ist, sondern aus dem letzten Drittel des Bildschirms. Damit er von weiter rechts starten kann, müßte die X-Position größer 255 werden. In einer 8-Bit Speicherstelle lassen sich aber nur Werte bis 255 darstellen. Größere Werte müssen in mehrere Bytes zerlegt werden (siehe G1.3). Für die X-Position stehen insgesamt 320 Punkte zur Verfügung, die aber nicht mit einem Byte angesprochen werden können. Für die X-Position wird eine weitere Speicherstelle benötigt. Dies ist das Register V + 16 (53264). Jedes Bit des Registers V + 16 ist wieder für ein Sprite zuständig. Bit 0 für Sprite 0 bis Bit 7 für Sprite 7. Ist ein Bit gesetzt ('1'), erhöht sich die X-Position des entsprechenden Sprites um 256.

V + 16	53264	0000 0001	X-Position Sprite0 um 256 erhöht.

Die tatsächliche X-Position errechnet sich dann aus dem „normalen" X-Positionsregister und dem gesetzten oder gelöschten Bit in V + 16. Beispiel:

V + 16 = 0000 0001	X-Position	+	256
V + 1 = 0000 1111	X-Position		15
Tatsächliche Position			271

Mit einer normalen Zerlegung in High- und Lowbyte kann ein Übertrag errechnet werden. Ergänzen Sie das Programm mit den Zeilen:

```
620 for x = 319 to 0 step-1
625 hb=int(x/256): lb=x-Hb*256
627 if hb=0 then poke v+16,0: goto 630
628 poke v+16, 1
630 poke v, lb
```

Der Sprite bewegt sich nun von ganz rechts nach links. Auch hier tritt die Erscheinung auf, daß am Bildschirmrand der Sprite teilweise nicht sichtbar ist. Die Ursache liegt wiederum in der nur teilweisen Nutzung des Bildschirms.

B6.8 Dehnen in X- und Y-Richtung

Sprites werden normal in einer Größe von 24*21 Bildschirmpunkten dargestellt. Es besteht die Möglichkeit, den Sprite in der X- und/oder Y-Achse zu vergrößern und zwar jeweils um den Faktor 2. Wird er z.B. in der X-Achse gedehnt, so besteht er jetzt aus 48*21 Bildschirmpunkten. Jeder Spritepunkt wird doppelt gesetzt: aus einem gesetzten Punkt werden zwei. Ebenso ist es möglich, den Sprite in der Y-Achse zu dehnen. Dabei entsteht ein 24*42 Muster. Durch das gleichzeitige Dehnen in X- und Y-Richtung wird der gesamte Sprite um den Faktor 4 vergrößert.

Für das Dehnen in der Y-Achse ist das Register V+23 (53271) zuständig. Hier repräsentiert jedes Bit wieder einen „Einschalter" für jeden Sprite. Bit 0 für Sprite 0 und Bit7 für Sprite 7. Ist ein Bit gesetzt, wird der entsprechende Sprite Y-gedehnt dargestellt.

```
POKE V+23,  1  0000 0001  Sprite 0 Y-gedehnt
POKE V+23, 16  0001 0000  Sprite 4 Y-gedehnt
POKE V+23,106  0110 1010  Spitte1+3+5+6 Y-gedehnt
```

Das Dehnen in X-Richtung verwaltet das Register V+29 (53277). Es arbeitet genauso wie das Y-Dehnregister. Ein gesetztes Bit ('1') schaltet die X-Dehnung ein.

```
POKE V+29,  1  0000 0001  Sprite 0 X-gedehnt
POKE V+29,  3  0000 0011  Sprite 0+1 X-gedehnt
POKE V+29,255  1111 1111  alle Sprites X-gedehnt
```

Ergänzen Sie das Programm folgendermaßen:

```
614 poke v+23, 1: rem Sprite 0 Y-dehnen
615 poke v+29, 1: rem Sprite 0 X-dehnen
```

Jetzt fährt ein doch schon beachtlicher LKW über den Bildschirm. Das Dehnen in nur einer Richtung verzerrt den Sprite teilweise beachtlich. Berücksichtigen Sie dies bei der Entwicklung eines Sprites.

Als Zusammenfassung des bisher behandelten Themas ist noch einmal das Programm zusammengefaßt und mit den Zeilen 600 bis 690 ergänzt worden. Es werden 4 Sprites aktiviert, die einfachheitshalber ihre Form alle aus Block 11 holen. Sie erhalten alle eine andere Farbe und werden verschieden gedehnt.

```
10 rem****************** sprite demo ********************
20 print chr$(147): poke 53280,0
30 rem**********************************************
50 :
100 rem************* einlesen der daten in block 11 ******
110 for i=11*64 to 11*64+62: rem 704 bis 766
120 read x: poke i,x
130 next i
140 rem**********************************************
150 :
200 rem***************** sprite0 aus block 11 ***********
210 poke 2040, 11
220 rem**********************************************
250 :
300 rem************** startadresse vic *****************
310 v = 53248
320 rem**********************************************
350 :
400 rem************* einschalten sprite0 ****************
410 poke v+21, 1
420 rem**********************************************
450 :
500 rem************* farbe sprite0 (weiss) ***************
510 poke v+39, 0
```

```
520 rem*********************************************
550 :
600 rem********** beispiel mit 4 sprite (0-3) ************
610 rem sprite0 in zeile 210 bestimmt
620 poke 2041, 11: rem sprite1
630 poke 2042, 11: rem sprite2
640 poke 2043, 11: rem sprite3
650 :
660 poke v+1, 50:rem y fuer sprite0
670 poke v+3,100:rem y fuer sprite1
680 poke v+5,150:rem y fuer sprite2
690 poke v+7,200:rem y fuer sprite3
700 :
710 poke v+21, 15: rem 0000 1111
720 :
730 rem farbe sprite0 in zeile 510
740 poke v+40, 1:rem farbe sprite1
750 poke v+41, 4:rem farbe sprite2
760 poke v+42, 8:rem farbe sprite3
770 :
780 poke v+23, 12:rem 00001100 dehnen y
790 poke v+29, 10:rem 00001010 dehnen x
800 :
810 for x = 255 to 0 step -1
820 poke v  ,x:rem x fuer sprite0
830 poke v+2,x:rem x fuer sprite1
840 poke v+4,x:rem x fuer sprite2
850 poke v+6,x:rem x fuer sprite3
860 next x
870 :
900 rem********** weiter - abfrage (taste) ***************
910 get a$: if a$ = "" then 600
920 poke v+21,  0
930 end
940 rem*********************************************
950 :
10000 rem************** daten fur sprite *****************
10010 data 000,000,000,000,000,000,000,000,000,000,000,000
10020 data 000,000,000,007,224,000,012,032,000,020,032,000
```

```
10030 data 036,032,000,127,255,255,240,064,001,231,095,157
10040 data 106,192,043,248,255,227,007,000,028,000,000,000
10050 data 000,000,000,000,000,000,000,000,000,000,000,000
10060 data 000,000,000
10070 rem**********************************************
```

B6.9 Sprite-Sprite-Kollision

Speziell bei Spielen, und besonders hierfür sind Sprites prädestiniert, ist eine Abfrage der Berührung zweier Sprites sehr nützlich. Treffen sich zwei oder mehrere Sprites, so kann im Programm zu einem anderen Teil gesprungen werden, um ein Geräusch o.ä. zu erzeugen. Ein spezielles Register (V+30) registriert eine Sprite-Sprite-Kollision. Das Register V+30 (53278) ist wieder so aufgebaut, daß jedes Bit einen Sprite repräsentiert. Bit 0 für Sprite 0 bis Bit 7 für Sprite 7. Treffen sich zwei Sprites, so werden die entsprechenden Bits gesetzt ('1'). Als Beispiel treffen sich Sprite 0 und Sprite 3:

V+30 53278 0000 1001 = 9

Jeder Sprite, der an dieser Kollision beteiligt ist, setzt sein Bit. Unabhängig davon, ob an der Kollision 2 oder mehr Sprites beteiligt sind.

```
0011 0001  =  49  Kollision Sprite 0+4+5
0000 1111  =  15  Kollision Sprite 0+1+2+3
0000 0000  =   0  keine Kollision
```

Findet keine Kollision zwischen Sprites statt, enthält das Register V+30 den Wert Null. Mit einer einfachen Abfrage kann festgestellt werden, ob eine Kollision stattgefunden hat.

```
IF PEEK(V+30) = 3 THEN ... KOLLISION Sprite 0+1
```

Beim Auslesen (PEEKen) des Registers V+30 tritt eine Besonderheit ein. Immer dann, wenn das Register gelesen wird, wird gleichzeitig das Register gelöscht! Befindet sich zum Beispiel im Register V+30 der Wert 3 und er wird mit

```
PRINT PEEK(V+30)
```

ausgelesen, so wird der Wert „3" ausgegeben und anschließend das Register auf Null gesetzt. Wird es ein zweites mal abgefragt, bleibt der Wert Null erhalten. Erst bei einer weiteren Kollision werden die Bits neu gesetzt.

B6.10 Sprite-Hintergrund-Kollision

Berührt ein Sprite ein Hintergrundzeichen, z.B. Schrift, Grafik etc, so wird ebenfalls eine Meldung ausgegeben. Das Register V+31 (53279) ist identisch dem Sprite-Sprite-Kollisionsregister aufgebaut. Tritt eine Sprite-Hintergrund-Kollision auf, wird das entsprechende Bit gesetzt.

V+31	0000 0001 = 1	Kollision Sprite0/Zeichen
V+31	0000 1000 = 8	Kollision Sprite3/Zeichen

Nach einem Auslesen enthält das Register V+31 ebenfalls den Wert Null.

```
10 rem************** kollisionen **********************
20 print chr$(147): v=53248: poke 53280,0: poke 646,0
30 for i = 11*64 to 11*64+62: rem sprite0 einlesen
40 read x: poke i,x
50 next i
60 for i = 13*64 to 13*64+62: rem sprite1 einlesen
70 read x: poke i,x
80 next i
90 :
100 poke 2040, 11: rem sprite0 aus block 11
110 poke 2041, 13: rem sprite1 aus block 13
120 :
130 poke v+39, 1:  rem farbe sprite0
140 poke v+40, 5:  rem farbe sprite1
150 :
160 poke v+23, 3:  rem y-dehnen sprite0+1
170 poke v+29, 3:  rem x-dehnen sprite0+1
180 :
185 poke v+21, 3:  rem sprite0+1 einschalten
190 poke v+ 1, 100:rem y-position sprite0
200 poke v+ 3, 100:rem y-position sprite1
210 poke v,200:    rem x-position sprite0
```

```
220 :
230 for i = 0 to 255
240 poke v+2, i:   rem x-position sprite1
250 print chr$(19);"sprite-sprite kollision:";peek(v+30)
255 print "sprite-hintergrund koll:";peek(v+31)
260 next i
270 print:print:print:print:print
280 print tab(15);"xxxx"
290 goto 230
300 :
10000 rem*************** daten f. sprite0 ****************
10010 data 000,000,000,000,000,000,000,000,000,000,000,000
10020 data 000,000,000,007,224,000,012,032,000,020,032,000
10030 data 036,032,000,127,255,255,240,064,001,231,095,157
10040 data 106,192,043,248,255,227,007,000,028,000,000,000
10050 data 000,000,000,000,000,000,000,000,000,000,000,000
10060 data 000,000,000
10070 rem**********************************************
15000 :
20000 rem*************** daten f. sprite1 ****************
20010 data 000,000,000,000,000,000,000,000,000,000,000,000
20020 data 000,000,000,000,000,000,000,000,000,000,000,000
20030 data 000,000,000,003,240,000,004,008,000,012,004,000
20040 data 127,255,254,255,135,255,227,255,143,028,000,112
20050 data 000,000,000,000,000,000,000,000,000,000,000,000
20060 data 000,000,000
20070 rem**********************************************
20080 :
```

In dem „Kollisions-Programm" fahren zwei Autos aufeinander. Die Werte des Sprite-Sprite-Kollisionsregisters sowie des Sprite-Hintergrund-Kollisionsregisters werden ständig angezeigt. Im zweiten Durchlauf sind vier „X" dem Auto in den Weg gelegt. Sobald der Sprite einen dieser Buchstaben berührt, ändert sich das Sprite-Hintergrund-Register.

Auffällig ist, daß das Auto „vor" den Zeichen aber „hinter" dem LKW fährt. Auch das hat seine Ursachen.

B6.11 Prioritäten

Fahren zwei Sprites übereinander, so ist immer der Sprite „vorne", der die niedrigere Sprite-Nummer trägt (0 bis 7), d.h., daß Sprite 0 den Sprite 1 verdecken würde. Ob ein Sprite vor oder hinter einem Zeichen fahren soll, bestimmt das sogenannte „Priorität-Register" (V+27). Hier wird für jeden Sprite (pro Bit ein Sprite) die Priorität festgelegt. Normal befindet sich im Register V+27 der Wert Null. Damit haben alle Sprites Priorität vor dem Hintergrund.

```
V+27 0000 0000  =  0
```

Um ein Sprite hinter den Zeichen fahren zu lassen, braucht nur das entsprechende Bit gesetzt werden:

```
V+27 0000 0001  =  1   Sprite 0 hinter Zeichen
V+27 0001 1100  =  28  Sprite 2+3+4
```

Probieren Sie dies einmal aus, indem Sie Sprite 1 (Auto) hinter die Zeichen legen.

```
POKE V+27, 2          0000 0010
```

Das Auto fährt nun hinter den Zeichen.

Für Spiele sind Sprites wirklich eine schöne Sache. Die Handhabung ist nach anfänglichen Schwierigkeiten doch nicht so kompliziert, wie es den Anschein hatte. Das wesentliche Problem wird in der Bewegungsgeschwindigkeit der Sprites liegen. Werden in einem Spiel z. B. mehrere Sprites gleichzeitig bewegt und es sind viele Abfragen notwendig, so wird es doch recht langsam. Hier bietet nur eine Programmierung in Maschinensprache eine Lösung. Mit den Grundlagen über die Sprite-Handhabung ist auch eine Programmierung in Maschinensprache möglich. Um das Thema Sprite endgültig abzuschließen, fehlen noch die mehrfarbigen Sprites.

B7.0 Multi-Color-Sprite

Jetzt wird es einmal wieder richtig bunt. Im Multi-Color-Modus können Sprites aus insgesamt vier verschiedene Farben bestehen. Die Handhabung von

Multi-Color-Sprites ist identisch mit den normalen Sprites. Ausschließlich die Form und Farbgebung ist etwas anders.

Ein Sprite besteht normal aus 24*21 gesetzten oder nicht gesetzten Punkten. Jedes gesetzte Bit ('1') wird einzeln interpretiert und in der Spritefarbe gesetzt. Betrachtet man die 24*21 Bitmatrix nicht bitweise sondern paarweise, so erhält man eine Matrix von 12*21 Feldern (siehe hierzu auch B5.1)

00 01 10 11 ein Byte zerlegt in Paare

Es sind pro Bitpaar vier Kombination möglich: 00, 01, 10, 11. Jedes Bitpaar kann eine andere Farbe repräsentieren. Die Auflösung halbiert sich in der X-Achse und es werden pro Farbe immer 2 Punkte gesetzt. Als erstes werden die vier Farben bestimmt und in den entsprechenden Registern abgelegt.

00	Hintergrundfarbe V+33 (53281)
10	Spritefarbe V+39 bis V+46 (53287-53294)
01	Sprite-Mehrfarbregister V+37 (53285)
11	Sprite Mehrfarbregister V+38 (53286)

Neu sind die beiden Register V+37 und V+38. In ihnen werden die Farben für die Bitkombinationen „01" und „11" abgelegt. Je nach Kombination der Bitpaare wird die Farbe aus einem der 4 Farbregister geholt. Damit der Computer über das paarweise interpretieren informiert ist, muß ihm mitgeteilt werden, welche Sprites im Multi-Color-Modus interpretiert werden sollen. Das Register V+28 (53276) schaltet einen Sprite bei gesetztem Bit ('1') auf Multi-Color Ausgabe. Bit 0 für Sprite 0 bis Bit 7 für Sprite 7.

V+28 0000 0001	=	0	Sprite 0 auf Multi
V+28 0001 0001	=	17	Sprite 0+4 auf Multi

Damit ist es möglich, normale und Multi-Color-Sprites gleichzeitig auf dem Bildschirm darzustellen, da jedes Sprite einzeln auf Mehrfarbausgabe eingestellt werden kann. Betrachten Sie hierzu einmal das folgende Programm. In den DATA-Zeilen liegen die Daten für eine Fahne (Schwarz/Rot/Gelb). Es sind vier verschiedene Werte vorhanden (0, 170, 85 und 255).

00 00 00 00	=	0	jedes Paar „00". Farbe aus V+33
10 10 10 10	=	170	jedes Paar „10" aus Spritefarbe
01 01 01 01	=	85	jedes Paar „01". Farbe aus V+37
11 11 11 11	=	255	jedes Paar „11". Farbe aus V+38

Das Entwickeln von mehrfarbigen Sprites ist etwas kompliziert, da immer aus einem Bitpaar ein farbiger Punkt erkannt werden muß. Es bedarf schon etwas Übung, um schöne Multi-Color-Sprites zu entwerfen. Es ist noch kein Meister vom Himmel gefallen. Abschließend noch ein Demo-Programm mit einem Multi-Color-Sprite.

```
10 rem************ multicolor sprite ****************
20 print chr$(147): v=53248
30 for i=11*64 to 11*64+62
40 read x: poke i,x
50 next i
60 :
70 poke 2040, 11
80 :
100 poke v+37, 2: rem rot
110 poke v+38, 7: rem gelb
120 poke v+39, 0: rem schwarz
130 :
140 poke v+23, 1: rem y-dehnen
150 poke v+29, 1: rem x-dehnen
160 rem**************** fahnenmast zeichnen **********
170 for i= 0 to 20
180 print tab(25);chr$(165)
190 next
200 for i=0 to 38
210 print chr$(166);
220 next
230 rem*******************************************
240 poke v+28, 1: rem multi sprite0
250 :
260 poke v+21, 1: rem sprite0 an
270 :
280 poke v, 176
290 for x= 160 to 60 step-1
300 poke v+1, x
310 for j=0 to 100: next j
320 next x
330 rem*******************************************
340 :
```

```
1000 rem*************** daten f. sprite0 (mc) **********
1010 data 000,000,000,000,000,000,000,000,000,000,000,000
1020 data 000,000,000,170,170,170,170,170,170,170,170,170
1030 data 085,085,085,085,085,085,085,085,085,255,255,255
1040 data 255,255,255,255,255,255,000,000,000,000,000,000
1050 data 000,000,000,000,000,000,000,000,000,000,000,000
1060 data 000,000,000
1070 rem**********************************************
```

B8.0 Hochauflösende Grafik

Endlich ist es soweit und Sie können ein neues Kapitel der Grafikvielfalt aufschlagen. Hochauflösende Grafik (Hires = HIgh RESolution) bestimmt heute wesentlich das Leistungsmerkmal eines Computers. Die Auflösung (wieviel Punkte pro Fläche) ist das entscheidende Kriterium für die Feinheit einer Grafik. Der Bildschirm des C = 64 besteht aus 320 * 200 Punkten. Das ergibt eine Auflösung von 64000 Punkten (320*200) in der hochauflösenden Grafik ist es möglich, jeden dieser Punkte zu setzen oder zu löschen. Ein gesetzter Punkt entspricht einem gesetzten Bit. Ebenso wie für den „normalen" Bildschirmspeicher muß auch für den hochauflösenden Bildschirm ein „Bildschirmspeicher" (Bit-Map) vorhanden sein und zwar mit einer Größe von 64000 Bit. 64000 Bit sind 8000 Byte (64000/8). Jedes Bit eines Bytes repräsentiert einen bestimmten Punkt auf dem Bildschirm.

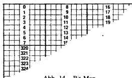


Abb. 14 Bit-Map

Soll der Punkt in der oberen, linken Ecke gesetzt werden, so muß Bit 7 vom Byte 0 der Bit-Map gesetzt werden (1000 0000 = 128).

B8.1 Vorbereitungen für Hires

Einige Fragen müssen vorab geklärt werden. Wo kann überhaupt eine Bit-Map liegen? Die gesamte Grafik wird vom Video-Controler gesteuert, von daher muß die Bit-Map in dem adressierbaren 16 kByte Bereich des Video-Controlers angesiedelt sein (siehe A6.1). Im Einschaltzustand verwaltet der Video-Controler die ersten 16 kByte (0-16383). Die Bit-Map benötigt einen Platz von knapp 8 kByte (8000 Byte). Sie paßt 2 mal in den adressierbaren 16 kByte Bereich des Video-Controlers (0-7999 und 8192-16191). Sofort wird deutlich, daß die Bit-Map nicht in den Bereich 0-7999 gelegt werden darf. Sie würde wichtige Informationen u.a. auf der Zero-Page zerstören (siehe A3.0). Es bleibt nur noch der Bereich von 8192-16191. Mit Bit 3 des Registers 53272 wird bestimmt, ob die Bit-Map in der unteren oder oberen Hälfte des 16 kByte Bereichs liegt.

```
53272 xxxx 0xxx   Bereich    0 - 7999
53272 xxxx 1xxx   Bereich 8192 - 16191
```

Gesetzt wird das Bit mit

```
POKE 53272, PEEK(53272) OR 32
```

und mit

```
POKE 53272, PEEK(53272) AND 247
```

wird es gelöscht. Im Einschaltzustand ist das Bit 3 gelöscht ('0'). Abschließend muß dem Rechner mitgeteilt werden, daß ab sofort im Bit-Map-Modus gearbeitet werden soll. Das Bit 5 im Register 53265 schaltet die hochauflösende Grafik ein.

```
53265 xx0x xxxx   Hires aus
53265 xx1x xxxx   Hires an
```

Jetzt kann endgültig die hochauflösende Grafik ein- und ausgeschaltet werden.

```
10 rem****** einschalten hires ******
20 poke 53272, peek(53272) or 8
30 poke 53265, peek(53265) or 32
40 rem****************************
```

```
50 get a$: if a$ = "" then 50
60 rem****** ausschalten hires ******
70 poke 53272, peek(53272) and 247
80 poke 53265, peek(53265) and 223
90 rem****************************
```

In den Zeilen 20 und 30 ist die Bit-Map auf die zweite Hälfte des 16 kByte Bereichs verlegt und eingeschaltet worden. Nach Drücken einer Taste wird in den normalen Modus umgeschaltet.

Was ist denn nun zu sehen? Ein ziemliches Chaos. Doch immer der Reihe nach. Der Inhalt der Adressen 8192 bis 16191 (die Bit-Map) wird als gesetzte oder gelöschte Punkte dargestellt. Was sich jetzt gerade in diesem Bereich befindet, ist zu sehen. Durch das Löschen jedes Bits der Bit-Map ('0') kann der Bildschirm gelöscht werden.

```
42 for i = 8192 to 16191: poke i,0: next
```

Übrig bleiben einige farbige Felder in der Größe von 8*8 Punkte. Wo kommen die denn nun her? Die Antwort auf diese Frage ist der Farb-Speicher. Wie bei jedem Bildschirm kann die Hintergrundfarbe und die Zeichenfarbe bestimmt werden. Für die Farbgebung der Bit-Map ist aber nicht der Farb-RAM zuständig. Die Farbe wird, so komisch es auch klingt, vom Bildschirmspeicher bestimmt. Aus diesem Grunde ist es auch nicht möglich, Text und Grafik gleichzeitig darzustellen. Jede der 1000 Bildschirmspeicherstellen übernimmt die Farbgebung für ein 8*8 Punkte großes Feld. Es ist die Zeichenfarbe und die Hintergrundfarbe für dieses Feld enthalten. Dabei geben die 4 niederwertigen Bits die Hintergrundfarbe und die 4 hochwertigen Bits die Zeichenfarbe wieder.

0001 0000	Hintergrundfarbe	(0000) = 0 Schwarz
	Zeichenfarbe	(0001) = 1 Weiß
0001 0000	= 16	

Um den gesamten Bildschirm z.B. auf die Hintergrundfarbe „Rot (2)“ und Zeichenfarbe „Weiß (1)“ zu setzen, muß der gesamte „Bildschirmspeicher“ mit

0001 0010 = 18

geladen werden.

```
44 for i = 1024 to 2023: poke i,18: next
```

Endlich ist ein sauberer Bildschirm mit der Hintergrundfarbe „Rot" und der Zeichenfarbe „Weiß" entstanden. Durch Setzen einzelner Bits der Bit-Map erscheint nun ein weißer Punkt.

```
46 poke 8194, 128: rem 1000 0000
```

Ein einzelner Punkt (Bit 7) wird im Byte 3 der Bit-Map gesetzt.

```
48 poke 8199, 255: rem 1111 1111
```

Alle Bits des Byte 8 der Bit-Map sind gesetzt.

Zur Errechnung des richtigen Farbwertes, bestehend aus Hintergrund- und Zeichenfarbe, ist folgende Gleichung gut zu verwenden:

CO = 16*ZF+HF

„HF" = Hintergrundfarbe, „ZF" = Zeichenfarbe und „CO" = zu POKEnder Wert.

B8.2 Verschieben der Bit-Map

Bei der bisher angelegten Bit-Map taucht ein Problem auf. Sie liegt im Bereich 8192-16191 und damit mitten im BASIC-RAM. Ein längeres BASIC-Programm würde sehr bald diesen Bereich erreichen. Der Bereich oberhalb 8191 müßte geschützt werden (siehe A4.5), was zur Folge hätte, daß nur noch 6 kByte für Programme zur Verfügung stehen! Eine Lösung bietet nur das Verschieben des Arbeitsbereichs des Video-Controlers während der Hochauflösung. Es bietet sich der Bereich 16384 bis 32767 an. Die Bit-Map liegt dann von 24576 bis 32767 und der dazugehörige „Farbbildschirm" von 23552 bis 24575. Das Verschieben des Arbeitsbereichs des Video-Controlers ist ausführlich unter A6.1-A6.3 beschrieben.

```
60000 rem*********** hires an ************************
60010 poke 56576,peek(56576) and 252 or 2
60020 poke 53272,peek(53272) or 8
60030 poke 53272,peek(53272) and 15 or 112
60040 poke 648, 92
60050 poke 53265,peek(53265)or32
60060 return
60070 rem*****************************************
60080 :
60100 rem********** hires aus ************************
60110 poke 56576,peek(56576) and 252 or 3
60120 poke 53272, 21
60130 poke 648, 4
60140 poke 53265,peek(53265) and 223
60150 return
60160 rem*****************************************
60170 :
```

Immer wenn die hochauflösende Grafik eingeschaltet wird, wird der Video-Controler nach 16192 umgeschaltet und der Bildschirm wird neu bestimmt. Beim Zurückschalten in den normalen Text-Modus wird der Video-Controler wieder auf den Bereich 0-16192 gelegt und der Bildschirm wieder nach 1024. Somit steht der Bereich 2049 bis 23551 für Programme und Variablen frei zur Verfügung (21 kByte). Der Bereich oberhalb 23551 muß vor dem Überschreiben (speziell der Stringvariablen) geschützt werden.

```
10 rem***************** schützen ab 23551 **************
20 poke 52, 92: poke 56, 92
30 rem*****************************************
```

Nach und nach wird ein Grafikprogramm ab der Zeilennummer 60000 entstehen. Den Anfang machte das Ein- und Ausschalten der hochauflösenden Grafik. Zwei weitere, schon bekannte Teile (Löschen Bit-Map und Farbe setzen) folgen.

```
60200 rem*************** bit-map loeschen **************
60210 for i=24576 to 32767
60220 poke i,0
60230 next i
60240 return
```

```
60250 rem*******************************************
60260 :
60300 rem*************** farbe  setzen ****************
60310 for i=23552 to 24575
60320 poke i,co
60330 next i
60340 return
60350 rem*******************************************
60360 :
```

Probieren Sie die bisherigen Unterprogramme einmal aus.

```
100 gosub 60000: rem hires an
110 gosub 60200: rem bit-map löschen
120 co=1*16+2: rem 2=hintergrundf. 1=zeichenfarbe
130 gosub 60030: rem farbe setzen
140 get a$: if a$ = "" then 140
150 gosub 60100: rem hires aus
160 end
```

Alle Vorbereitungen für das Arbeiten mit hochauflösender Grafik sind getroffen, so daß nun mit dem „Zeichnen" begonnen werden kann.

B9.0 Punkt setzen

Jede Grafik, sei es eine Linie, ein Kreis oder ein Block, besteht immer aus gesetzten Punkten. Die zentrale Routine ist dabei das Setzen eines Punktes innerhalb der Bit-Map. Das eigentliche Problem besteht im Zuordnen eines Koordinatensystems zur Bit-Map.
Die Bit-Map (siehe B8.0) besteht aus 320 Punkten in der X- und 200 in der Y-Achse. Der Null-Punkt beider Achsen befindet sich in der oberen, linken Ecke. Um einen Punkt (z.B. X=100 / Y=70) innerhalb dieses Koordinatensystems zu setzen, muß errechnet werden, in welchem Byte der Bit-Map dieser Punkt (Bit) liegt. Betrachtet man den hochauflösenden Bildschirm ebenfalls aufgeteilt in 25 Zeilen und 40 Spalten à 8 Bytes (25*40*8=8000), muß das Feld (8*8), das Byte innerhalb des Feldes und das entsprechende Bit des Bytes errechnet werden. An einem Beispiel soll der Rechenvorgang erläutert werden.

Als Koordinaten sind X=100 und Y=70 angenommen. Die Zeile, in der sich der Punkt befindet, errechnet sich aus:

```
Zeile  = INT  ( Y/8)
       = INT  (70/8)
       = 8
```

In der 8. Zeile befindet sich der Punkt. Jede Zeile besitzt 320 Byte (40*8).

```
Feld   = 320 * Zeile
       = 320 * 8
       = 2560
```

Das 8*8-Feld beginnt mit Byte 2560 der Bit-Map. Die Y-Achse liegt in einem der Bytes 2560-2567. Das genaue Byte ergibt sich aus dem Rest des Zeilenwertes Y/8. Multipilziert mit 8 ergibt sich das Byte innerhalb des Feldes.

```
Rest   = (Y/8-INT(Y/8))*8
       = (8.75-8)*8
       = 0.75*8
       = 6
```

Das 6. Byte innerhalb des Feldes ist das Byte, welches der Y-Reihe entspricht.

```
Reihe  = Feld+Rest
       = 2560+6
       = 2566
```

Das genaue Byte, in dem sich der Punkt befindet, wird von der X-Koordinate bestimmt. Es wird das Feld der X-Achse ausgerechnet und mit 8 (8 Byte pro Feld) multipliziert.

```
Byte   = INT(X/8)*8+Reihe
       = INT(100/8)*8+2566
       = 12*8+2566
       = 2662
```

Der gesuchte Punkt liegt innerhalb des Bytes 2662 der Bit-Map. Das genaue Bit ergibt sich wieder aus dem Rest des Spaltenwerts X/8.

```
Bit   = 7-(X/8-INT(X/8))*8
      = 7-(100/8-INT(100/8))*8
      = 7-(12.5-12)*8
      = 7-4
      = 3
```

Das 3. Bit innerhalb des Bytes muß gesetzt werden. Der Wert ist von 7 subtrahiert worden, da das Bit 0 immer „rechts" steht.

> Das hier errechnete Byte ist das 2662. Byte der Bit-Map. Die Bit-Map beginnt bei 24576, so daß das Byte der Bit-Map in Adresse 24576 + 2662 = 27238 liegt.

Zusammengefaßt entstehen folgende Gleichungen:

```
BYTE  = 24576+INT(Y/8)*320+(Y/8-INT(Y/8))*8+INT(X/8)*8
BIT   = 7-INT((X/8-INT(X/8))*8)
```

Die Gleichung ließe sich mathematisch noch vereinfachen, dieses würde allerdings das Verständnis nicht erleichtern. Doch nun zum Programmteil „Punkt-setzen":

```
60400 rem***************** punkt setzen *****************
60410 if x<0 or x>319 or y<0 or y>199 then 60450
60420 by=24576+int(y/8)*320+(y/8-int(y/8))*8+int(x/8)*8
60430 bi=7-int((x/8-int(x/8))*8)
60440 poke by, peek(by) or (2^bi)
60450 return
60460 rem*********************************************
60470 :
```

In den Zeilen 60420 und 60430 ist die Umrechnung in Byte und Bit wiederzufinden. Damit keine Fehler eintreten , wenn die X- und Y-Koordinaten außerhalb des Bildschirmes liegen, wird in Zeile 60410 der Bereich geprüft. Die Zeile 60440 setzt das entsprechende Bit, wobei der sonstige Inhalt des Bytes nicht verändert wird.

Die erste Grafik kann entstehen.

```
100 gosub 60000: gosub 60200
110 co = 1*16+0: gosub 60300
120 for x=0 to 319
130 y=99+sin(x*/180)*80
140 gosub 60400
150 next
160 get a$: if a$ = "" then 160
170 gosub 60100
180 end
```

Es entsteht eine Sinus-Kurve über die gesamte X-Achse (0-319). Der Sinus-Wert wird in Bogenmaß umgewandelt (/180), so daß Werte zwischen -1 und +1 entstehen. Durch das Multiplizieren mit 80 liegt der Sinus-Wert zwischen -80 und +80. Der Nullpunkt der Y-Achse ist auf 99 (Mitte Bildschirm) gelegt. Dadurch liegen die Y-Koordinaten endgültig zwischen 19 (99-80) und 179 (99+80) und nutzen den Bildschirm gut aus.

Das Zeichnen der Sinus-Kurve dauert mit Löschen des Bildschirms und Setzen der Farben gut 90 Sekunden. Das ist schon sehr lang. Doch hier liegen halt die zeitlichen Grenzen von BASIC-Programmen. Besonders an den Nerven zerrt das Warten, bis die Bit-Map gelöscht worden ist, weil hier so gut wie nichts passiert. Für alle diejenigen von Ihnen, denen es ebenfalls zu lange dauert, biete ich ein kurzes Maschinenprogramm zum Löschen der Bit-Map und Setzen der Farbe an. Wenn Sie wollen, bauen sie es sich in das Programm ein.

```
35 :
40 rem************** maschinen programm ****************
41 data 169,000,133,250,169,096,133,251
42 data 169,000,162,000,160,000,145,250
43 data 200,208,251,230,251,232,224,032
44 data 208,244,096,169,000,133,250,169
45 data 092,133,251,173,012,003,162,000
46 data 160,000,145,250,200,208,251,230
47 data 251,232,224,004,208,244,096
48 for i = 49152 to 49206
49 read q: poke i,q: next i
50 rem************************************************
55 :
```

Ändern Sie dann noch die Programmteile „Bit-Map" löschen und „Farbe setzen" entsprechend.

```
60200 rem**************** bit-map loeschen ***************
60210 sys 49152
60220 return
60230 rem************************************************
60250 :
60300 rem**************** farbe  setzen *****************
60310 poke 780,co
60320 sys 49179
60330 return
60340 rem************************************************
60350 :
```

B10.0 Linie zeichnen

Der Grundstein für die hochauflösende Grafik ist mit dem „Punktsetzen" gelegt. Alle grafischen Formen (Kreise, Linien, Flächen etc) beruhen auf dem Setzen von Punkten an bestimmten Orten. Alle Formem lassen sich in zwei wesentliche Bestandteile zerlegen; der Geraden und dem Kreis (Ellipse) oder Kreisteilen (Bögen). Eine besondere Schwierigkeit bildet das Zeichnen von Geraden. In zwei (Sonder-) Fällen ist das Zeichnen von Geraden recht einfach: der Senkrechten und der Waagerechten.

```
100 y=100: for x=0 to 319: gosub 60400: next
110 x=160: for y=0 to 199: gosub 60400: next
120 get a$: if a$ = "" then 120
130 gosub 60100: end
```

Bei der Waagerechten ist die Y-Achse auf 100 (Mitte) gesetzt und die X-Koordinate zwischen 0 und 319 verändert worden. Die Senkrechte entsteht durch das Ändern der Y-Koordinate. Probleme tauchen auf, wenn z.B Diagonalen oder Geraden mit Steigung gezeichnet werden sollen. Jetzt hilft Ihnen nur noch die Mathematik weiter. Die Berechnung einer Geraden ist in der Mathematik ein altes und bekanntes Thema der analytischen Geometrie. Eine Gerade (Linie) besteht aus vielen hintereinander gesetzten Punkten, die eine geschlossene Linie ergeben. Jeder dieser einzelnen Punkte läßt sich mit der „Geradengleichung" errechnen. Eine Gerade ist allgemein durch einen Punkt und

ihre Richtung bestimmt. Die Richtung ist der Winkel zu einer gedachten X-Achse. Eine Gerade ist meistens (jedenfalls in der grafischen Anwendung) begrenzt, sie beginnt und endet an einem bestimmten Punkt. Die Koordinate, an der eine Gerade beginnt, sei mit X1;Y1 und ihr Endpunkt mit X2;Y2 bezeichnet. Mit der „Zweipunktegleichung" ist jeder Punkt der Gerade zu errechnen.

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

Durch Umstellen der Zweipunktegleichung nach X und Y (die Koordinaten, wo die Punkte einer Geraden liegen) erhalten Sie folgende Gleichungen:

X = (X2-X1)/(Y2-Y1)*(Y-Y1)+X1

Y = (Y2-Y1)/(X2-X1)*(X-X1)+Y1

Bei der Umsetzung der beiden Gleichungen in das Grafikprogramm tritt eine Schwierigkeit auf: soll die X- oder die Y-Koordinate berechnet werden? Dazu eine Überlegung vorab. Bei der Waagerechten, die im Demoprogramm gezeichnet wurde, ist die X-Koordinate verändert worden. Daraus folgt, daß bei einer Waagrechten die X-Koordinate und bei einer Senkrechten die Y-Koordinate berechnet werden müßte. Bei einer (exakten) Diagonalen ist es egal, ob die X- oder die Y-Koordinate berechnet wird, es führt zum gleichen Ergebnis. Stellen Sie sich bitte vor, es wird eine Gerade mit einer sehr geringen Steigung (z.B. 5 Grad) von X1 = 0 nach X2 = 319 gezeichnet. Der jeweilige Y-Wert verändert sich nur sehr wenig (0-10), der X-Wert erheblich (0-319). Es ist also notwendig, den Y-Wert in Abhängigkeit zum X-Wert auszurechnen. Bei einer sehr steilen Geraden (z.B. 80 Grad) muß der X-Wert bestimmt werden. Die Grenze liegt genau bei 45 Grad. Alle Geraden mit einer Steigung unter 45 Grad werden aus X-Werten und Steigungen über 45 Grad aus Y-Werten errechnet. Ebenso verhält es sich bei Steigungen über oder unter 135 Grad (90+45). Die Steigung einer Geraden ergibt sich aus dem Verhältnis

TAN(Steigung) = (Y2-Y1)/(X2-X1)

Der Tangens von 45 Grad ergibt den Wert 1. Ist der Winkel kleiner als 45 Grad, geht der Tangenswert gegen Null (TAN(0)=0). Über 45 Grad geht der Tangenswert gegen Unendlich. Mit dem Tangenswert kann selektiert werden, ob die X- oder Y-Koordinate errechnet werden muß. Doch nun zum eigentlichen Programmteil.

```
60500 rem************* linie zeichnen *****************
60510 ne=1
60520 if x1=x2 then 60560
60530 if y1=y2 then 60620
60540 if abs(y2-y1)/abs(x2-x1) <1 then 60620
60550 :
60560 if y1>y2 then ne=-1
60570 for y =y1 to y2 step ne
60580 x=(x2-x1)/(y2-y1)*(y-y1)+x1
60590 gosub 60400
60600 next y
60610 return
60620 if x1>x2 then ne=-1
60630 for x =x1 to x2 step ne
60640 y=(y2-y1)/(x2-x1)*(x-x1)+y1
60650 gosub 60400
60660 next x
60670 return
60680 rem******************************************
60690 :
```

Das Unterprogramm besteht im Wesentlichen aus zwei Teilen: dem Errechnen des Punktes in Abhängigkeit von der Y- (Zeile 60560) und der X-Koordinate (Zeile 60620). In Zeile 60520 und Zeile 60530 wird geprüft, ob es sich um eine Waagerechte oder Senkrechte handelt, da in diesen Fällen die Steigung (Zeile 60540) nicht berechnet werden muß (es wird auch eine Division durch Null verhindert (X1 = X2)). Ist die Steigung (Zeile 60540) kleiner als 45 Grad (1), wird die Y-Koordinate in Abhängigkeit zur X-Achse berechnet. Die Schrittweite (NE) beträgt immer 1. In dem Fall, wo X1 größer als X2 oder Y1 größer als Y2 ist, wird die Schrittweite negiert (-1). Ist ein Punkt berechnet, erfolgt ein Sprung in das Unterprogramm zum Setzen eines Punktes (GOSUB 60400).

Auch zu diesem Unterprogramm wieder ein kleines Testprogramm:

```
100 x1=0: y1=0: x2=319: y2=199: gosub 60500
110 x1=0: y1=199: x2=319: y2=0: gosub 60500
120 x1=160: y1=100: x2=0: y2=100: gosub 60500
130 x1=160: y1=0: x2=160: y2=199: gosub 60500
140 get a$: if a$ = "" then 140
150 gosub 60100
160 end
```

Ohne Mathematik geht es in der Grafik leider nicht. Ich hoffe, daß die knappe Erläuterung über eine Gerade nicht gar zu trocken oder unverständlich war. Das Zeichnen einer Geraden ist nun einmal „nicht ohne".
Mit dem Unterprogramm zum Zeichnen von Geraden lassen sich schon nützliche Grafikausgaben verwirklichen. Hierzu ein kleines Programm.
Bis zu 20 Werte können grafisch als Säulen dargestellt werden. Der größte Wert aller Daten bestimmt das Diagramm und alle anderen Werte werden in dieses Verhältnis umgerechnet. Der größte Wert befindet sich in der Variablen WE(0). In den Variablen WN(...) befinden sich die Y-Koordinaten für den Bildschirm.

```
100 print chr$(147)
110 print " -- grafische darstellung von werten --"
120 print
130 input " * wieviele werte (bis 20) :";an
140 if an<1 or an>20 then print chr$(145);: goto 130
150 print
160 dim we(an),wn(an)
170 for i= 1 to an
180 print " * wert";i;" :";
190 input we(i)
200 su=su+we(i)
210 if we(i) > we(0) then we(0)=we(i)
220 next i
230 print chr$(147)
240 print " * eingaben :";an
250 print " * summe    :";su
260 print " * max. wert:";we(0)
270 print:print
280 for i=1 to an
290 wn(i)=we(i)*199/we(0)
300 next i
310 print " * grafik ...... bitte taste !!"
320 get a$: if a$ = "" then 320
330 gosub 60000: rem hires an
340 gosub 60200: rem loeschen
350 co=1*16+0: gosub 60300: rem farbe
360 for i=1 to an
370 x1=(i*16)-16:x2=x1:y2=199
```

```
380 y1=199-wn(i):gosub 60500
390 x1=x1+10:x2=x1:gosub 60500
400 x1=x1-10:y2=y1:gosub 60500
410 next
420 x1=0:y1=199:x2=319:y2=199:gosub 60500
430 get a$: if a$ = "" then 430
440 gosub 60100
450 end
```

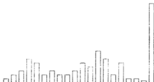


Abb. 15 Säulengrafik (Ausdruck)

Ein mögliches Ergebnis zeigt die obige Zeichnung. Eine Schwierigkeit taucht beim Anpassen der Werte an das Koordinatensystem des Bildschirms auf. Normalerweise befindet sich bei einem Koordinatensysten der Nullpunkt für die X- und Y-Achse in der unteren, linken Ecke. Beim „Bildschirmkoordinatensystem" liegt der Nullpunkt hingegen in der oberen, linken Ecke, so daß eine Anpassung notwendig wird. Besonders bei Funktionsdarstellungen ist es wichtig, ein gebräuchliches Koordinatensystem zu erhalten.

Schon in der Zeile 290 wurde die Y-Koordinate vom gedachten an das Bildschirmkoordinatensystem angepaßt. Da die Y-Achse ihren Nullpunkt unten links erhalten soll, wurde der errechnete Wert in Zeile 380 von 199 (der neue Nullpunkt) abgezogen. Allgemein ist es mit den Gleichungen

X=(XX-XA)/(XE-XA)*319
Y=(YE-YY)/(YE-YA)*199

möglich, Werte in ein anderes Koordinatensystem zu „transformieren". XA;XE ist die Unter- und Obergrenze der X-Achse des neuen Systems und

YA;YE die der Y-Achse. Beim Zeichnen einer Sinus-Kurve liegt der Wert für die Y-Koordinaten zwischen -1 und 1. Es bietet sich ein System mit den Ausmaßen von YA=-1;YE=1 an. Die X-Achse kann zwischen XA=0;XE=10 liegen. XX und YY sind die X- und Y-Werte des neuen Systems, X und Y sind in das Bildschirmkoordinatensystem umgerechnet.

```
100 print chr$(147)
110 input " * koordinaten (xa,xe):";  ,xe
120 input " * koordinaten (ya,ye):";ya,ye
130 gosub 60000: gosub 60200
140 co=1*16+0: gosub 60300
150 for xx = xa to xe step (xe-xa)/319
160 x=(xx-xa)/(xe-xa)*319
170 yy=sin(xx)
180 y=(ye-yy)/(ye-ya)*199
190 gosub 60400
200 next xx
210 get a$: if a$ = "" then 210
220 gosub 60100
230 end
```

Jedes denkbare Koordinatensystem ist so darstellbar. Bitte beachten Sie bei der Festlegung des Koordinatensystems, daß die Funktion auch in diesem Bereich verlaufen muß, um sie sichtbar zu machen. Auch Geraden können so i in anderes System transformiert werden.

B11.0 Die Ellipse

Der Kreis ist eine Sonderform der Ellipse: der X-Radius ist gleich dem Y-Radius (XR=YR). Bei einer Ellipse ist der X- und Y-Radius unterschiedlich. Der Ort, wo sich eine Ellipse im einem System befindet, wird durch seinen Mittelpunkt (XM;YM) bestimmt. Die X- und Y-Koordinate für einen Ellipsenpunkt berechnet sich aus einer Sinus- und einer Cosinus-Funktion.

X=XR*COS(Winkel)
Y=YR*SIN(Winkel)

Der Winkel gibt die Gradzahl (0-360 Grad) des zu berechnenden Punktes an. Um eine geschlossene Ellipse zu erreichen, muß der Winkel von 0 bis 360 Grad verändert werden. Der Ort einer Ellipse ist durch XM;YM bestimmt und wird durch

X=XM+XR*COS(Winkel)
Y=YM+YR*SIN(Winkel)

berücksichtigt. Ein Ellipsenteil bzw. Kreisteil (z.B. von 45 Grad nach 135 Grad) wird erzielt, indem nur diese Winkel berücksichtigt und als Punkte dargestellt werden.

```
60700 rem********** ellipse  zeichnen ****************
60710 for wi = wa to we
60720 x=xm+xr*cos(wi*π/180)
60730 y=ym+yr*sin(wi*π/180)
60740 gosub 60400
60750 next wi
60760 return
60770 rem*****************************************
```

Als Übergabeparameter sind die bekannten Variablen benutzt worden:

XM;YM	=	Mittelpunkt
WA;WE	=	Winkelanfang; Winkelende
XR;YR	=	Radien

Probieren Sie den neuen Programmteil aus:

```
100 gosub 60000: gosub 60200
110 co=1*16+0: gosub 60300
130 xm=160: ym=100:xr=60: yr=60: wa=0: we=360: gosub 60700
140 xm=50: ym=70: xr=100: yr=50: wa=0: we=90 : gosub 60700
150 get a$: if a$ = "" then 150
160 gosub 60100
170 end
```

Ein Kreis (Zeile 130:xr=yr) und ein Kreisteil (90 Grad) ist entstanden. Ist der Kreis auf Ihrem Bildschirm nicht ganz rund, so liegt es an Ihrem Fernseher/Monitor. Oft ist es möglich, die „vertikale Frequenz" an der Rückseite eines Gerätes zu ändern.

Tortengrafik ist heute in aller Munde. Ohne großen Aufwand läßt sich mit den Grafikunterprogrammen eine einfache Tortengrafik erstellen. Versuchen Sie es einmal selber. Hier eine mögliche Lösung.

```
100 print chr$(147): dim we(20),wn(20)
110 input " * wieviele werte (bis 20):";an
120 if an<1 or an>10 then print chr$(145);: goto 110
130 print: for i=1 to an
140 print " * wert";i;" :";
150 input we(i)
160 if we(i)<=0 then print chr$(145);: goto 140
170 ss=ss+we(i)
180 next i
190 print chr$(147)
200 print " * summe :";ss
210 for i = 1 to an
220 wn(i)=(we(i)*360/ss)*π/180
230 :
240 next i
250 print: print
260 print " * grafik ....... bitte taste!!"
270 get a$: if a$ = "" then 270
280 gosub 60000: gosub 60200
290 co=2*16+0: gosub 60300
300 :
310 xm=160:ym=100:xr=100:yr=40:wa=0:we=360:gosub 60700
320 ym=120:we=180:gosub 60700
330 x1=60: y1=100:x2=60:y2=120:gosub 60500
340 x1=260:x2=260:gosub 60500
350 :
360 w1=0:x1=160:y1=100
370 x2=260:y2=100:gosub 60500
380 for i = 1 to an-1
390 w1=w1+wn(i)
```

```
400 x2=160+100*cos(w1)
410 y2=100+40*sin(w1)
420 gosub 60500
430 if w1< then x1=x2:y1=y2+20:gosub60500:x1=160:y1=100
440 next i
450 get a$: if a$ = "" then 450
460 gosub 60100
470 end
```

Abb. 16 Tortengrafik (Ausdruck)

Bis zu 20 Werte können in der Tortengrafik angezeigt werden. In den Zeilen 210 bis 240 werden alle Werte transformiert und der zugehörige Kreisanteil berechnet. Zeile 300 bis 350 liefert die „ungeteilte" Torte. Als Ausgangspunkt zeichnet Zeile 360-370 die 0-Grad Linie. Nacheinander werden alle Winkel in der Hilfsvariable „w1" addiert und geben den Winkel des Tortenstücks an, der in Zeile 400-410 in X- und Y-Koordinaten umgerechnet wird. Interessant ist die Zeile 430. Ist der Winkel kleiner 180 Grad (=3.14152), kann an der Stirnseite der Torte noch ein senkrechter Strich gezogen werden und die Grafik wirkt somit plastischer.

Das Grafikpaket ließe sich noch erheblich weiter ausbauen, z.B. Unterprogramme für Rechtecke, Dreiecke, Blöcke etc. Alle weiteren Formen beziehen sich jedoch ausschließlich auf die vorhandenen Unterprogramme. Ein Rechteck, Dreieck oder Vieleck besteht aus Linien. Ein Block (gefülltes Rechteck) ebenfalls nur aus Linien. Benötigen Sie spezielle Formen, dann schreiben Sie ein Unterprogramm und bauen es in das Grafikpaket ein.

B12.0 Zeichen in hochauflösender Grafik

Die schönste Tortengrafik nützt nicht viel, wenn aus ihr nicht hervorgeht, um welche Werte es sich handelt (Namen) oder ie groß ihr tatsächlicher Wert ist.

Doch leider ist es normalweise nicht möglich, hochauflösenc :Grafik und Text zu mischen. Der Bildschirmspeicher wird bei der hochauflö enden Grafik als Farbspeicher benutzt, so daß hier keine „Zeichen" mehr abg legt werden können bzw. nur eine Änderung der Farbe ergeben (siehe B8.1 Um in Hires ein Zeichen (Buchstaben) auf den Bildschirm zu bringen, müßt er in Punkte zerlegt und jeder dieser Punkte in die Bit-Map gesetzt werden. I as erscheint sehr kompliziert, ist aber doch recht einfach. Überlegen Sie noch einmal, wie Zeichen im Zeichen-ROM abgelegt sind: Jedes Zeichen ist in 8 Byte zerlegt und jedes gesetzte Bit ('1') eines Bytes repräsentiert einen gesetz en Punkt auf dem Bildschirm (siehe B3.0). Die 8 Byte eines Zeichens brauche nun nur aus dem Zeichen-ROM gelesen und in entsprechende Bytes der Bit-M ap gelegt zu werden. Das Verfahren ist wirklich denkbar einfach. Natürlich ist es nicht möglich, Zeichen direkt aus dem Zeichen-ROM zu lesen, desh alb muß der Zeichensatz oder die Zeichen, die verwendet werden sollen in einen RAM-Bereich kopiert werden. Dabei ist es nicht notwendig, den g samten Zeichenvorrat (512 Zeichen) zu kopieren, da die gesamten Grafikze chen in der hochauflösenden Grafik überflüssig sind. Im folgenden Beispie wird sich auf die ersten 64 Zeichen des Zeichen-ROMs beschränkt. Dies ist das Alphabet, die Zahlen und alle Sonderzeichen. Die 512 Bytes (64*8) mi ssen in einen geschützten RAM-Bereich kopiert werden. Es bieten sich mel rere Plätze an, so z.B. der Kassettenpuffer oder der 4 kByte Bereich ab 4915 . Gewählt wurde der Bereich 50000 bis 50511. Das Kopieren des Zeichen-RC Ms ist ausführlich unter B4.0 beschrieben.

```
60800 rem************* 64 zeichen kopieren **** *********
60810 poke 56334, peek(56334) and 254
60820 poke 1, peek(1) and 251
60830 for i = 0 to 511
60840 poke (50000+i), peek(53248+i)
60850 next i
60860 poke 1, peek(1) or 4
60870 poke 56334, peek(56334) or 1
60880 return
60890 rem************************************ *********
60895 :
```

Die ersten 64 Zeichen (0-63) befinden sich nun in den A lressen 50000 bis 50511 (512 Byte). Als erstes Zeichen liegt der „Klamme affe" (Byte 0-7), dann das „A" (Byte 8-15) bis zum „?" im RAM-Speicher Das „Zeichnen"

der Zeichen ist jetzt sehr einfach: um das „A" in die obere, linke Ecke der Bit-Map zu zeichnen, müssen die Bytes 0-7 der Bit-Map mit den Bytes 50008 - 50015 geladen werden.

```
100 gosub 60800: gosub 60000: gosub 60200
110 co=5*16+0: gosub 60300
120 for i = 0 to 7
130 poke 24576+i, peek(50008+i)
140 next i
150 get a$: if a$ = "" then 150
160 gosub 60100
170 end
```

Es ist eine einfache Lösung des Problems. Alle Zeichen können in die Bit-Map wie in den normalen Bildschirm geladen werden. Aus Vereinfachungsgründen können die Zeichen nur in diese 8*8-Felder gezeichnet werden. Es stehen die bekannten 40 Spalten und 25 Zeilen zur Verfügung. Das nachfolgende Programm versetzt Sie in die Lage, einen beliebigen Text in die Bit-Map zu laden und damit zu zeichnen.

```
60900 rem************** zeichen ausgeben **************
60910 if sx<0 or sx>39 or sy<0 or sy>24 then return
60920 s3=0:for s1 = 1 to len (s$)
60930 bc=asc(mid$(s$,s1,1))
60940 if bc>95 or bc<32 then 61010
60950 if bc<64 then 60970
60960 bc=bc-64
60970 for s2=50000+bc*8 to 50000+bc*8+7
60980 poke (24576+(320*sy+8*sx)+s3), peek(s2)
60990 s3=s3+1
61000 next s2
61010 next s1
61020 return
61030 rem*****************************************
60940 :
```

In der Stringvariable „S$" ist der Text enthalten, der ab der Zeile „SY" und Spalte „SX" in die Bit-Map gezeichnet werden soll. Der Text ist in „S$" als ASCII-Zeichen abgelegt, die Reihenfolge im neuen „Zeichensatz" ist allerdings der Bildschirmcode (siehe B3.0). In den Zeilen 60950 und 60960 wird

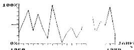
vom ASCII- in den Bildschirmcode umgerechnet. Ist ein Zeichen nicht zeichenbar (Bildschirmcode größer als 64) folgt das nächste Zeichen von S$ (Zeile 60940). Die eigentlich Umrechnung findet in den Zeilen 60970 bis 61010 statt. Aus der Zeile/Spalte wird das „Startbyte" ermittelt, ab dem die 8 Byte untereinander abgelegt werden. Auch hierzu ein „Demo":

```
100 gosub 60800
110 print chr$(147)
120 input " * text   :"; S$
130 input " * zeile  :"; sy
140 input " * spalte :"; sx
150 gosub 60000: gosub 60200
160 co=5*16+0: gosub 60300
170 gosub 60900
180 get a$: if a$ = "" then 180
190 gosub 60100
200 end
```

Somit ist auch eine Bezeichnung (Text) innerhalb der hochauflösenden Grafik möglich. Abschließend möchte ich Ihnen eine mögliche Anwendung zeigen, in der Grafik und „Text" gemischt wird.

```
100 print chr$(147)
110 gosub 60800: rem lade zeichen
120 print: print
130 dim we(21)
140 for i = 1 to 21
150 we(i)=int(rnd(0)*100)
160 next i
200 print: print
210 print " * grafik ......... bitte taste !!"
220 get a$: if a$ = "" then 220
230 gosub 60200: rem hires loeschen
240 co=5*16+0: gosub 60300: rem farbe
250 gosub 60000: rem hires an
260 x1=50:y1=100:x2=250:y2=100:gosub 60500
270 x2=50:y2=0:gosub 60500
280 s$="100%":sy=0:sx=2:gosub 60900
290 s$="0%":sy=12:sx=4:gosub 60900
```

```
300 s$="jahre":sy=12:sx=32:gosub 60900
310 s$="1960":sy=14:sx=4:gosub 60900
320 s$="1980":sy=14:sx=29:gosub 60900
330 s$="umsatz der letzten 20 jahre"
340 sy=20:sx=6:gosub 60900
350 x1=50:for i = 1 to 20
360 x2=50+i*10
370 y1=100-we(i):y2=100-we(i+1)
380 gosub 60500:x1=x2
390 next
400 y1=100:gosub60500
410 for i=1 to 21
420 ds=ds+we(1)
430 next
440 x1=35:y1=100-ds/20:x2=50:y2=y1:gosub 60500
450 y1=100:y2=104
460 for x1=50 to 250 step 10
470 x2=x1:gosub 60500
480 next
490 x1=46:x2=50
500 for y1=0 to 100 step 10
510 y2=y1:gosub60500
520 next
530 s$="menue bitte eine taste !!"
540 sy=22:sx=7:gosub60900
550 get a$: if a$= "" then 550
560 gosub 60100: print chr$(147)
570 print " * grafik .......... (g)"
580 print " * ende ............ (e)"
590 print: print
600 print " * bitte waehlen sie ..."
610 get a$: if a$ = "" then 610
620 if a$="e" then end
630 if a$="g" then gosub 60000: goto 550
640 goto 610
```



UMSATZ DER LETZTEN 20 JAHRE
MENUE BITTE EINE TASTE !!

Abb. 17 Umsatz (Ausdruck)

B13.0 Das Grafikpaket

An dieser Stelle möchte ich noch einmal das gesamte Grafikprogramm zusammengefaßt auflisten. Sie können es immer dann laden, wenn Sie Grafik programmieren wollen.

```
10 rem*************** schuetzen ab 23551 **************
20 poke 52, 92: poke 56, 92
30 rem**********************************************
35 :
40 rem************** maschinen programm ***************
41 data 169,000,133,250,169,096,133,251
42 data 169,000,162,000,160,000,145,250
43 data 200,208,251,230,251,232,224,032
44 data 208,244,096,169,000,133,250,169
45 data 092,133,251,173,012,003,162,000
46 data 160,000,145,250,200,208,251,230
47 data 251,232,224,004,208,244,096
48 for i = 49152 to 49206
49 read q: poke i,q: next i
50 rem**********************************************
55 :
60000 rem************** hires an *********************
60010 poke 56576,peek(56576) and 252 or 2
```

```
60020 poke 53272,peek(53272) or 8
60030 poke 53272,peek(53272) and 15 or 112
60040 poke 648, 92
60050 poke 53265,peek(53265)or32
60060 return
60070 rem********************************************
60080 :
60100 rem*************** hires aus *******************
60110 poke 56576,peek(56576) and 252 or 3
60120 poke 53272, 21
60130 poke 648, 4
60140 poke 53265,peek(53265) and 223
60150 return
60160 rem********************************************
60170 :
60200 rem*************** bit-map loeschen *************
60210 sys 49152
60220 return
60230 rem********************************************
60250 :
60300 rem*************** farbe  setzen ****************
60310 poke 780,co
60320 sys 49179
60330 return
60340 rem********************************************
60350 :
60400 rem**************** punkt setzen ****************
60410 if x<0 or x>319 or y<0 or y>199 then 60450
60420 by=24576+int(y/8)*320+(y/8-int(y/8))*8+int(x/8)*8
60430 bi=7-int((x/8-int(x/8))*8)
60440 poke by, peek(by) or (2^bi)
60450 return
60460 rem********************************************
60500 rem************* linie zeichnen *****************
60510 ne=1
60520 if x1=x2 then 60560
60530 if y1=y2 then 60620
60540 if abs(y2-y1)/abs(x2-x1) <1 then 60620
60550 :
```

```
60560 if y1>y2 then ne=-1
60570 for y =y1 to y2 step ne
60580 x=(x2-x1)/(y2-y1)*(y-y1)+x1
60590 gosub 60400
60600 next y
60610 return
60620 if x1>x2 then ne=-1
60630 for x =x1 to x2 step ne
60640 y=(y2-y1)/(x2-x1)*(x-x1)+y1
60650 gosub 60400
60660 next x
60670 return
60680 :
60700 rem********** ellipse  zeichnen *****************
60710 for wi = wa to we
60720 x=xm+xr*cos(wi*π/180)
60730 y=ym+yr*sin(wi*π/180)
60740 gosub 60400
60750 next wi
60760 return
60770 rem*******************************************
60780 :
60800 rem************ 64 zeichen kopieren ************
60810 poke 56334, peek(56334) and 254
60820 poke 1, peek(1) and 251
60830 for i = 0 to 511
60840 poke (50000+i), peek(53248+i)
60850 next i
60860 poke 1, peek(1) or 4
60870 poke 56334, peek(56334) or 1
60880 return
60890 rem*******************************************
60895 :
60900 rem************** zeichen ausgeben **************
60910 if sx<0 or sx>39 or sy<0 or sy>24 then return
60920 s3=0:for s1 = 1 to len (s$)
60930 bc=asc(mid$(s$,s1,1))
60940 if bc>95 or bc<32 then 61010
60950 if bc<64 then 60970
```

```
60960 bc=bc-64
60970 for s2=50000+bc*8 to 50000+bc*8+7
60980 poke (24576+(320*sy+8*sx)+s3), peek(s2)
60990 s3=s3+1
61000 next s2
61010 next s1
61020 return
61030 rem**********************************************
```

Hires an	gosub 60000
Hires aus	gosub 60100
Bit-Map löschen	gosub 60200
Farbe setzen	gosub 60300
	co = Farbe
Punkt setzen	gosub 60400
	x = X-Koordinate (0-319)
	y = Y-Koordinate (0-199)
Linie zeichnen	gosub 60500
	x1 = X-Anfangskoordinate
	y1 = Y-Anfangskoordinate
	x2 = X-Endkoordinate
	y2 = Y-Endkoordinate
Ellipse zeichnen	gosub 60700
	xm = X-Mittelpunkt
	ym = Y-Mittelpunkt
	xr = X-Radius
	yr = Y-Radius
	wa = Winkelanfang
	we = Winkelende
Zeichen kopieren	gosub 60800
Zeichen ausgeben	gosub 60900
	sx = Spalte (0-39)
	sy = Zeile (0-24)
	s$ = Text

B14.0 Punkt löschen

Bisher wurde ausschließlich das Zeichnen (Setzen) eines Punktes behandelt. Die Berechnungsgrundlage für das Löschen eines Punktes ist identisch mit dem des Setzens, das Bit muß nur gelöscht ('0') werden. Das Löschen ist mit folgender AND-Verknüpfung möglich:

```
60440 poke by, peek(by) and (255-2^bi)
```

Ingesamt bietet die hochauflösende Grafik des C=64 viele und interessante Möglichkeiten. Die Auflösung von 64000 Punkten ist durchaus akzeptabel und für den Heim- und Hobbyprogrammierer mehr als ausreichend. Das Hauptproblem ist die Verarbeitungsgeschwindigkeit in BASIC. Bei großen und umfangreichen Grafiken verstreichen schon einige Minuten, bis die Grafik fertiggestellt ist. Hier hilft nur eine Programmierung in Maschinensprache.

MUSIK

C1.0 Drei Stimmen und vieles mehr!

Eines der wesentlichen Merkmale des C=64 ist sein hervorragendes musikalisches Talent. Der eingebaute SID-Chip (Sound Interface Divice) hat die Regelmöglichkeiten eines kleinen Synthesizers. Der SID besitzt 3 unabhängige Oszillatoren (Stimmen), die über Filter „verfremdet" werden können. Leider ist die Programmierung von Musik (Töne überhaupt) sehr mühselig, da das COMMODORE BASIC über keinerlei „Musik"-Befehle verfügt.

Der SID-Chip belegt die Adressen ab 54272 des C=64. In den Adressen 54272 bis 54296 liegen alle wichtigen Register des Soundchips. Die Register haben teilweise für alle Stimmen eine gemeinsame Funktion. Andere Register sind jeweils nur für eine Stimme zuständig. 4 verschiedene Schwingungsarten stehen jeder Stimme zur Verfügung (Dreieck, Sägezahn, Rechteck, Rauschen), die schon den Charakter eines Tones bestimmen (hart,weich...). Weiterhin läßt sich der „Lautstärkeverlauf" mit einem Hüllkurvengenerator beeinflussen (wieder für jede Stimme). Mit den vorhandenen 3 Filtern (Tief-, Hoch- und Bandpass) und 2 Ringmodulatoren ist eine umfangreiche Beeinflussung der Töne möglich.

C2.0 Die Hüllkurve

Viele Instrumente unterscheiden sich schon durch ihr Lautstärkeverhalten. Bei einer Geige schwingt der Ton langsam an, bis er seine volle Lautstärke erreicht. Im Gegensatz zu einem Klavier; hier ist die volle Lautstärke sofort er-

reicht. Auch das Ausklingverhalten ist bei fast allen Instrumenten unterschiedlich. Ein Saiteninstrument klingt z.B. länger nach als ein Blasinstrument. Mit dem Hüllkurvengenerator des SID ist eine solche Beeinflussung möglich.

C2.1 Attack - das Anschwellen

Die Attack-Phase einer Hüllkurve ist das „Einschwingen“ bis zum lautesten Punkt. Die Zeit, bis der Ton seine volle Lautstärke erreicht (Attack-Time), ist für jede Stimme regelbar.

C2.2 Decay - die Lautstärkespitze

Durch einen harten Anschlag des Klaviers entsteht eine Lautstärkespitze, danach klingt der eigentliche Ton leiser.

Die Attack-Phase endet mit der Lautstärkespitze. Die Decay-Time ist nun die Zeit, bis der Ton seine eigenliche Lautstärke (Sustain) erreicht.

C2.3 Sustain - die Lautstärke

Mit der Decay-Time fällt der Ton in seine eigentliche Lautstärke. Er behält diese Lautstärke solange bei, bis er beendet wird und in die Ausklingphase tritt.

C2.4 Release - Ausklingen

Die Release-Time ist die Zeit, die vergeht, bis der Ton vom Sustainlevel ausgehend vollständig abgeklungen ist.

C2.5 Die ADSR - Kurve

Das Lautstärkeverhalten, beeinflußt durch Attack, Decay, Sustain und Release, sieht grafisch so aus:

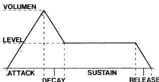


Abb. 18 ADSR-Kurve

C2.6 Programmieren der ADSR-Kurve

Jede Stimme des C=64 besitzt einen Hüllkurvengenerator und er belegt pro Stimme 2 Register. Um sich einer einfacheren Bezeichnung zu bedienen, wird die Startadresse des SID-Chip mit

SI = 54272

bezeichnet. Das Register 0 ist dann SI und Register 7 ist SI+7. Die Register der Hüllkurvengeneratoren:

1.Stimme : SI + 5 und SI + 6 (54277/54278)
2.Stimme : SI + 12 und SI + 13 (54284/54285)
3.Stimme : SI + 19 und SI + 20 (54291/54292)

Die Programmierung der 3 Hüllkurvengeneratoren ist identisch, so daß sich die folgende Beschreibung auf die 1.Stimme beschränkt.

In dem Register SI+5 liegen die Werte für Attack und Decay. In SI+6 für Sustain und Release. Die Register sind in zwei gleichgroße Teile geteilt. 4 Bits für Attack (Sustain) und 4 Bits für Decay (Release).

SI+5 = 0000 0000
Attack Decay

Mit den jeweils 4 zur Verfügung stehenden Bits lassen sich 16 Kombinationen bilden (0000 bis 1111). Das Register SI+6 arbeitet identisch: Bit 0-3 für Release und Bit 4-7 für Sustain. Jede dieser 4 Funktionen kann unabhängig voneinander in den verschiedenen Kombinationen geschaltet werden.
Um Stimme 1 auf Attack 7 und Decay 4 einzustellen, werden die jeweiligen Bits gesetzt.

7 = 0111 4 = 0100 ergibt 01110100 = 116 (Dez)

Dieser Wert für Attack und Decay wird dann in das Register gelegt:

```
POKE SI+5,116
```

Mit einer kleinen Umrechnung lassen sich die Dezimalwerte leichter ermitteln:

```
POKE SI+5, ATTACK * 16 + DECAY
```

wobei die Werte für Attack und Decay nicht größer als 15 werden dürfen. Sustain und Release werden ebenso programmiert.

```
POKE SI+6, SUSTAIN * 16 + RELEASE
```

Um die Hüllkurven der Stimmen 2 und 3 zu bestimmen, brauchen nur die entsprechenden Register verwendet zu werden.

C2.7 Werte sind Zeitspannen

Die Werte für Attack, Decay und Release sind Zeitspannen. Die Tabelle zeigt den Zusammenhang.

WERT	ATTACK	DECAY	RELEASE
0	2ms	6ms	6ms
1	8ms	24ms	24ms
2	16ms	48ms	48ms
3	24ms	72ms	72ms
4	38ms	104ms	104ms
5	56ms	168ms	168ms
6	68ms	204ms	204ms
7	80ms	240ms	240ms
8	100ms	300ms	300ms
9	250ms	750ms	750ms
10	500ms	1.5 s	1.5 s
11	800ms	2.4 s	2.4 s
12	1.0 s	3.0 s	3.0 s
13	3.0 s	9.0 s	9.0 s
14	5.0 s	15.0 s	15.0 s
15	8.0 s	24.0 s	24.0 s

Die wirkliche, genaue Zeitspanne ergibt sich durch multiplizieren der obigen Werte mit dem Faktor 1,01497242.

C2.8 Volumen und Sustain

Am Ende der Attack-Phase wird der lauteste Punkt des Tones erreicht. Diese „Gesamtlautstärke" (Volumen) ist für alle Stimmen gleich. Sie wird in dem Register

Volumen: SI +24 (54296)

eingestellt. Auch das Volumen kann einen Wert zwischen 0 und 15 einnehmen, da wiederum nur die niederwertigen 4 Bits benutzt werden.

xxxx 1111 = 15 (volle Lautstärke)

„15" ist die volle Lautstärke und bei 0 „erklingt" der Ton still (nicht hörbar). Der Sustainlevel (siehe C2.3) gibt die Lautstärke wieder, die nach der Decay-Phase erreicht wird (Sustain-Time siehe C4.2). Ist das Sustain = 15 so ist die Decay-Time gleich Null. Bei einem Sustain von Null, ist die Release-Time gleich Null. Diese beiden Sonderbedingungen sind unabhängig von der eingestellten Decay- und Release-Time.

Zusammenfassung:

Volumen		: POKE SI+24,	VOLUMEN	(0-15)
Attack/Decay	S1	: POKE SI+ 5,	ATTACK	* 16 + DECAY
Sustain/Release	S1	: POKE SI+ 6,	SUSTAIN	* 16 + RELEASE
Attack/Decay	S2	: POKE SI+12,	ATTACK	* 16 + DECAY
Sustain/Release	S2	: POKE SI+13,	SUSTAIN	* 16 + RELEASE
Attack/Decay	S3	: POKE SI+19,	ATTACK	* 16 + DECAY
Sustain/Release	S3	: POKE SI+20,	SUSTAIN	* 16 + RELEASE

C3.0 Die Frequenz

Die zu erzeugenden Frequenzen liegen in einem Bereich von 0 Hertz bis ca. 4000 Hertz. Die Frequenzen werden durch den Systemtakt des C=64 erzeugt. Anzumerken ist, daß es zwei C=64 Versionen gibt: die amerikanische und die PAL-Version (unterschiedliche Netzfrequenzen). Die in der Bundesrepublik verwendetete PAL-Version ist mit 985248.445 Hertz etwas langsamer als die amerikanische Version.
Die Frequenzen werden beim C=64 in Werte zwischen 0 und 65535 angegeben. Dieser Wert ist mit der Gleichung zu berechnen:

WERT = FREQUENZ * 17.0284116

Dabei ist die Frequenz in Hertz anzugeben. Der Kammerton „A" (440 Hertz) hat den Wert

440 * 17.0284116 = 7492.5...

Umgekehrt lassen sich die Werte wieder in Frequenzen umrechnen:

FREQUENZ = WERT / 17.0284116

Der Wert ist in High- und Lowbyte zu zerlegen (siehe G1.3).

FH = INT(WERT/256)
FL = WERT - FH*256

Für den Lowfrequenz- und Highfrequenswert stehen jeder Stimme zwei Register zur Verfügung.

1. Stimme	SI	= Frequenz low	(FL)	(54272)
	SI + 1	= Frequenz high	(FH)	(54273)
2. Stimme	SI + 7	= Frequenz low	(FL)	(54279)
	SI + 8	= Frequenz high	(FH)	(54280)
3. Stimme	SI +14	= Frequenz low	(FL)	(54286)
	SI +15	= Frequenz high	(FH)	(54287)

Um Stimme 1 mit der Frequenz 440 Hertz zu laden, müssen High- und Lowbyte in die Register gePOKEt werden:

POKE SI, FL : POKE SI+1, FH

Im Anhang G2.3 finden Sie die High- und Lowbytewerte der chromatischen Tonleiter über 8 Oktaven. So entfällt die Umrechnung in High- und Lowbyte.

C4.0 Die Wellenform

Vier verschiedene Wellenformen sind vorhanden, um einen Ton zu erzeugen:

Dreieckswelle (Triangle)

Abb. 19 Dreieck-Welle

Sägezahnwelle (Sawtooth)

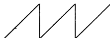


Abb. 20 Sägezahn-Welle

Rechteckwelle (Pulse)

Abb. 21 Rechteck-Welle

Rauschen (Noise)

Jede der drei Stimmen kann auf eine dieser Wellen eingestellt werden. Jede Stimme besitzt das sogenannte „Controlregister", in dem u.a. die Wellenform eingestellt wird.

1.Stimme	SI + 4	(54276)
2.Stimme	SI + 11	(54283)
3.Stimme	SI + 18	(54290)

Benutzt werden in diesen Registern jeweils nur die 4 hochwertigen Bits (Bit 7-4). Jedes dieser 4 Bits entspricht einem Einschalter für die entsprechende Wellenform:

Bit 7	Rauschen
Bit 6	Rechteck
Bit 5	Sägezahn
Bit 4	Dreieck

Um eine Stimme auf Sägezahn einzustellen, wird das Bit 5 gesetzt ('1'):

0010 0000 = 32 (Dezimal)

Stimme 1 würde z.B. mit „POKE SI + 4, 32" auf eine Sägezahnwelle ei gestellt.

0001 0000	=	16	=	Dreieck
0010 0000	=	32	=	Sägezahn
0100 0000	=	64	=	Rechteck
1000 0000	=	128	=	Rauschen

C4.1 Pulsweite

Bei der Bestimmung der Wellenform gibt es einen Sonderfall. Wurde als Wellenform das Rechteck gewählt, ist es notwendig, noch die sogenannte Pulsweite zu bestimmen. Eine Rechteckwelle ist ein periodisches „ein- und ausschalten".
Die Pulsweite bestimmt das Verhältnis zwischen „ein- und ausgeschaltet" sein in den Registern:

1.Stimme	SI + 2 und SI + 3	(54274/54275)
2.Stimme	SI + 9 und SI + 10	(54281/54282)
3.Stimme	SI + 16 und SI + 17	(54288/54289)

Die Pulsweite ist ein 12-Bit-Wort und kann daher die Werte von 0 bis 4095 annehmen. Die niederwertigen 8 Bits stehen jeweils in den Registern SI+2, SI+9 und SI+16. Die hochwertigen 4 Bits stehen in SI+2, SI+10, SI+17. Mit der Einstellung 2048 für die Pulsweite ist das Verhältnis zwischen ein- und ausgeschaltet gleich.

C4.2 Einschalten des Tones – das Gate-Bit

Als letztes wird der Ton eingeschaltet. Für jede Stimme ist ein „Einschalter" vorhanden. Es ist immer das 1. Bit (Bit 0) der Controlregister (Einstellung der Wellenform ,siehe C4.0). Dieses Bit nennt sich „Gate Bit". Wird es gesetzt,

beginnt der Ton (mit der Attack-Phase, siehe C2.5). Der Ton klingt solange mit dem Sustainlevel, bis das Gate-Bit gelöscht ('0') wird. Dann tritt die Release-Phase ein. Mit dem Gate- Bit wird also die Tondauer bestimmt!

C5.0 Der erste Ton

Nach soviel Theorie soll nun der erste Ton aus dem Lautsprecher klingen.

```
10 si = 54272        :rem startadresse sid
20 poke si+ 5,  9 :rem attack=0/decay=9 (00001001)
30 poke si+ 6,135 :rem sustain=15/release=8 (11111000)
40 poke si+24, 15 :rem volumen=15  = volle lautstärke
```

An diesem Punkt sind die globale Lautstärke (Volumen) und die ADSR-Werte für Stimme 1 bestimmt worden. Die Frequenz des Tones soll 440 Herz betragen. Das entsprechende High- und Lowbyte entnehmen Sie der Tabelle in Anhang (G2.3).

```
50 poke si    , 69 :rem lowbyte der frequenz 440 hz
60 poke si+ 1, 29 :rem highbyte der frequent 440 hz
```

Als Wellenform wird die Dreieckswelle gewählt und gleichzeitig das Gate-Bit gesetzt. Damit ist auch der Ton eingeschaltet.

```
70 poke si+ 4, 16+1
```

Die 16 ist der Dezimalwert für die Dreieckswelle, mit der 1 wird das Gate-Bit (Bit 0) gesetzt (16 + 1 = 0001 0001). Der Ton erklingt solange, bis das Gate-Bit wieder gelöscht wird. Eine Zeitschleife bestimmt die Dauer des Tones.

```
80 for t = 1 to 2500 : next
90 poke si + 4, 16 : rem loeschen des gate-bit
```

Nach der Zeitschleife (Zeile 80) wird das Gate-Bit gelöscht – auf Null gesetzt – (16 = 0001 0000). Der Ton klingt mit seiner Release-Phase aus. Probieren Sie ein bißchen mit anderen Werten. Probieren geht über studieren. Achten Sie nur darauf, daß Sie bei Anwahl einer Rechteckwelle noch die Pulsweite einstellen müssen. Um eine andere als die 1. Stimme zu aktivieren, brauchen Sie nur die entsprechenden Register zu ändern.

C5.1 Probieren geht über studieren

Schon mit diesen Möglichkeiten sind unzählige Kombinationen programmierbar. Das folgende Programm bietet Ihnen die Möglichkeit, alle Kombinationen auszuprobieren. In den DATA-Zeilen am Schluß des Programmes ist eine kleine Melodie abgelegt. Nach dem High- und Lowbyte folgt die Tondauer (1/4 = 250). Auch in diesem Programm wird nur die 1. Stimme benutzt.

```
100 si = 54272
110 print chr$(147);
120 input "volumen :";v
30 if v<0 OR V>15 then end
140 print "wellenform :"
150 print "     1. dreieck "
160 print "     2. saegezahn"
170 print "     3. rechteck"
180 print "     4. rauschen"
190 input "bitte waehlen sie :";
200 if w<1 or w>4 then 190
210 if w< 3 then 250
220 input "pulse-breite (0-4095) :";p
230 if p<0 or p>4095 then 220
240 ph = int(p/256): pl = p-ph*256
250 if w=1 then w=16
260 if w=2 then w=32
270 if w=3 then w=64
280 if w=4 then w=128
290 input "attack  (0-15) :";
300 input "decay   (0-15) :";
310 input "sustain (0-15) :";
320 input "release (0-15) :";r
330 poke si+ 5, a*16+d
340 poke si+ 6, s*16+r
350 poke si+24, v
360 if w=64 then poke si+2,pl: poke si+3,ph
370 restore
380 read fh, fl, da
390 if fh = -1 then si+4,0: goto 120
400 poke si, fl: poke si+1, fn
```

```
410 poke si+4, w+1
420 for t = 1 to da: next t
430 poke si+4, w
440 for t = 1 to 50: next t
450 goto 380
500 data 014,162,125,008,180,250,009,196,125
510 data 011,158,375,013,010,125,014,162,125
520 data 014,162,125,014,162,250,014,162,500
530 data 014,162,125,008,180,250,009,196,125
540 data 011,158,250,011,158,250,009,196,250
550 data 008,180,750
560 data  -1, -1, -1
```

C6.0 Mehrstimmigkeiten

Die 3 Stimmen des SID sind, wie inzwischen bekannt, unabhängig voneinander regelbar. Daher ist eine Programmierung von Zweiklängen oder Akkorden möglich.

C6.1 Der Akkord

Um ein Stimme zu aktivieren, müssen folgende Register eingestellt werden:

Volumen
ADSR-Kurve
Frequenz
Wellenform
Gate-Bit

Wenn zwei Töne erklingen sollen, ist es natürlich notwendig, die zweite Stimme ebenso zu bestimmen. Die Werte können selbstverständlich unterschiedlich sein. Im folgenden Beispiel wird der C-Dur Akkord (C-E-G) erklingen. Die 3 Hüllkurven werden identisch eingestellt (Attack=5, Decay=5, Sustain=7, Release=8).

C	=	2228	FH	=	8 / FL	= 180
E	=	2807	FH	=	10 / FL	= 247
G	=	3338	FH	=	13 / FL	= 10

```
10 si = 54272
20 poke si+ 5, 5*16+5: poke si+ 6, 7*16+8
30 poke si+12, 5*16+5: poke si+13, 7*16+8
40 poke si+19, 5*16+5: poke si+20, 7*16+8
50 poke si+24,15
60 poke si   ,180: poke si+ 1, 8
70 poke si+ 7,247: poke si+ 8,10
80 poke si+14, 10: poke si+15,13
90 w = 32
100 poke si+4,w+1: poke si+11,w+1: poke si+18,w+1
110 for t = 1 to 2000: next t
120 poke si+4, w: poke si+11, w: poke si+18, w
130 for t = 1 to 100: next
140 poke si+24,0
```

Die Zeitschleife (Zeile 130) bewirkt, daß die Release-Phase ausklingen kann, bis mit SI + 24,0 die Lautstärke ausgeschaltet wird. Deutlich ist zu erkennen, daß die Programmierung der Stimmen immer nach demselben Schema erfolgt. Im obigen Beispiel erklangen alle 3 Stimmen gleichzeitig. Bei einem Akkord ist das ja auch sinnvoll. In der mehrstimmigen Musik (Polyphonie) spielen die Stimmen nicht nur in verschiedenen Frequenzen, sondern auch zeitlich verschoben.

C6.2 Prinzip der Polyphonie

Um ein 2- oder 3-stimmiges Musikstück zu programmieren, geht man von folgender Idee aus:
Das Musikstück wird in seine kleinste Zeiteinheit aufgeteilt (kleinste Notendauer). Hat die kleinste Note aller Stimmen eine Dauer von 1/16, so werden alle anderen Noten in diesem Zeitfaktor umgerechnet. Das bedeutet, daß eine 1/4 Note in vier 1/16 Noten aufgeteilt wird. Pausen werden ebenso in die kleinste Zeiteinheit umgerechnet.

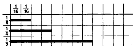


Abb. 21a Prinzip Polyphonie

Die Melodie jeder Stimme wird in diese Einheiten zerlegt, wobei jede Note die Dauer von 1/16 Noten besitzt.

```
1.Stimme:  (A   A   A   A) (E   E   E   E) (C   C   C   C)
2.Stimme:  (-   -) (A   A) (-   -) (A   A) (-   -) (A   A)
3.Stimme:  (A) (E) (C) (-) (A) (E) (C) (-) (A) (E) (C) (-)
Zeiteinh:   1   2   3   4   5   6   7   8   9  10  11  12
```

Der Strich (-) bedeutet hierbei eine Pause von je 1/16. Die Stimme 1 besteht nur aus 1/4 Noten, Stimme 2 aus 1/8 und Stimme 3 aus 1/16 Noten. In den so entstandenen 12 Zeiteinheiten erklingen pro Stimme je ein Ton oder ein Teil eines Tones. In Zeiteinheit 1 klingt Stimme 1 mit einem A, Stimme 2 hat Pause und Stimme 3 ebenfalls mit A. Ein Ton beginnt immer dann, wenn das Gate-Bit gesetzt ist und endet, wenn das Gate-Bit gelöscht wird (siehe C4.2). Der 1/4 Ton A der Stimme 1 beginnt in Zeiteinheit 1 und endet in Zeiteinheit 4. Daraus folgt, daß das Gate-Bit in den Zeiteinheiten 1-3 der Stimme 1 gesetzt sein muß. In der Zeiteinheit 4 wird das Gate-Bit gelöscht; bei einer Dauer von 1/8 entsprechend. Bei Tönen mit der Dauer von 1/16 wird das Gate-Bit gesetzt, damit der Ton erklingen kann.
Pausen werden erzielt, indem die Wellenform ausgeschaltet wird und hierdurch kein Ton erklingt.
Damit sind pro Ton und Zeiteinheit 3 Parameter notwendig. Zum Einen das High- und Lowbyte der Frequenz und weiterhin die Wellenform mit gesetztem oder gelöschtem Gate-Bit. An einem Beispiel soll dieses verdeutlicht werden. Als Wellenform ist die Dreieckswelle gewählt (16). Die High- und Lowbyte-Werte für die Frequenz sind aus dem Anhang G2.3 entnommen. Als erstes wird die Stimme 1 „umgerechnet":

A / Oktave 4 :Highbyte= 29 / Lowbyte= 69

In den ersten drei Zeiteinheiten ist das Gate-Bit gesetzt.

1. Zeiteinheit: HB = 29 , LB = 69 , WELLE = 16 + 1

Die „WELLE" ist die Wellenform (16) plus dem Gate-Bit (+1). Die beiden folgenden Zeiteinheiten sind identisch:

2. Zeiteinheit: HB = 29 , LB = 69 , WELLE = 16 + 1
3. Zeiteinheit: HB = 29 , LB = 69 , WELLE = 16 + 1

In der 4. Zeiteinheit endet der Ton. Das Gate-Bit wird gelöscht.

4. Zeiteinheit: HB = 29 , LB = 69 , WELLE = 16 + 0

Die Zeiteinheiten 5 bis 12 bauen sich ebenso auf:

5. - 7. Zeiteinheit: HB = 21 , LB =237 , WELLE = 16 + 1
8. Zeiteinheit: HB = 21 , LB =237 , WELLE = 16 + 0
9. - 11. Zeiteinheit: HB = 34 , LB =207 , WELLE = 16 + 1
12. Zeiteinheit: HB = 34 , LB =207 , WELLE = 16 + 0

Die Umrechnungen der Stimmen 2 und 3 sind identisch. Die Pausen werden dadurch erzielt, daß die Welle(nform) ausgeschaltet wird (0). Die ersten beiden Zeiteinheiten der 2. Stimme könnten also so aussehen:

1. - 2. Zeiteinheit: HB = ??? , LB =??? , WELLE = 0 !!!!

Die Frequenz (HB und LB) ist dabei unwichtig, weil der Ton ja ausgeschaltet ist (WELLE = 0). Das folgende Programm faßt noch einmal die Grundlagen der mehrstimmigen Programmierung zusammen. Für das Programm ist die 12-teilige Melodie aus C6.2 benutzt worden.

```
10 rem****************** 3  stimmigkeit *******************
14 :
15 rem***************** sid - startadresse ****************
16 :
20 si = 54272
24 :
25 rem**************** array f. stimme+welle **************
26 :
```

```
30 dim fh(2,11),fl(2,11),we(2,11)
34 :
35 rem***************** einlesen der stimmen *************
36 :
40 for s = 0 to 2
50 for i = 0 to 11
60 read fh(s,i),fl(s,i),we(s,i)
70 next i: next s
74 :
75 rem**************** setzen der adsr-kurven *************
76 :
80 poke si+ 5, 0: poke si+ 6,240
90 poke si+12, 0: poke si+13,112
100 pokesi+19, 0: poke si+20,240
104 :
105 rem***************** volumen  einstellen *************
106 :
110 poke si+24,15
114 :
115 rem***************** abspielen der melodie ************
116 :
120 for i = 0 to 11
130 poke si  ,fl(0,i):poke si+ 7,fl(1,i):poke si+14,fl(2,i)
135 poke si+1,fh(0,i):poke si+ 8,fh(1,i):poke si+15,fh(2,i)
140 poke si+4,we(0,i):poke si+11,we(1,i):poke si+18,we(2,i)
144 :
145 rem**************** zeitschleife fuer 1/16 ************
146 :
150 for t = 1 to 50:next t
154 :
160 next i
164 :
165 rem******************* ende ?????? *******************
166 :
170 get a$:if a$ <> "" then poke si+24,0: end
180 for t = 1 to 200: next t
190 goto 120
999 :
```

```
1000 rem************** werte fuer stimme 1 ***************
1001 :
1010 data  29, 69, 17: rem  highbyte, lowbyte, wellenform
1020 data  29, 69, 17
1030 data  29, 69, 17
1040 data  29, 69, 16
1050 data  21,237, 17
1060 data  21,237, 17
1070 data  21,237, 17
1080 data  21,237, 16
1090 data  17,103, 17
1100 data  17,103, 17
1110 data  17,103, 17
1120 data  17,103, 16
1999 :
2000 rem************** werte fuer stimme 2 ***************
2001 :
2010 data   0,   0,  0
2020 data   0,   0,  0
2030 data  17, 103, 33
2040 data  17, 103, 32
2050 data   0,   0,  0
2060 data   0,   0,  0
2070 data  29,  69, 33
2080 data  29,  69, 32
2090 data   0,   0,  0
2100 data   0,   0,  0
2110 data  21, 237, 33
2120 data  21, 237, 32
2999 :
3000 rem************** werte fuer  timme 3 ***************
3001 :
3010 data  14, 162, 17
3020 data  10, 247, 17
3030 data   8, 180, 17
3040 data   0,   0,  0
3050 data  14, 162, 17
3060 data  10, 247, 17
3070 data   8, 180, 17
```

```
3080 data  0,   0,  0
3090 data 14, 162, 17
3100 data 10, 247, 17
3110 data  8, 180, 17
3120 data  0,   0,  0
3999 :
```

Die (1/16) Töne der drei Stimmen liegen in den DATA-Zeilen 1000 bis 3999. Für jeden 1/16 Notenwert ist das entsprechende High- und Lowbyte, sowie die Wellenform mit oder ohne gesetzem Gate-Bit angegeben. Diese Werte sind in 3 Variablenfelder eingelesen worden (Zeile 40-70). Das Einlesen in Variablen ist notwendig, da nur so ein ausreichendes Tempo erreicht wird. Das eigentliche Spielen der Melodie wird in den Zeilen 120 bis 140 erledigt. Die Länge einer Zeiteinheit (1/16 Note) wird durch die Zeitschleife in Zeile 150 bestimmt.
Ich hoffe, daß schon an dieser Stelle deutlich wird, welche unendliche Vielfalt im musikalischen Talent des SID-Chips steckt. Leider werden diese Fähigkeiten durch das COMMODORE BASIC nicht berücksichtigt und es ist schon eine mühselige Arbeit, eine kleine, 3-stimmige Melodie ertönen zu lassen. Für die sehr kleine Melodie in unserem Beispiel waren schon 108 DATA's nötig. Durch eine „Verschlüsselung" der Noten ist es allerdings möglich, mit wesentlich weniger Datenbytes auszukommen.

C6.3 Verschlüsselung von Noten

An dieser Stellen werden zwei Möglichkeiten der Verschlüsselung von Noten vorgestellt. Auch in diesem Punkt sind Ihrer Kreativität keine Grenzen gesetzt.

Eine Oktave ist in 12 Töne geteilt (chromatische Tonleiter). Der Oktavton eines Grundtones hat immer den doppelten Wert (A = 440 Hz , A' = 880 Hz). Die chromatische Tonleiter ergibt sich durch das Multiplizien mit dem Faktor 1.05946309.

$$A = 440 \text{ Hz} \quad A\# = 440 * 1.05946309 = 466.163 \text{ Hz}$$

Analog dazu einen Halbton nach „unten":

Grundton / 1.05946309

Die chromatische Tonleiter läßt sich von einem beliebigen Grundton folgend berechnen:

Faktor	=	1,05946309
Prime	=	Grundton * Faktor ∧ 0
kleine Sekunde	=	Grundton * Faktor ∧ 1
große Sekunde	=	Grundton * Faktor ∧ 2
kleine Terz	=	Grundton * Faktor ∧ 3
Oktave	=	Grundton * Faktor ∧ 12 (= 2)

Schon mit diesen Daten ist eine Verschlüsselung möglich, indem z.B. in den DATA's nur das Verhältnis vom Ton zum Grundton angegeben wird (Interval). An einem Beispiel soll dieses erläutert werden.

```
10 rem********* beispiel einer verschluesselung **********
14 :
15 rem*************** sid - startasresse ****************
16 :
20 si = 54272
24 :
25 rem*************** adsr-kurve stimme 1 **************
26 :
30 poke si+ 5, 19: poke si+ 6,240
34 :
35 rem***************** volumen  einstellen *************
36 :
40 poke si+24, 15
44 :
45 rem****************** einlesen  und spielen ***********
46 :
50 read nc
55 if nc = 99 then poke si+24,0:end
60 n = 2228 * 1.05946309 ∧ nc
65 rem 2228 = parameter f. c - 3 (130 hz)
70 fh = int(n/256): fl = n - fh *256
80 poke si,fl: poke si+1,fh
90 poke si+4, 33
100 for t = 1 to 100: next t
```

```
110 poke si+4,32
120 goto 50
999 :
1000 rem******************* intervalldaten *************
1001 :
1010 data 0,2,4,5,7,9,11,12,11,9,7,5,4,2,0,99
1999 :
```

Die C-Dur Tonleiter erklingt auf- und abwärts. Als Grundton ist der Ton C-3 mit dem Parameter 2228 angegeben. Durch eine Umrechnung wird der Parameter des tatsächlichen Tones errechnet und dann in High- und Lowbyte umgerechnet.

C6.4 Bessere Verschlüsselung

Es gibt allerdings eine wesentlich bessere Art der Notenverschlüsselung. Hierzu werden in einer Zahl die Note, die Oktave und die Länge verschlüsselt, so daß wesentlich weniger DATA's eingegeben werden müssen. Den Noten (C bis H) werden Zahlen von 0 bis 11 zugeordnet.

Note		Wert
C	=	0
C#	=	1
D	=	2
D#	=	3
....		....
H	=	11

Die Dauer einer Note wird wieder in der kleinsten Zeiteinheit angegeben. Ist die kleinste Zeiteinheit aller Stimmen die 1/16 Note, so bekäme eine 1/4 Note die Dauer 4 (4*1/16 = 1/4).
Die Oktave, in der sich der Ton befindet, ist mit den Werten von 0 bis 7 gekennzeichnet (0 die tiefste Oktave C-0). Der Notencode ist dann mit der Gleichung:

Notencode = ((8 * Dauer) + Oktave) * 16 + Notenwert

zu berechnen. Das Decodieren des Notencodes ist folgendermaßen möglich:

```
Dauer      = INT( Notencode /128 ):Anzahl der Zeiteinheiten
Oktave     = ( Notencode - 128 * Dauer ) / 16
Notenwert  = Notencode -128 * Dauer - 16 * Oktave
```

Bei Verschlüsselung des Tones A-3 (A in der 3. Oktave) mit einer Länge von 1/4 (bei der kleinsten Zeiteinheit von 1/16) erhalten Sie den Wert

$$((8*4)+3)*16+9 = 569 \;!!!$$

Eine Pause wird erzielt, indem die Dauer der Pause als negativer Wert angegeben wird. Hierdurch wird der gesamte Notencode negativ.

$$((8*-4)+3)*16+9 = -455$$

Die Oktave und der Notenwert spielen keine Rolle, da bei einer Pause die Wellenform ausgeschaltet wird (0) und der Ton nicht erklingt.

Das nun folgende Programmbeispiel zeigt die Anwendung der Verschlüsselung von Noten. Zu bemerken ist noch, daß die Frequenz des tatsächlichen Tones durch die Teilung der Grundfrequenz erreicht wird. Die Grundfrequenzen, hier C-7 bis H-7, sind in den Variablen fq(0) bis fq(11) abgelegt.

```
0 rem************* 3 stimmiges musikprogramm ************
1 :
2 print chr$(147);"bitte warten"
3 pri t: print "ich rechne .";
4 :
5 rem***************** sid-startadresse *****************
6 :
10 si = 54272
14 :
15 rem************* loeschen der sid-register *************
16 :
20 for i= si to si+24
30 poke i, 0
40 next i
44 :
45 rem************* array f. toene+wellenform *************
```

```
46 :
50 dim fh(2,300), fl(2,300), we(2,300)
60 dim fq(11)
64 :
65 rem************ wellenform stimme 1 - 3  ************
66 :
70 w(0)=16:w(1)=64:w(2)=32
74 :
75 rem************ stimme 2 pulsebreite 2048 ************
76 :
80 poke si+9,0: poke si+10,8
84 :
85 rem************ parameter oktave7 c7 - h7 ************
86 :
87 fq(0)=35643:        rem c 7
88 fq(1)=37762:        rem c#7
89 fq(2)=40008:        rem d 7
90 fq(3)=42387:        rem d#7
91 fq(4)=44907:        rem e 7
92 fq(5)=47578:        rem f 7
93 fq(6)=50407:        rem f#7
94 fq(7)=53404:        rem g 7
95 fq(8)=56580:        rem g#7
96 fq(9)=59944:        rem a 7
97 fq(10)=63508:       rem a#7
98 fq(11)=67284:       rem h 7 !!!!!!!
99 :
100 rem************ einlesen und umrechnen ************
101 :
110 for i = 0 to 2
120 z = 0
130 read nc
140 if nc = 0 then 310
150 w1 = w(i)+1
160 w2 = w(i)
170 if nc<0 then nc=-nc:w1=0:w2=0
180 d% = nc/128
190 o% = (nc-128*d%)/16
200 n% = nc-128*d%-16*o%
```

```
210 fr = fq(n%)
220 if o% = 7 then 260
230 for j = 6 to o% step -1
240 fr = fr/2
250 next j
260 hf% = fr/256: lf% = fr-256*hf%
270 if d%= 1 then fh(i,z)=hf%: fl(i,z)=lf%:we(i,z)=w1:z=z+1:
    goto 130
280 for j=1 to d%-1: fh(i,z)=hf%: fl(i,z)=lf%: we(i,z)=w1:
    z=z+1: next j
290 fh(i,z)=hf%: fl(i,z)=lf%: we(i,z)=w2
300 z = z+1:print ".";: goto 130
310 if z>gz then gz=z
320 next i
324 :
325 rem************ adsr-kurve d. stimmen1-3 *************
326 :
330 poke si+ 5, 0: poke si+ 6,240
340 poke si+12,85: poke si+13,133
350 poke si+19,10: poke si+20,197
354 :
355 rem************ volumen einstellen  voll **************
356 :
360 poke si+24,15
364 :
365 rem************* beginn des abspielen *****************
366 :
370 for j = 0 to gz
380 poke si  ,fl(0,j):poke si+ 7,fl(1,j):poke si+14,fl(2,j)
390 poke si+ 1,fn( ,j):poke si+ 8,fn(1,j):poke si+15,fn(2,j)
400 poke si+ 4,we( ,j):poke si+11,we(1,j):poke si+18,we(2,j)
410 for t = 1 to 50: next
420 next j
424 :
425 rem*********************** ende  **********************
426 :
430 for t = 1 to 300: next
440 poke si+24,0
450 print: print: print
```

```
460 print "noch einmal hoeren mit   - goto 360 -"
470 end
999 :
1000 rem*********** notencode fuer 1.stimme **************
1001 :
1010 data 594,594,594,596,596
1020 data 1618,587,592,587,585,331,336
1030 data 1097,583,585,585,585,587,587
1040 data 1609,585,331,337,594,594,593
1050 data 1618,594,596,594,592,587
1060 data 1616,587,585,331,336,841,327
1070 data 1607
1998 data 0:    rem ende 1.stimme
1999 :
2000 rem*********** notencode fuer 2.stimme **************
2001 :
2010 data 583,585,583,583,327,329
2020 data 1611,583,585,578,578,578
2030 data 196,198,583,326,578
2040 data 326,327,329,327,329,326,578
2050 data 585,1606,582,322,324,582,587
2060 data 329,327,1606,583
2070 data 327,329,587,331,329
2080 data 329,328,1609,578,834
2090 data 324,322,327,585,1602
2998 data 0:    rem ende 2.stimme
2999 :
3000 rem*********** notencode fuer 3.stimme **************
3001 :
3010 data 567,566,567,304,306,308,310
3020 data 1591,567,311,310,567
3030 data 306,304,299,308
3040 data 304,171,176,306,291,551,306,308
3050 data 310,308,310,306,295,297,299,304
3060 data 1586,562,567,310,315,311
3070 data 308,311,297
3080 data 1586,567,560,311,309
3090 data 308,309,306,308
3100 data 1577,299,295,306,310,311,304
```

```
3110 data 562,546,1575
3998 data 0:     rem ende 3.stimme
```

Ich hoffe, daß Ihnen das Programm trotz des Umfanges einige Freude bereitet. Damit Sie selber mehrstimmig programmieren können, stelle ich Ihnen ein Programm vor, daß die Eingaben von Ton, Oktave und Dauer in den Notencode umrechnet. Der Vollständigkeit halber ist auch die Decodierung mit aufgenommen. Wenn Sie eigene Stücke codieren, achten Sie bitte immer darauf, daß die Dauer des Tones einem Mehrfachen der kleinsten Einheit des gesamten Stückes entspricht.

C6.5 Codieren und Decodieren

```
10 rem************* codieren / decodieren ***************
14 :
15 rem******************** menue ***********************
16 :
20 print chr$(147);
30 print "                menue"
40 print:print:print
50 print:print " 1.)  codieren ............"
60 print " 2.)  decodieren .........."
70 print " 3.)  programmende ........"
80 print
90 print "      bitte waehlen sie ...."
100 get a$:if a$ = "" then 100
110 if a$ = "3" then end
120 if a$ = "1" then 200
130 if a$ = "2" then 500
140 goto 100
144 :
195 rem***************** codieren **********************
196 :
200 print chr$(147)
210 print "              codierung"
```

```
220 print
230 input "* note    (c - h)   :";n$
235 if n$= "x" then 20
240 input "* oktave (0 - 7)   :";o%
250 input "* dauer             :";d%
260 print
270 restore:for i=0 to 11
280 read ns$
290 if ns$=n$ then 330
300 next i
310 if n$ = "p" then d%=-d%:goto 370
320 print "   !! fehler !!  note nicht bekannt !!":
330 if o% >1 and o% <8 then 350
340 print "   !! fehler !!  oktave unmoeglich  !!":
350 if d% >0 then 370
360 print "   !! fehler !!  dauer ist ""null""   !!":
370 if fe=1 then fe=0: goto 220
380 nc = ((8*d%)+o%)*16+i
390 print
400 print "    * notencode
410 goto 220
414 :
495 rem******************** decodieren ********************
496 :
500 print chr$(147)
510 print "                decodierung"
520 print
530 input "* notencode  :";nc
535 if nc = 0 then 20
540 d% = nc/128
550 o% = (nc-128*d%)/16
560 if nc <0 then n$="pause":d%=-d%:goto 610
570 n% = nc-128*d%-16*o%
580 restore: for i = 0 to n%
590 read n$
600 next i
610 print
620 print "   * note       : ";n$
630 print "   * oktave     :";o%
```

```
640 print "   * dauer      :";d%
650 goto 520
999 :
1000 rem*********** data fuer die noten-strings ***********
1001 :
1010 data C,C#,D,D#,E,F,F#,G,G#,A,A#,H.
1999 :
```

C7.0 Verfremdung von Tönen

Im SID befinden sich 3 Typen von Filtern, die getrennt oder zusammen benutzt werden können. Für die Filter werden die Register SI+21 bis SI+24 verwendet (54293-54296). Dabei werden drei Filtertypen unterschieden (Hochpass-, Tiefpass- und Bandpass-Filter). Ein Filter besitzt die Eigenschaft, bestimmte Frequenzen eines Tones zu dämpfen ('sperren, nicht durchlassen'). Mit einer Filterfrequenz wird der Arbeitspunkt des Filters eingestellt (Frequenz, wo der Filter anfängt zu arbeiten). Diese Filterfrequenz wird in den Registern SI+21 (Bits 0 bis 2) und SI+22 (Bits 3 bis 10) eingestellt. Der Wert für die Filterfrequenz ist ein 11-Bit-Wort, wobei die niederwertigen 3 Bits in den ersten 3 Bits des Registers SI+21 liegen. Die Filterfrequenz (in Hertz) errechnet sich aus:

Filterfrequenz = (30 + Wert * 5.818181...)

Um den Wert (11-Bit-Wert) einer Filterfrequenz zu berechnen, stellt man die obige Gleichung nach 'Wert' um:

Wert = (Filterfrequenz - 30) / 5.8181818....

Beispiel: Filterfrequenz = 5988 Hertz

Wert = (5988 - 30) / 5.8181
Wert = 1024 !!!

1024 Dezimal ist 10000000000 Binär (Die 3 niederwertigen Bits befinden sich in den Bits 0-2 des Registers SI+21).

```
POKE SI+21 ,  0      (xxxxx000)
POKE SI+22 ,128      (10000000)
```

Die Filterfreuqenz ist in 4095 Stufen regelbar (11 Bit). Sie ist für alle Filtertypen sowie für alle Tongeneratoren zuständig.
Nachdem die Filterfrequenz bestimmt und in den Registern SI+21 und SI+22 abgelegt ist, bestimmen die Bits 0-3 des Registers SI+23, welche Stimme durch den Filter läuft. Durch das Setzen der Bits

Bit 0	= 1.Stimme über Filter (xxxx0001)	= 1
Bit 1	= 2.Stimme über Filter (xxxx0010)	= 2
Bit 2	= 3.Stimme über Filter (xxxx0100)	= 4
Bit 3	= Externes Signal ü. Filter (xxxx1000)	= 8

werden sie eingeschaltet.

Das Bit 3 des Registers SI+23 legt ein externes Signal auf die Filter. Ein externes Signal kann in den SID-Synthesizer eingespielt und beeinflußt werden. Es ist möglich, eine Gitarre (mit Vorverstärker), eine Orgel, etc in den Synthesizer einzuspielen und zu verändern. Die Eingangsspannung des externen Signals darf nicht mehr als 3 Volt betragen. Der Eingang (EXT IN) liegt an der Audio/Video-Buchse (die 5-polige) an Pin 5 (AUDIO IN).

Die 4 hochwertigen Bits (Bit7-4) des Registers SI+23 stellen die „Resonanz" der Filterfrequenz ein (16 Stufen 0000xxxx - 1111xxxx). Bestimmte Frequenzbereiche der Filterfrequenz werden hervorgehoben. Die stärkste Wirkung wird mit 1111xxxx (15) erziehlt und mit 0000xxxx hat die Resonanz keine Wirkung.

```
POKE SI+23, 1     0000 0001  Resonanz 0 , Stimme 1
```

Die Stimme 1 wird so über den Filter geleitet, wobei die Resonanz gleich Null ist. Bisher ist noch nicht bestimmt worden, über welchen der 3 Filter das Signal laufen soll. Durch die Bits 4-6 des Registers SI+24 wird bestimmt, welcher der 3 Filter eingeschaltet ist. Durch Setzen der Bits wird der entsprechende Filtertyp aktiviert.

Bit 4	= Tiefpass-Filter (Lowpass)
Bit 5	= Hochpass-Filter (Highpass)
Bit 6	= Bandpass-Filter (Bandpass)

Die niederwertigen 4 Bits dieses Registers (SI+24) dienen der Bestimmung des Volumens (siehe C2.8). Das letzte (7.) Bit hat eine Sonderfunktion: durch

Setzen von Bit 7 wird die Stimme 3 „ausgeschaltet". Die Stimme 3 (wenn sie eingeschaltet ist) arbeitet zwar, aber der Ton wird nicht ausgegeben (siehe C8.1).

C7.1 Hochpass-Filter

Der Hochpass-Filter läßt alle Frequenzen oberhalb der Filterfrequenz durch und dämpft die darunterliegenden. Die Skizze zeigt das theoretische Verhalten eines Hochpass-Filters. Die Dämpfung beträgt pro Oktave 12 dB.

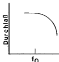


Abb. 22 Hochpass

C7.2 Tiefpass-Filter

Der Tiefpass-Filter dämpft die hohen Frequenzen ab der Filterfrequenz um 12 dB pro Oktave.

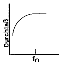


Abb. 23 Tiefpass

C7.3 Bandpass-Filter

Nur in einem schmalen Band (Bereich) um die Filterfrequenz werden die Frequenzen „durchgelassen", alle anderen werden gedämpft. Die Dämpfung beträgt hier 6 dB pro Oktave.

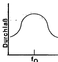


Abb. 24 Bandpass

C7.4 Kombinationen der Filtertypen

Erwähnt werden soll an dieser Stelle auch noch, daß eine Kombination der Filter möglich ist. Durch Setzen der entsprechenden Bits im Register SI+24 werden die verschiedenen Filtertypen angeschaltet. Das Resultat ist z.B. eine UND-Kombination von zwei Filterfunktionen.

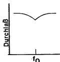


Abb. 25 Kombinierter Hoch- und Tiefpass

Sie sehen, daß es schon jetzt unzählige Kombinationsmöglichkeiten zur Tonerzeugung gibt.
Viele Filtereinstellungen haben keine oder nur eine geringe Wirkung. Das nun folgende Programm soll Ihnen die Möglichkeit geben, ein bißchen mit den Filtern zu experimentieren. Die Filterfrequenz wird um einen konstanten Ton vergrößert bzw. verkleinert. Da eine Filterfrequenz oberhalb des Wertes 1240 keine wesentliche Änderung ergibt, wird die Filterfrequenzmodifikation nur bis zu diesem Wert durchgeführt. Viel Spaß.

```
10 rem******************** filter test ******************
14 :
15 rem************** waehlen: filter,welle **************
16 :
20 si=54272
30 poke si+ 5,19: poke si+ 6,240: rem adsr stimme 1
40 poke si  ,180:poke si+ 1,  8: rem frequenz stimme1(c-3)
50 poke si+ 2,  0:poke si+ 3,  4: rem pulseb.stimme1 (1024)
55 poke si+23,  1:              rem stimme 1 ueber filter
60 print chr$(147)
70 print "           testen der filter"
80 print:print:print
90 print " (1) dreieck  (2) saegezahn"
100 print " (3) rechteck  (4)  rauschen"
110 print "  * bitte waehlen"
120 get a$: if a$= ""then 120
130 if a$=1 then w = 16:goto 200
140 if a$=2 then w = 32:goto 200
150 if a$=3 then w = 64:goto 200
160 if a$=4 then w =128:goto 200
170 poke si+24,0: poke si+4,0:end
200 print
210 print "(1) lowpass  (2) Hhighpass (3) bandpass"
220 get a$: if a$= "" then 220
230 if a$ = "1" then fi = 16:goto 270
240 if a$ = "2" then fi = 32:goto 270
250 if a$ = "3" then fi = 64:goto 270
260 goto 220
270 gosub 1000
280 goto 80
```

```
999 :
1000 rem*********** filter auf- abwaerts ****************
1001 :
1010 poke si+24, fi+15:           rem filter + volumen
1020 poke si+ 4, w + 1:  rem wellenform + gate-bit stimme 1
1030 for i = 0 to 155
1040 for t=1 to 20:next
1050 poke si+22,i: rem filterfrequenz aendern (0 - 1240)
1060 next i
1070 for i = 155 to 0 step -1
1080 for t=1 to 20:next
1090 poke si+22,i: rem filterfrequenz aendern (1240 - 0)
1100 next
1110 for i = 15 to 0 step -1
1130 poke si+24,i: rem volumen langsam auf still
1140 next
1150 return
1154 :
```

C7.5 Ringmodulation

Bei einer Ringmodulation werden zwei Stimmen miteinander addiert oder subtrahiert. Die Stimme, die moduliert werden soll, muß eine Dreieckswelle besitzen. Eine Ringmodulation erzeugt durchweg unharmonische Klänge und ist für die Erzeugung von Geräuschen jeder Art sehr nützlich.
Jede der 3 Stimmen kann mit einer anderen Stimme ringmoduliert werden.

Stimme 1	mit	Stimme 3
Stimme 2	mit	Stimme 1
Stimme 3	mit	Stimme 2

In den jeweiligen Control-Registern (SI+4,SI+11,SI+18) dient Bit 2 für die Einschaltung der Ringmodulation. Wird Stimme 1 auf Ringmodulation geschaltet, wird sie mit Stimme 3 moduliert (siehe oben). Selbstverständlich muß dann Stimme 3 auch aktiviert sein.

Stimme 1	POKE SI+ 4,	16+4+1	0001 0101
Stimme 2	POKE SI+ 11,	16+4+1	0001 0101
Stimme 3	POKE SI+ 18,	16+4+1	0001 0101

Hier wurden die einzelnen Stimmen auf Dreieckwelle (16) und Ringmodulation (4) geschaltet. Weiterhin ist das Gate-Bit gesetzt (1). Ich möchte hier nicht weiter auf die Ringmodulation des SID eingehen, denn es bieten sich auch hierbei eine Unzahl von Möglichkeiten. Ihrer Phantasie sind keine Grenzen gesetzt. Um aber den Effekt der Ringmodulation zu testen, biete ich Ihnen hier ein kurzes Demonstrationsprogramm.

```
10 rem****************** ringmodulator ******************
20 si = 54272
30 for i = si to si+27: poke i,0: next
40 poke si+5,9:   rem a=0/d=9  s=0/r=0
50 print chr$(147);
60 input"fre. stimme 3 (0-255) :";m
70 if m>255 then poke si+24,0:end
80 poke si+15, m: rem highbyte s3
90 poke si+24,15: rem volumen
100 read fh,fl,d
110 if fh = -1 then restore:goto 60
120 poke si+1,fh: poke si,fl
130 poke si+4,16+4+1: rem 0001 0101
140 for t=1 to d: next t
150 poke si+4,16+4+0: rem 0001 0100
160 goto 100
999 :
1000 rem************ data fuer die melodie ***************
1001 :
1010 data 014,162,125,008,180,250
1020 data 009,196,125,011,158,375
1030 data 013,010,125,014,162,125
1040 data 014,162,125,014,162,250
1050 data 014,162,500, -1, -1, -1
1060 :
```

C7.6 Synchronisation

Neben der Ringmodulation gibt es weiterhin die Möglichkeit der Synchronisation zweier Stimmen. Bei der Synchronisation werden die beiden

Stimmen miteinander logisch UND-verknüpft. Ist eine der beiden Wellenformen gleich Null, ist auch die Ausgabe gleich 0. Jede Stimme kann synchronisiert werden:

Stimme 1 mit Stimme 3
Stimme 2 mit Stimme 1
Stimme 3 mit Stimme 2

Um eine Synchronisation einzuschalten, wird das Bit 1 der Control Register (SI+4, SI+11, SI+18) gesetzt. Hierbei kann jede der 4 Wellenformen benutzt werden. Beispiele:

```
Stimme 1 :   POKE SI + 4  , 32+2+1
Stimme 2 :   POKE SI +11  , 16+2+1
```

Es lassen sich sehr schöne Klänge erzeugen. Es treten Effekte der harmonischen Verschiebung auf. Ändern Sie das Ringmodulator-Demonstrationsprogramm folgendmaßen ab und sie erhalten ein Beispielprogramm zur Synchronisation:

```
130 poke si+4, 16+2+1:     rem 0001 0011
140 for t = 1 to d/2: next t
150 poke si+4, 16,2,0:     rem 0001 0010
```

C8.0 Besonderheiten der Stimme 3

Alle bisher beschriebenen SID-Register (SI bis SI+24) konnten nur beschrieben, aber nicht gelesen werden. Es ist in den Registern SI+27 möglich, den momentanen Zustand des Oszillators (Stimme) 3 auszulesen. Ist für die Stimme 3 die Dreieckwelle als Wellenform gewählt worden, ändert sich der Inhalt von 0 bis 255 und wieder zurück auf 0. Bei einer Sägezahnwelle steigt der Inhalt von 0 bis 255 und fällt dann sofort wieder auf 0. Bei der Wellenform Rechteck findet man nur den Inhalt 0 oder 255, deren Länge in Abhängigkeit zur eingestellten Pulsbreite steht. Bei Rauschen befinden sich im Register SI+27 zufällig verteilte Zahlen zwischen 0 und 255 (dieses Register dient auch der Erzeugung von Zufallszahlen - RND -). Das Register SI+27 ist aus diesen Gründen ohne weiteres für Steuerzwecke zu nutzen.

Im Register SI+28 ist der momentane Verlauf der Hüllkurve (ADSR) von Stimme 3 enthalten. Der Inhalt liegt wieder zwischen 0 und 255 (abhängig von der ADSR-Kurve, siehe C2.5). Damit kann ebenfalls eine Steuerung der anderen Stimmen erfolgen, z.B. in Abhängigkeit vom Zustand der Decay- oder Release-Phase.

SI+27 = momentaner Zustand der Wellenform Stimme 3
SI+28 = momentaner Zustand der ADSR-Kurve Stimme 3

C8.1 „Ausschalten" der Stimme 3

Wird der Oszillator (Stimme) 3 ausschließlich zu Steuerzwecken genutzt, ist es nicht sinnvoll, daß er auch hörbar ist. Obwohl die Stimme 3 arbeitet (schwingt), klingt sie nicht, wenn das Bit 7 im Register SI+24 gesetzt wird.

POKE SI+24, 128+15

128 schaltet Stimme 3 auf „nicht hörbar" und die 15 ist das Volumen.

Die Anwendung der Stimme 3 zu Steuerwecken hat beachtliche Änderungen des Klanges zur Folge. So kann man z.B. den Zustand des Registers SI+27 verwenden, um ein Vibrato (schnelle Frequenzänderung) zu erzielen, indem man den Zustand von Oszillator 3 zur Frequenz einer anderen Stimme addiert. Als letztes Programm ein Beispiel zum Vibratoeffekt:

```
20 si = 54272
30 for l = si to si+24: poke l,0: next
40 poke si+ 3,  8:  rem pulsebreite s1
50 poke si+ 5, 41:poke si+6,89:rem adsr1
60 poke si+14,127:rem frequenz stimme 3
70 poke si+18, 16:rem dreieck  stimme 3
80 poke si+24,143:rem volumen
90 read nc,da
100 if nc = 0 then poke si+24,0:end
110 poke si+4,65
120 for t = 1 to da *2
```

```
130 fq=nc+peek(si+27)
140 fh% = fq/256: fl%=fq-fh%*2565
150 poke si,fl%:poke si+1,fh%
160 next
170 poke si+4,64
190 goto 90
1010 data 4817,2,5103,2,5407,2;8583,4
1020 data 5407,2,8583,4,5407,4,8583,12
1030 data 9634,2,10207,2,8583,2
1040 data 9634,4,10814,2,8583,2
1050 data 9634,4,8583,24,0,0
```

C9.0 Musik – ein Buch mit sieben Siegeln ?

Die musikalischen Fähigkeiten des C=64 sind im Homecomputerbereich ungeschlagen. Leider hat es COMMODORE versäumt, die Fähigkeiten mit „normalen" BASIC-Befehlen programmierbar zu machen. Ebenso wie in anderen Bereichen (SPRITE, hochauflösende Grafik, Diskette etc), zeigt sich auch hier eine nicht unerhebliche Schwäche. Es ist kaum möglich, mal eben einen Ton zu erzeugen; hierzu muß schon ein „bißchen" gePOKEt werden. Wer sich allerdings intensiv mit den musikalischen Fähigkeiten des C=64 auseinandersetzt, dem stehen Tür und Tor für einen „Supersound" offen.

DISKETTE

Die Diskettenstation ist das verbreiteste Geräte zur Datenspeicherung. Gerade die Kombination C=64 mit der Diskettenstation VC-1541 war durch ihren Preis von rund 1200,— DM die erste preiswerte „Komplettanlage". Der Vergleich der Ladezeiten eines 16 kByte langen Programms zwischen einer Diskettenstation und einem Kassettenrecorder ist schon beachtlich: der Recorder benötigt rund 5 1/2 Minuten, die Diskette hingegen nur 50 Sekunden!
Es soll an dieser Stelle nicht unerwähnt bleiben, daß die Diskettenstation VC-1541 recht langsam arbeitet. Die Übertragungsgeschwindigkeit ist im Gegensatz zu anderen Systemen bis zum Faktor 10 langsamer.
Da die Möglichkeiten einer Diskettenstation wesentlich umfangreicher und komfortabler sind, muß auf die „Floppy" genauer eingegangen werden.

D1.0 Was ist eine „Disk" ?

Eine „Disk" ist eine runde Plastikscheibe, die mit einem magnetisierbaren Stoff beschichtet und deren Größe abhängig vom benutzten System ist. Die VC-1541 benutzt die für Kleincomputer üblichen 5 1/4 Zoll Disketten. Auch der Recorder (Datasette) benutzt ein ähnliches Medium; nähmlich ein beschichtetes Band. Doch hier wird auch der grundlegende Unterschied klar. Auf einem Band können Daten nur hintereinander abgelegt werden, auf einer Scheibe kann räumlich in die Scheibe „gegriffen" werden.

Die Daten, die auf die Disk gelegt werden sollen, müssen selbstverständlich eine bestimmte Form besitzen. Als erstes müssen Bereiche auf der Disk gekennzeichnet werden, wo die Daten hingelegt werden dürfen. Es werden „Spuren" (Track) auf die Scheibe magnetisiert, die den Bereich kennzeichnen, wo Informationen gespeichert werden können. Die Anzahl der Spuren ist wieder abhängig vom benutzten System: bei der VC-1541 sind es 35 Spuren. Diese Spuren sind „Ringe", die in „Sektoren" geteilt werden. Da diese „Ringe" (Spuren) immer kleiner werden, je weiter sie zum Mittelpunkt kommen, haben sie eine unterschiedliche Sektorenzahl (17 bis 21 Sektoren). Sektoren werden auch Blöcke genannt, und jeder Block kann 256 Byte (Zeichen) aufnehmen.
Dieses „Vorbereiten" einer Disk ist immer notwendig und wird „Formatieren" genannt. Neugekaufte Disketten sind immer unformatiert und müssen vor dem Benutzen formatiert werden. Beim Formatieren wird aber noch mehr getan: es wird der Diskette ein Name gegeben, sowie die „Directory" angelegt. Die Directory ist das Inhaltsverzeichnis über die auf der Disk befindlichen Programme. Die Directory belegt die gesamte Spur 18 auf der Disk. In ihr können bis zu 144 Einträge abgelegt werden (das bedeutet, daß max. 144 Programme auf eine Disk gespeichert werden können).

Aus diesem Inhaltverzeichnis holt sich die Diskettenstation auch die Informationen, wo sich das Programm auf der Disk befindet (Spur/Sektor), wie lang es ist, usw. Da nun solche Dinge nicht von alleine passieren, muß irgendwo ein „Programm" liegen, welches die Diskettenstation befähigt,dies zu tun. Ein solches Programm nennt sich „Disk Operating System", kurz DOS. Das DOS verwaltet alles, was mit der Disk zu tun hat. Anders als bei fast allen andern Rechnertypen liegt das DOS nicht im Rechner (C=64), sondern direkt in der Diskettenstation. Der große Vorteil hierbei ist, daß das DOS keinen Speicherplatz im C=64 belegt und (vielleicht noch) extra geladen werden muß. Man spricht auch von einer „intelligenten" Diskettenstation.

D1.1 Der Weg vom Computer zur Floppy

Um Daten vom Rechner zur Diskettenstation zu leiten, muß ein Weg vom Rechner zur Diskettenstation eröffnet werden. Der Befehl, um einen „Weg" nach „draußen" zu eröffnen, heißt:

OPEN nummer

Die „Nummer" ist eine Zahl zwischen 1 und 127. Diese sogenannte „Kanalnummer" kann beliebig gewählt werden. Nach der Kanalnummer muß weiterhin die Nummer des Gerätes angegeben werden, zu dem die Daten gesendet werden sollen. Die Floppy ist „von Haus aus" auf die Gerätenummer 8 eingestellt. Mit

```
OPEN 1,8
```

ist ein Weg vom Computer zur Floppy geschaltet. In dem erwähnten DOS (in der Diskettenstation) ist es weiterhin möglich, bestimmte „Befehlsgruppen" anzusprechen. Mit

```
OPEN 1,8,15
```

wird der Befehlskanal 15 angesprochen. Über den Befehlskanal 15 wird direkt auf das DOS zugegriffen. Jetzt braucht nur noch mitgeteilt zu werden, was die Diskettenstation tun soll.

```
OPEN 1,8,15,"befehl"
```

D2.0 Formatieren

Um eine neue Diskette einsatzbereit zu machen, muß sie formatiert werden (siehe D1.0). Der Formatierbefehl lautet

```
NEW
```

Der Befehl kann, wie jeder andere Diskettenbefehl, mit dem ersten Zeichen ('N') abgekürzt werden. Beim Formatieren wird der Diskette ein Name und ein Identifizierungsmerkmal (ID) gegeben (das DOS erkennt anhand der ID, ob eine Diskette gewechselt worden ist). Das ID kann frei gewählt werden (Zwei Buchstaben/Zahlen).

```
OPEN 1,8,15,"NEW:diskettenname,id"    oder
OPEN 1,8,15,"N:diskettenname,id"
```

Das Formatieren dauert ca 1 1/2 Minuten (das Knacken ist „normal"). Zu bemerken ist, daß sich der C=64 sofort mit einem „READY" meldet und die Diskettenstation trotzdem weiterarbeitet (intelligente Floppy). Der Computer kann sofort weiter benutzt werden!

Wenn das Formatieren beendet worden ist (die rote Lampe ist wieder aus), muß der Kanal wieder geschlossen werden. Mit

CLOSE 1

wird der „Weg" 1 geschlossen.

D2.1 Lesen des Fehlerkanals

Tritt bei einer Diskettenoperation ein Fehler auf, fängt die rote Lampe an der Floppy zu blinken an und der Arbeitsvorgang wird abgebrochen. Da der Fehler nicht im C=64 aufgetreten ist, sondern in der Floppy, befindet sich die Fehlermeldung noch in der Diskettenstation. Mit folgendem Programm kann die Fehlermeldung ausgelesen werden.

```
10 open 1,8,15
20 input #1,a$,s$,d$,f$
30 close 1
40 print a$,s$,d$,f$
50 end
```

In A$ steht die Nummer des Fehlers, in S$ die Bezeichnung des Fehlers und in D$ und F$ die Spur und der Sektor, wo der Fehler aufgetreten ist.

An einem Beispiel soll verdeutlicht werden, was bei einem Fehler passiert. Geben Sie bitte

OPEN 1,8,15,"N TEST,01"

ein. Die rote Lampe an der Floppy fängt an zu blinken. Fehlermeldung (es fehlt der Doppelpunkt)! Starten Sie das oben gezeigte Programm mit RUN und es erscheint die Meldung:

34 SYNTAX ERROR 00 00

Die 34 ist die Fehlernummer, SYNTAX ERROR ist der zugehörige Fehlertext und Spur/Sektor sind 0, da diese Angaben nicht benötigt werden. Weiter ist festzustellen, daß die Fehlermeldelampe ausgegangen ist. Bei nochmaligem Start des Programms erhalten Sie die Meldung
„00 OK 00 00“. Alles in Ordnung.

D3.0 SAVE

Nachdem das Programm zum Auslesen des Fehlerkanals eingegeben ist, soll es auch gespeichert werden. Zum Speichern eines Programms wird der übliche „SAVE“-Befehl benutzt.

```
SAVE "name",8
```

Als Gerätenummer ist die „8“ für die Diskettestation anzugeben. Der Programmname darf 16 Zeichen umfassen.

D3.1 LOAD

Um das Programm wieder zu laden, wird der übliche „LOAD“- Befehl benutzt.

```
LOAD "name",8
```

Natürlich wieder mit der Geräteadresse 8 für die Diskettenstation.

D3.2 VERIFY

Mit „VERIFY“ wird der Inhalt des Computers (RAM) mit einem Programm auf der Diskette verglichen und auf Fehler überprüft. Sind beide Programme identisch (es liegen keine Fehler vor), meldet der Rechner ein „O.K“.

```
VERIFY "name",8
```

D3.3 SAVE „@:...." – Überschreiben

Wird versucht, das Programm ein zweites Mal mit demselben Namen auf die Diskette abzuspeichern, wird eine Fehlermeldung ausgegeben. „FILE EXISTS", File (Programm) ist vorhanden! Das ist ja auch irgendwie klar. Wie soll das DOS wissen, wenn zwei Programme denselben Namen tragen, welches gemeint ist? Aus diesem Grund müssen Programme, Files oder Daten auf der Diskette immer einen unterschiedlichen Namen tragen. Um ein vorhandenes Programm zu überschreiben (korrigieren), muß der Überschreibungsbefehl verwandt werden:

```
SAVE "@:name",8
```

Das „neue" Programm kommt nun an die Stelle des „alten" Programms. Das alte Programm ist damit überschrieben und gelöscht (natürlich kann man ein Programm auch mehrmals unter verschiedenen Namen speichern).

D4.0 Maschinenprogramme laden

Maschinenprogramme werden nicht wie BASIC-Programme direkt an den Anfang des BASIC-Speichers (2049) geladen (siehe auch E3.0), sondern absolut an die Stelle, von der sie später gestartet werden sollen (z.B. SYS 49152). Im gespeicherten Maschinenprogramm befinden sich in den ersten beiden Bytes die Anfangsadresse (lowbyte/highbyte), wo das nachfolgende Programm abgelegt werden soll. Aus diesem Grund ist der Ladebefehl für Maschinenprogramme etwas anders:

```
LOAD "name",8,1
```

Die „1" hinter der Gerätenummer bewirkt, daß das zu ladende Programm absolut geladen wird. Nach dem Laden muß noch ein „NEW" eingegeben werden, um alle „Zeiger" des Betriebssystems wieder in den ursprünglichen Zustand zu bringen. Gestartet wird das Programm dann mit

```
SYS Startadresse
```

D5.0 Laden der Directory

Das Inhaltsverzeichnis der Diskette (Directory) kann in den Rechner geladen und angezeigt werden. Leider wird hierbei ein im Rechner befindliches BASIC-Programm gelöscht !!! Mit dem Befehl LIST läßt sich die Directory anzeigen.

```
LOAD "$",8
LIST
```

Aus der Directory sind folgende Daten zu entnehmen:

1) Die erste Zeile gibt den Namen und das ID der Diskette wieder. Es folgt das „Aufzeichnungsformat" (bei C=64 immer 2A)

2) einzelne Programminformationen, die wie folgt aufgebaut sind:
   a. Die Länge des Programms in Blöcken
   b. Der Programmname (16 Zeichen max.)
   c. Der Programmtyp (PRG, SEQ, REL,USR)

Ein Programm mit der Bezeichnung „PRG" ist ein Programmfile, also ei BASIC- oder Maschinenprcgramm (die anderen Bezeichnungen werden später erklärt).

Aus der Anzahl der Blöcke läßt sich die ungefähre Länge des Programms ermitteln:

LÄNGE = BLÖCKE * 256

D5.1 Die JOKER

Das „Fragezeichen" (?) und der „Stern" (*) sind sogenannte JOKER. Mit den Jokern können verschiedene Gruppen aus der Directroy ausgewählt (selektiert) werden. Mit

```
LOAD "$A*",8
```

werden nur Files aus der Directory gelesen, die mit einem „A" anfangen. Wenn nur sequentielle Files aus der Directory gelesen werden sollen, kann eine Auswahl mit

LOAD "$*=S",8

stattfinden. Auch andere Filetypen können selektiert werden:

- *=S sequentielle Files
- *=P Programme
- *=R relative Files
- *=U User-Files

Mit Hilfe des Sterns (*) lassen sich auch Programme laden. Durch

LOAD "A*",8

wird das erste Programm, das mit „A" anfängt, in den Speicher geladen. Und mit

LOAD "*",8

wird das allererste Programm geladen. Durch das Fragezeichen (?) werden nichtrelevante Buchstaben gekennzeichnet. So wird ein Programm, dessen Name 4-stellig ist und mit „A" beginnt, durch den Befehl

LOAD "A???",8

geladen. Die Joker-Funktionen haben in der praktischen Anwendung nur dann eine Bedeutung, wenn sich sehr viele Files auf einer Diskette befinden, die nicht ohne weiteres überblickt werden können. Wichtig ist der Stern (*) allerdings beim Löschen.

D6.0 SCRATCH – Löschen

Files werden auf der Diskette mit dem Befehl „SCRATCH" gelöscht.

OPEN 1,8,15,"SCRATCH:file1,file2, ... oder
OPEN 1,8,15,"S:file1,file2,...."

Dabei können gleichzeitig mehrere Files gelöscht werden (durch Komma trennen), wobei nicht mehr als 40 Zeichen zwischen den Anführungsstrichen stehen dürfen. Mit dem Joker (*) können alle Programme auf der Diskette gelöscht werden:

OPEN 1,8,15,"S:*"

Seien Sie mit diesem Befehl sehr vorsichtig, damit nicht Programme gelöscht werden, die Sie noch benötigen!

D6.1 RENAME – Umbenennen

Um einem File einen anderen Namen auf der Diskette zu geben, bedient man sich des RENAME-Befehls:

OPEN 1,8,15,"RENAME:neuername=altername" oder
OPEN 1,8,15,"R:neuername=altername"

So können Namen von schon gespeicherten Files geändert werden.

D6.2 VALIDATE – Aufräumen

Die gesamte Diskette wird auf freie oder nicht ordnungsgemäß belegte Blöcke geprüft. Diese Blöcke werden nach VALIDATE als „frei" (nicht belegt) gekennzeichnet. Es ist durchaus möglich, daß nach

OPEN 1,8,15,"VALIDATE" oder
OPEN 1,8,15,"V"

mehr Blöcke als „frei" gekennzeichnet sind, als vor der Durchführung des Befehls.

D6.3 COPY – Kopieren auf der Disk

Es ist möglich, eine Datei auf der Diskette zu kopieren. Da es aber nicht sinnvoll ist, auf derselben Disk ein Programm zu kopieren (kopieren auf eine andere Disk wäre wesentlich sinnvoller), hat der Befehl COPY in dieser Hinsicht fast keine Bedeutung. Weiterhin hat der COPY-Befehl die Funktion, mehrere sequentielle Files zu einem neuen, gesamten File zusammenzufassen.

```
OPEN1,8,15,"COPY:neuerfile=alterfile1,alterfile2."
OPEN1,8,15,"C:neuerfile=alterfile1,alterfile2,.."
```

So werden mehrere Dateien zu einer neuen zusammengefaßt, z.B. kann aus Monatsabrechnungen eine Quartalsdatei erstellen werden, etc.

D6.4 INITIALIZE – Initialisieren

Die ID sollte bei allen Disketten unterschiedlich gewählt werden (siehe C2.0). Ist dies nicht der Fall, kann mit

```
OPEN 1,8,15,"INTIALIZE"
OPEN 1,8,15,"I"
```

der Floppy mitgeteilt werden, daß sich eine andere Disk (mit derselben ID) im Laufwerk befindet.

D7.0 Sequentielle Datei

Um nur Programme abzuspeichern und sie wieder einzuladen, ist die Diskettenstation viel zu schade. Der wesentliche Vorteil einer Diskettenstation liegt im schnellen Zugriff auf Daten. In einer relativen Datei (siehe D16.1) kann in Sekundenschnelle auf viele tausend Daten zugegriffen werden. Bei einer Datasette ist nur eine sequentielle Datei möglich (die Daten liegen alle hintereinander). Wird die „letzte" Eintragung gesucht, muß die gesamte Datei von „vorne" durchsucht werden. Beim Diskettenlaufwerk ist eine sequentielle Datei ebenfalls möglich und bietet sich auf Grund der einfachen Handhabung für kleine Datenmengen an.

D7.1 Eröffnen einer sequentiellen Datei

Bei jeder Datei, die außerhalb des Computers liegt (z.B. Floppy), muß in einer bestimmten Weise vorgegangen werden.

1. Datei eröffnen
2. Daten übertragen (Speicher-Rechner, Rechner-Speicher)
3. Datei schließen

Nur wenn eine Datei ordnungsgemäß geschlossen worden ist (CLOSE), sind alle Daten auf der Diskette gespeichert.

Eine sequentielle Datei besteht aus mehreren Daten bzw. Datensätzen. Datensätze sind in Felder aufgeteilt, wobei jedes Feld wieder Daten oder Datensätze enthalten kann. Wenn eine Datei nur die Zahlen von 1 bis 99 enthalten soll, so sind 99 Daten vorhanden, wobei das Datenfeld immer eine Zahl enthält. Bei einer Telefondatei ist es ein bißchen anders. Es wird die Telefonnummer, der Name und die Adresse abgespeichert. Diese 3 Daten (Nummer, Name, Adresse) bilden einen Datensatz mit 3 Feldern. Bei einem Telefonverzeichnis von 100 Teilnehmern werden 100 Datensätze benötigt.

```
Datensatz (1): FELD 1    FELD 2    FELD 3
Datensatz (n): FELD 1    FELD 2    FELD 3
```

Die Datensätze liegen wiederum auf der Diskette hintereinander. Jedes Feld und damit jeder Datensatz kann deshalb auch eine unterschiedliche Länge besitzen. Jedes Feld kann im Programm einer Variablen zugewiesen (INPUT) und in die Datei geschrieben werden (PRINT). Da die Datensätze eine unterschiedliche Länge aufweisen können, ist der genaue Ort eines Feldes nicht definiert. Um Datensätze wiederzufinden, müssen die Felder in einer vorher vereinbarten Reihenfolge liegen. Z.B. name1-nummer1, name2-nummer2, etc. Es darf nicht irgendwo nummer9-name9 liegen !!!.
Um eine Datei zu eröffnen, ist natürlich wieder ein Weg vom Computer zur Floppy notwendig.

```
OPEN 1,8,2,"name,typ,opt"
```

Die Sekundäradresse kann einen Wert zwischen 2 und 14 annehmen. Damit wird der Bereich für die Ein- und Ausgabe der Daten angesprochen. (15 ist für

das Betriebssystem (DOS), 0 und 1 für das Speichern von Programmen reserviert). Weiter wird der Datei ein Name gegeben; es wird festgelegt, welche Art von Datei es sein soll und ob diese Datei beschrieben oder gelesen werden soll. Die Filetypen unterscheiden sich wie folgt:

P - Programm
U - User-File
R - relative Datei
und S - sequentielle Datei
M - siehe D12.0

Folgende Kennzeichen besagen, was mit der Datei passieren soll:

W - Schreiben (write)
R - Lesen (read)
A - Anhängen (append)

D7.2 Beschreiben einer Datei

Die sequentielle Datei mit dem Namen „TEST1" wird eröffnet und anschließend werden die Zahlen 1 bis 99 (a) in die Datei geschrieben:

```
10 open 1,8,2,"test1,s,w"
20 for a = 1 TO 99
30 print#1, a
40 next a
50 close 1
60 end
```

Bei dem Übermittlungsbefehl „PRINT" muß natürlich die „Wegnummer" (in diesem Fall 1) angegeben sein (es wird auf die Diskette gePRINTet). Zum Programmschluß ist die Datei zu schließen (wichtig!).

Nach dem Starten des Programms werden die Zahlen 1 bis 99 in die Datei „TEST1" geschrieben.

D7.3 Lesen aus einer Datei

Die Daten, die sich in der Datei TEST1 befinden, sollen wieder in den Speicher eingelesen werden.

```
10 open 1,8,2,"test1,s,r"
```

Das „r" (für Lesen) kennzeichnet, daß die sequentielle Datei (s) mit dem Namen TEST1 gelesen werden soll. Um Daten aus einer Datei zu holen, benutzt man den INPUT # -Befehl. Hinter dem Nummernzeichen (#) folgt die Kanalnummer, wie sie beim OPEN-Befehl bezeichnet worden ist (in diesem Fall 1). Da bekannt ist, daß 99 Daten folgen, werden 99 Daten über eine Schleife eingelesen und ausgedruckt.

```
20 for i = 1 TO 99
30 input#1, a
40 print a,
50 next i
60 close 1
70 end
```

Es werden alle 99 Daten auf dem Bildschirm gePRINTet. Sollen die Daten im Programm weiter verwendet werden, so ist es natürlich möglich, sie i in Array zu legen.

```
10 dim a(100)
20 open 1,8,1,"test1,s,r"
30 for i = 1 to 99
40 input#1, a(i)
50 next i
60 close 1
70 end
```

Die Daten befinden sich im Array A(...) und können weiterbearbeitet werden.

D7.4 Anhängen von Daten

In der Datei TEST1 befinden sich 99 Daten. Es sollen aber noch 10 weitere Daten angehängt werden. (Z.B. die Zahlen von 1000 bis 1009).

```
10 open 1,8,2,"test1,s,a"
20 for a = 1000 to 1009
30 print#1, a
40 next a
50 close 1
60 end
```

In dem OPEN-Befehl steht das „A" für Anhängen (append). Nach Durchführung des Programms befinden sich (99 + 10 =) 109 Daten in der Datei „TEST1". Das Anhängen von Daten an eine sequentielle Datei ist nur bei der Diskette, nicht aber bei der Datasette möglich.

Es wurden 10 Daten angehängt, die vorderen 99 Daten blieben unverändert. Um alle 109 Daten einzulesen, muß das Programm „LESEN" entsprechend geändert werden. (for i = 1 to 109). Das bedeutet, daß Sie immer wissen müssen, wieviele Daten vorhanden sind. Doch was machen Sie, wenn Sie das nicht wissen ??

D7.5 Statusvariable (ST)

3 Variablen benutzt der C=64 selber. Es sind die Variablen

	TI	Interne Uhr (zählt 1/60 Sekunden)
	TI$	Uhr (6-stelliger String "HHMMSS")
und	ST	Statusvariable für Ein/Ausgabe-Operationen

Treten bei einer Ein/Ausgabe-Operation Probleme auf, steht in der Variable ST ein bestimmter Wert. Weiterhin steht in der Variable ST der Wert „64", wenn das Dateienende erreicht ist. So läßt sich mit einer einfachen Prüfung feststellen, ob das Ende einer Datei erreicht ist.

```
IF ST = 64 THEN CLOSE 1 : END
```

Ist die Anzahl der vorhandenen Daten nicht bekannt, so ist das Dateienende also mit der Statusvariablen festzustellen.

```
10 open 1,8,2,"test1,s,r"
20 input#1, a
30 print a
40 if st = 64 then close 1: end
50 goto 20
```

D7.6 Verändern von Dateien

Daten einer Datei müssen geändert werden können. Bei einer sequentiellen Datei geschieht dieses in folgender Reihenfolge:

1. Daten in den Rechner einlesen (Array)
2. Daten im Rechner ändern
3. Gesamte, geänderte Datei abspeichern (die alte Datei überschreiben).

Ein kleines Beispielprogramm, wie eine Datei eingelesen, Daten geändert und die geänderte Datei wieder abgespeichert wird, soll das Prinzip verdeutlichen. In den Zeilen 10 bis 70 wird die vorhandene Datei mit dem Namen TEST1 in das Array A(...) eingelesen, wobei die Variable „I" als Zähler dient. Das Ende der Datei wird mit der Statusvariablen (ST) abgefragt. Das Array A(...) wird auf 200 Daten festgelegt (Maximal 200 Daten). Nach dem Einlesen steht in „I" die Anzahl der vorhandenen Daten. Es wird gefragt, welches Datum ('Index-nummer') geändert werden soll und überprüft, ob die Eingabe in dem Bereich 1 bis „I" liegt (Zeile 100 bis 110). Der Inhalt wird ausgedruckt und mit der nächsten Eingabe geändert. Sind alle Änderungen durchgeführt, wird die geänderte Datei über die alten Daten geschrieben. Mit dem Klammeraffen (@) wird das Überschreiben gekennzeichnet. Damit liegt die geänderte Datei an der Stelle der alten Datei.

```
10 rem ************** einlesen der daten ***************
20 dim a (200)
30 open 1,8,2,"test1,s,r"
40 i = i+1: rem zaehler
50 input#1, a(I)
60 if st = 64 then close 1: goto 90
```

```
70 goto 40
80 rem ************ ende einlesen der datei *************
90 print "es sind";i;" daten vorhanden"
100 input "welches datum aendern ";j
110 if j<1 or j>i then 100
120 print a(j)
130 input "welchen wert";a(j)
140 print"(w)eiter aendern  (e)nde  ?"
150 get a$: if a$ = "" then 150
160 if a$ = "w" then 100
170 if a$ = "e" then 190
180 goto 150
190 rem *********** ueberschreiben der datei ************
200 open 1,8,15,"@:test1,s,w"
210 for j = 1 to i
220 print#1, a(j)
230 next j
240 close 1
250 end
```

D8.0 Datensätze

Nachdem bisher nur nummerische Daten in einer Datei verwaltet worden sind, soll nun die Handhabung von Datensätzen behandelt werden (siehe D7.1). Eine Telefondatei besteht z.B aus folgenden Feldern:

FELD1 = Name
FELD2 = Ort
FELD3 = Straße
FELD4 = Telefonnummer

Der Datensatz besteht dann aus den Feldern 1 bis 4. Für jeden Eintrag (Datensatz) sind also 4 zusammenhängende Felder notwendig.

DATENSATZ 1 =FELD1/1 + FELD2/1 + FELD3/1 + FELD4/1
DATENSATZ n =FELD1/n + FELD2/n + FELD3/n + FELD4/n

Die Datensätze werden dann in der sequentiellen Datei hintereinander abgelegt:

DATENSATZ1 / DATENSATZ2 / DATENSATZ3 / DATENSATZn

Jedem Feld wird eine Variable zugeordnet, die mit INPUT # und PRINT # gelesen oder beschrieben werden kann. Nach jeder PRINT # Anweisung wird außerdem automatisch noch ein RETURN (CHR$(13)) in die Datei geschrieben. Mit einer INPUT # Anweisung werden die Daten bis zum RETURN in eine Variable übernommen. Dabei ist zu beachten, daß bei

```
PRINT#1, A
```

das „RETURN" hinter A gesetzt wird. Bei der Eingabe mehrerer Variablen hingegen

```
PRINT#1, A,B,C,D
```

hinter dem „D", also der letzten Variablen in der PRINT # Zeile. Werden die Daten A, B, C, und D nun eingelesen,

```
INPUT#1, A,B,C,D
```

stehen sie in den entsprechenden Variablen. Wenn allerdings mit dem Befehl

```
INPUT#1, A
```

versucht wird, nur einen Wert einzulesen, entsteht ein Fehler, da bei INPUT # die Zeichen bis zum RETURN (und dies steht hinter dem „D") eingelesen werden! Aus diesem Grund ist es notwendig, die Daten wieder so einzulesen, wie sie in die Datei geschrieben worden sind.

```
PRINT#1, A,B,C
INPUT#1, A,B,C

PRINT#1, A
PRINT#1, B
INPUT#1, A
INPUT#1, B
```

```
70 goto 40
80 rem ************ ende einlesen der datei *************
90 print "es sind";i;" daten vorhanden"
100 input "welches datum aendern ";j
110 if j<1 or j>i then 100
120 print a(j)
130 input "welchen wert";a(j)
140 print"(w)eiter aendern  (e)nde  ?"
150 get a$: if a$ = "" then 150
160 if a$ = "w" then 100
170 if a$ = "e" then 190
180 goto 150
190 rem *********** ueberschreiben der datei ************
200 open 1,8,15,"@:test1,s,w"
210 for j = 1 to i
220 print#1, a(j)
230 next j
240 close 1
250 end
```

D8.0 Datensätze

Nachdem bisher nur nummerische Daten in einer Datei verwaltet worden sind, soll nun die Handhabung von Datensätzen behandelt werden (siehe D7.1). Eine Telefondatei besteht z.B aus folgenden Feldern:

FELD1 = Name
FELD2 = Ort
FELD3 = Straße
FELD4 = Telefonnummer

Der Datensatz besteht dann aus den Feldern 1 bis 4. Für jeden Eintrag (Datensatz) sind also 4 zusammenhängende Felder notwendig.

DATENSATZ 1 =FELD1/1 + FELD2/1 + FELD3/1 + FELD4/1
DATENSATZ n =FELD1/n + FELD2/n + FELD3/n + FELD4/n

Die Datensätze werden dann in der sequentiellen Datei hintereinander abgelegt:

DATENSATZ1 / DATENSATZ2 / DATENSATZ3 / DATENSATZn

Jedem Feld wird eine Variable zugeordnet, die mit INPUT # und PRINT # gelesen oder beschrieben werden kann. Nach jeder PRINT # Anweisung wird außerdem automatisch noch ein RETURN (CHR$(13)) in die Datei geschrieben. Mit einer INPUT # Anweisung werden die Daten bis zum RETURN in eine Variable übernommen. Dabei ist zu beachten, daß bei

```
PRINT#1, A
```

das „RETURN" hinter A gesetzt wird. Bei der Eingabe mehrerer Variablen hingegen

```
PRINT#1, A,B,C,D
```

hinter dem „D", also der letzten Variablen in der PRINT # Zeile. Werden die Daten A, B, C, und D nun eingelesen,

```
INPUT#1, A,B,C,D
```

stehen sie in den entsprechenden Variablen. Wenn allerdings mit dem Befehl

```
INPUT#1, A
```

versucht wird, nur einen Wert einzulesen, entsteht ein Fehler, da bei INPUT # die Zeichen bis zum RETURN (und dies steht hinter dem „D") eingelesen werden! Aus diesem Grund ist es notwendig, die Daten wieder so einzulesen, wie sie in die Datei geschrieben worden sind.

```
PRINT#1, A,B,C
INPUT#1, A,B,C

PRINT#1, A
PRINT#1, B
INPUT#1, A
INPUT#1, B
```

Werden Strings als Variablen in eine Datei geschrieben, sieht der Befehl etwas anders aus. Numerische Variablen werden nur mit einem Komma getrennt, Stringvariablen mit einem Komma in Anführungsstrichen.

PRINT # 1, A$",''B$'',''C$'',''

Eingelesen werden sie dann wieder normal.

INPUT # 1, A$,B$,C$

Somit ist es möglich, Datensätze in eine Datei zu schreiben und sie aus Dateien zu lesen. Datenfelder werden in der Regel Arrays zugeschrieben. Z.B

```
100 FOR I = 1 TO 10
110 INPUT#1, A$(I), B$(I)
120 NEXT I
```

Der Datensatz hat 2 Felder (A$(..), B$(..)). In den zusammenhängenden Arrayfeldern (1, 2, 3,..n) liegen die Daten der Felder: in A$(..) der Nachname, in B$(..) der Vorname.

DATENSATZ	FELD1	FELD2
1	A$(1)	B$(1)
2	A$(2)	B$(2)
3	A$(3)	B$(3)

Anhand der folgenden Telefon-Datei werden die Möglichkeiten einer sequentiellen Datei zusammengefaßt. Das Programm ist nicht fertig. Aus dem MENUE ist zu erkennen, daß einige Unterprogramme fehlen. Hier ist Ihrer Kreativität keine Grenze gesetzt.
Beachtet werden sollte die „Eröffnung" des Fehlerkanals gleich am Anfang des Programmes. Hierdurch werden Änderungen der Statusvariablen durch den Fehlerkanal vermieden. Der Fehlerkanal wird nach jeder Diskettenoperation abgefragt. Treten Fehler auf, stoppt das Programm.

Die Grenze der aufzunehmenden Daten liegt bei der Arraygröße (200 Datensätze). Die Variablen sind entsprechend dimensioniert. Werden mehr als 200 Datensätze eingelesen, tritt ein Fehler auf (sichern Sie das Programm so, daß es hier zu keiner Fehlermeldung des Systems kommt!)

D9.0 Telefon-Datei

```
10 rem **************** telefon  datei ****************
20 poke 53280, 0: poke 53281,0: poke646,5
30 dim n$(200),o$(200),s$(200),t$(200)
40 open15,8,15:rem fehlerkanal offen
50 n$(0)="thoma/manfred walter"
60 o$(0)="hamburg 93/2102"
70 s$(0)="ernastrasse 10"
80 t$(0)="7522748/040"
90 print chr$(147);
100 print "------------"
110 print "telefondatei"
120 print "------------"
130 print:print:print
140 print " * (n)eue oder (a)lter datei einlesen"
150 get in$: if in$ = "" then 150
160 if in$="n" then 190
170 if in$="a" then 245
180 goto 150
190 print:print
200 input " * name der neuen datei ";da$
210 open 1,8,2,da$+",s,w"
220 gosub 60000
230 print#1,n$(0),o$(0),s$(0),t$(0)
240 close1:goto 250
245 print:print:print:input " * name der alten datei ";da$
250 open1,8,2,da$+",s,r"
255 gosub 60000
260 input#1,n$(z),o$(z),s$(z),t$(z)
270 if st <> 64 then z=z+1: goto 260
280 close1
1000 rem **************** m e n u e ********************
1010 print chr$(147):print " * es sind";z;"datensaetze
     vorhanden"
1020 print:print
1030 print "       * (1) liste aller daten ......"
1040 print "       * (2) suchen nach telefonnr..."
1050 print "       * (3) suchen nach namen......."
```

```
1060 print "        * (4) suchen nach ort........."
1070 print "        * (5) eingabe neuer daten....."
1080 print "        * (6) loeschen von daten......"
1090 print "        * (7) aendern von daten......."
1100 print "        * (8) loeschen von dateien...."
1110 print "        * (9) programmende............"
1120 print:print:print "     *        bitte waehlen sie !"
1130 get in$: if in$ = "" then1130
1140 in = val(in$)
1150 if in <1 or in >9 then 1130
1160 on in gosub 10000,15000,20000,25000,30000,35000,40000,
     45000 50000
1170 goto 1000
1180 rem ***********************************************
1190 :
1200 :
10000 rem ************** liste aller daten ****************
10010 print chr$(147);
10020 print "-----------------"
10030 print "liste aller daten"
10040 print "-----------------"
10050 print:print
10060 for j =0 to z
10065 print "datensatz :";j
10070 print "name      :";n$(j)
10080 print "ort       :";o$(j)
10090 print "strasse   :";s$(j)
10100 print "telefonnr.:";t$(j)
10110 print
10120 next
10130 get in$: if in$= "" then 10130
10140 return
30000 rem ************** eingabe neuer daten **************
30010 print chr$(147);
30020 print "-------------------"
30030 print "eingabe neuer daten"
30040 print "-------------------"
30050 print:print:print
30060 z=z+1
```

```
30070 print " * datensatznummer";z
30080 print:print
30090 input "name      :";n$(z)
30100 input "ort       :";o$(z)
30110 input "strasse   :"s$(z)
30120 input "telefonnr.:";t$(z)
30130 print:print:print " * (w)eiter eingeben oder (m)enue"
30140 get in$:if in$ = "" then 30140
30150 if in$ = " " then return
30160 if in$ = "w" then 30000
30170 goto 30140
50000 rem *************** programm ende *****************
50010 print chr$(147);
50020 print "--------------------------"
50030 print "daten werden abgespeichert"
50040 print "--------------------------"
50050 print:print
50060 open1,8,2,"@:"+da$+",s,w"
50070 gosub 60000
50080 for j=0 to z
50090 print#1,n$(j)","o$(j)","s$(j)","t$(j)
50130 next
50140 close 1:close 15: end
60000 rem *************** lesen des fehlerkanals **********
60010 :
60020 input#15,a$,b$,c$,d$
60030 :
60040 if a$ ="00" then return
60050 print:print:print
60060 print " * diskettenfehler !!!!!"
60070 print:print "* ";a$;"  ";b$,c$;"  ";d$
60080 close1:close15
60090 end
60100 rem ************************************************
```

D10.0 Datensätze über 88 Zeichen

Mit einem INPUT-Befehl können maximal 88 Zeichen eingelesen (übergeben) werden. Bei Datensätzen, deren Zeichenanzahl mehr als 88 Zeichen beträgt, muß die GET - Funktion benutzt werden. Mit GET # wird jeweils ein Zeichen aus der Datei gelesen. Die Zeichen werden nacheinander in einer Stringvariablen abgelegt, bis ein RETURN (CHR$(13)) eingelesen wird (Ende Datensatz).

```
100 open 1,8,2,"test1,s,r"
110 wo$ = "": rem loeschen der ausgabevariablen
120 get#1, a$: rem ein zeichen lesen
140 if a$ = chr$(13) then 200: rem ende pruefen
150 wo$ = wo$ + a$
160 if st = 64 then close1: end: rem dateiende
170 goto 120: rem naechstes zeichen lesen
200 print wo$: rem ausgabe des datenfeld
210 goto 110: rem neues datenfeld
```

Der Nachteil der GET # - Leseschleife liegt in der Lesegeschwindigkeit und in der etwas komplizierteren Handhabung. Umgangen werden kann das Problem, indem die Datensätze in mehrere Teile zerlegt und nach dem Einlesen neu zusammengefaßt werden (siehe D18.3).

D11.0 USER-Files

User-Files sind sequentielle Files, die in der Directory nicht mit SEQ, sondern mit USR bezeichnet sind. Hierbei handelt es sich nicht um Dateien im eigentlichen Sinne. Meistens wird dieser Filetyp benutzt, um Ausgaben, die normalerweise auf dem Bildschirm erfolgen (Directory, Listing), auf die Diskette umzuleiten. Als Beispiel soll die Directory als „Datei" auf die Diskettte gelegt werden.

Laden der Directory

```
load "$",8
```

Öffnen und LISTen

```
open1,8,2,"liste,u,w"
cmd 1
list
close 1
```

„CMD Nummer" ist eine Art „Umleitungsbefehl". Alle Funktionen, die normalerweise auf dem Bildschirm ablaufen, werden nun auf das im OPEN Statement angesprochene Gerät (hier die Diskette) umgeleitet. Das „LISTing" der Directory liegt nun als USER-File auf der Disk.

Die „Datei" kann natürlich wieder ausgelesen werden. Da eine „Zeile" länger als 88 Zeichen sein kann, erfolgt das Einlesen mit GET #, bevor es mit PRINT gelistet wird.

```
100 open1,8,2,"liste,u,r"
110 get#1, a$
120 print a$;
130 if st = 64 then close 1: end
140 goto 110
```

D12.0 CLOSE

Werden Daten vom Computer zur Diskettenstation geleitet, werden sie nicht „sofort" auf die Disk geschrieben. Sie werden zuerst in einem Speicher der Floppy zwischengespeichert (Puffer). Wenn der Puffer gefüllt ist, werden die Daten auf die Disk geschrieben. Beim Schließen einer Datei (CLOSE) werden die sich noch im Puffer befindlichen Daten auf die Disk geschrieben. Ist die Datei nicht mit CLOSE geschlossen worden, fehlen die Daten, die sich noch im Puffer befinden. Weiterhin ist diese Datei nicht ordnungsgemäß geschlossen und in der Directory mit

```
0 NAME          *SEQ
```

bezeichnet (Stern!).
Beim Versuch, diese Datei im Read-Modus (R) zu lesen, wird eine Fehlermeldung vom DOS ausgegeben (WRITE FILE OPEN). D.h., nicht ordnungsgemäß geschlossene Dateien können nicht mehr gelesen werden !!!.

Um solche Dateien doch noch zu „retten", bietet der Modus „M" die Möglichkeit, sie einzulesen. Sinnvollerweise sollten die Daten dann in eine andere Datei geschrieben werden, die ordnungsgemäß geschlossen wird.

```
100 open 1,8,2,"schlechte,s,m"
110 open 2,8,2,"gute,s,w"
120 input#1, a$
130 print#2, a$
140 if st = 64 then close 1:
150 goto 120
```

Die „Schlechte"-Datei ist dann mit

open 1,8,15,"s:schlechte"
close 1

zu löschen.

Mit der Diskettenstation VC 1541 können zur selben Zeit 3 Dateien geöffnet sein (3 sequentielle Dateien!). Die rote Lampe an der Diskettenstation wird erst nach dem Schließen der letzten Datei gelöscht.

D13.0 Zusammenfassung der sequentiellen Datei

Die sequentielle Datei bietet mit geringem Programmieraufwand die Möglichkeit, kleinere Datenmengen zu verwalten. Bei einem ca. 8 kByte langem Programm können immerhin noch 30 kByte an Daten verwaltet werden. Bei einer Adress-Datei reicht das für rund 350 Datensätze. Die Größe der Datei ist nur abhängig von der Speicherkapazität des Rechners. Wenn sich alle Daten im Speicher des Computers befinden, ist der Datenzugriff sehr schnell (schneller als alle anderen Möglichkeiten des Rechners). Sobald aber größere Datenmengen anfallen, ist eine sequentielle Datenspeicherung nicht mehr sinnvoll anwendbar. Um große Datenmengen zu verwalten, die nicht alle in den Speicher des Computers passen, ist die relative Datenspeicherung notwendig.

D14.0 Disketten-Monitor

Für die weitere Bearbeitung der Diskette ist es notwendig, direkt auf sie Einfluß nehmen zu können. Mit Hilfe eines Disketten-Monitors ist es möglich, jeden Block der Diskette in den C=64 zu laden, ihn zu ändern und wieder auf die Diskette zurückzuschreiben. Für diejenigen von Ihnen, die keinen eigenen Disketten-Monitor besitzen, ist ein kleiner Monitor aufgelistet. Aufgrund der besseren Darstellung werden alle Zahlen als hexadezimale Zahlen ausgegeben (siehe G1.4).

```
10 rem****************** disk-monitor ******************
20 poke 53280,0:poke53281,0
30 open 1,8,15
40 open 2,8,2,"#"
50 he$="0123456789abcdef"
60 print chr$(147)
70 print " --- disk-mon ---"
80 print: print
90 input " * befehl :";be$
100 if left$(be$,1)= "x" then close1:close2:end
110 if len(be$)<>5 then 160
120 if left$(be$,1)= "w" then 190
130 if left$(be$,1)= "r" then 300
140 if left$(be$,1)= "a" then 640
150 if left$(be$,1)= "l" then 400
160 print chr$(145);:goto 90
170 rem************************************************
180 :
190 rem****************** block write ******************
200 if s=1 then 230
210 print " * kein block vorhanden !!"
220 goto 90
230 h$=mid$(be$,2,2):gosub 920:sp=de
240 h$=mid$(be$,4,2):gosub 920:se=de
250 print#1," u2 2 0";sp;se
260 gosub 850
270 goto 90
280 rem************************************************
290 :
```

```
300 rem****************** block read *********************
310 h$=mid$(be$,2,2):gosub 920:sp=de
320 h$=mid$(be$,4,2):gosub 920:se=de
340 print#1," u1 2 0";sp;se
350 gosub 850
360 if f1$<>"00" then goto 90
370 s=1:goto 90
380 rem**************************************************
390 :
400 rem******************* block list ********************
410 if s=1 then 440
420 print " * kein block vorhanden !!"
430 goto 90
440 h$=mid$(be$,2,2):gosub 920:an=de
450 h$=mid$(be$,4,2):gosub 920:en=de
460 if an<0 or an>255 or en< an or en>255 then 90
470 print#1," b-p 2";an
480 for j  = an to en step 8
490 de=j:gosub 1020
500 print " * ";chr$(18);h$;chr$(146);" ";
510 for jj = j to j+7
520 get#2,a$:
530 de=asc(a$+chr$(0))
540 gosub 1020
550 print h$;" ";
560 if de<32 then de=46:goto 580
570 if de>127 and de<160 then de=46
580 as$=as$+chr$(de)
590 next jj:printchr$(18);as$:as$=""
600 next j
610 goto 90
620 rem**************************************************
630 :
640 rem******************** block aendern ****************
650 if s=1 then 680
660 print " * kein block vorhanden !!"
670 goto 90
680 h$=mid$(be$,2,2):gosub 920:an=de
```

```
690 h$=mid$(be$,4,2):gosub 920:en=de
700 print: print
710 if an<0 or an>255 or en<an or en>255 then 90
720 for j=an to en
730 print#1," b-p 2";j
740 de=j:gosub 1020
750 print " * ";chr$(18);h$;chr$(146);":    ";
760 get#2, a$:de=asc(a$+chr$(0))
770 gosub 1020:printh$,
780 input h$:gosub 920
790 print#1," b-p 2";j
800 print#2, chr$(de);
810 next
820 goto 90
830 rem*************************************************
840 :
850 rem****************** fehlerkanal *****************
860 input#1, f1$,f2$,f3$,f4$
870 if f1$="00" then return
880 print " * ";f1$;",";f2$;",";f3$;",";f4$
890 return
900 rem*************************************************
910 :
920 rem****************** hexa - dezi ******************
930 de=0:for i=1 to 2
940 x=asc(mid$(h$,i,1))
950 if x<58 then x=x-48:goto 970
960 x=x-55
970 de=de+x*16^(2-i)
980 next i
990 return
1000 rem************************************************
1010 :
1020 rem***************** dezi - hexa ******************
1030 h$="":d=int(de/16):h$=mid$(he$,d+1,1):d=de-d*16
1040 h$=h$+mid$(he$,d+1,1)
1050 return
1060 rem************************************************
```

D14.1 R – Lesen eines Blockes

Mit „R“ wird ein Block der Diskette in den Computer eingelesen. Format des Befehls:

R spur sektor

Die Angaben für die Spur und den Sektor sind hexadezimal anzugeben!! Alle Ein- und Ausgaben des Disketten-Monitors erfolgen im hexadezimalen System. Beispiel: es soll die Spur $12, Sektor $01 eingelesen werden.

R1201

Hexadezimale Zahlen werden zur Unterscheidung von dezimalen Zahlen mit einem davorgestellten „$“ (Dollarzeichen) gekennzeichnet.

D14.2 L – Listen des Blockes

Nachdem ein Block eingelesen worden ist, wird er durch den Befehl „L“ auf dem Bildschirm angezeigt. Nach dem „L“ ist anzugeben, welcher Bereich des Blocks gelistet werden soll.

L anfang ende

Um den gesamten Block zu listen (256 Byte), ist die Eingabe:

L00FF

notwendig (die Angaben wieder in Hex!).

D14.3 A – Ändern von Daten

Einzelne Bytes des Blocks können mit dem Befehl „A“ geändert werden. Nach dem „A“ wird der zu ändernde Bereich angegeben.

A anfang ende

Um die Bytes $00 bis $0F zu ändern, muß

A000F

eingegeben werden. Es wird der alte Inhalt angezeigt und auf den neuen Wert (Hex!) gewartet. Wird nur die RETURN-Taste betätigt, bleibt der alte Inhalt des (der) Bytes unverändert.

D14.4 W – Zurückschreiben

Ist ein Block mit R eingelesen worden, kann er angezeigt (L) und bearbeitet (A) werden. Um den geänderten Block wieder auf die Disk zu schreiben, ist der Befehl „W" zu benutzen.

W spur sektor
W1201

Bei dem W (write) - Befehl ist besonders darauf zu achten, daß

1. der Block an die richtige Stelle geschrieben wird und
2. die alten Daten des überschriebenen Blocks nicht wieder „zurückgeholt" werden können!

D14.5 X – Programmende

Mit der Eingabe von „X" wird das Programm beendet. Beim Arbeiten mit dem Disketten-Monitor ist äußerste Vorsicht geboten. Wird ein Block an eine falsche Stelle zurückgeschrieben, hat dieses meist katastrophale Auswirkungen. Bitte beachten Sie das und legen sich vorsorglich eine Sicherheitskopie der Diskette an.

D15.0 Directory

Wie schon in D1.0 beschrieben, benutzt die Diskettenstation die gesamte Spur 18 für das Inhaltsverzeichnis (Directory). In der Spur 18 Sektor 0 steht die sogenannte „BAM“ (Block-Availability-Map). In der BAM werden Blöcke als frei oder belegt gekennzeichnet. Das DOS greift hierauf zurück, um zu prüfen, ob noch genügend Platz für ein Programm vorhanden ist etc. In der Spur 18 Sektor 1 befindet sich der erste Eintrag der Directory. Mit

R1201

wird dieser Block in den Disketten-Monitor eingelesen. Nach der Eingabe von

L0090

erhalten Sie folgenden Ausdruck auf dem Bildschirm (natürlich abhängig vom Inhalt der von Ihnen benutzten Diskette !!!):

```
>:00  12 04 82 14 04 42 55 43 .....buc
>:08  48 44 52 55 43 4b a0 a0 hdruck
>:10  a0 a0 a0 a0 a0 00 00 00      ...
>:18  00 00 00 00 00 00 19 00 ........
>:20  00 00 82 11 00 42 55 43 .....buc
>:28  48 57 45 52 54 a0 a0 a0 hwert
>:30  a0 a0 a0 a0 a0 00 00 00      ...
>:38  00 00 00 00 00 00 19 00 ........
>:40  00 00 82 13 05 4d 41 53 .....mas
>:48  54 45 52 44 52 55 43 4b terdruck
>:50  a0 a0 a0 a0 a0 00 00 00      ...
>:58  00 00 00 00 00 00 11 00 ........
>:60  00 00 82 0f 00 4c 49 53 .....lis
>:68  54 44 52 55 43 4b a0 a0 tdruck
>:70  a0 a0 a0 a0 a0 00 00 00      ...
>:78  00 00 00 00 00 00 01 00 ........
>:80  00 00 82 16 01 48 41 52 .....har
>:88  44 43 4f 50 59 a0 a0 a0 dcopy
```

Am linken Rand stehen die „Bytenummern“ des ersten in der Zeile befindlichen Bytes. Dann folgen 8 Bytes, deren Inhalt als Hex-Zahlen ausgegeben

sind. Dahinter 8 ASCII-Zeichen, die den Hex-Zahlen in der Zeile entsprechen. Der ASCII-Ausdruck ist sehr sinnvoll, weil der Inhalt des Blocks schneller zu erkennen ist.
Deutlich sind die Namen der vorhandenen Programme (Files) sichtbar. Der Programmname 'buchdruck' steht ab Byte $05. Da der Name 16 Zeichen betragen darf, ist der Name mit SHIFT/SPACE - Zeichen aufgefüllt ($a0 = 196 = SHIFT/SPACE). Bei allen Filenamen ist dieses gleich.
Die beiden Bytes vor dem Filenamen besagen, in welcher Spur und welchem Sektor das Programm (File) beginnt. 'buchdruck' beginnt in Spur $14 Sektor $04, 'buchwert' in Spur $11 Sektor $00. Vor diesen beiden Bytes steht der Filetyp. In diesem Beispiel steht vor allen Filenamen eine $82. $82 ist die „Kennzahl" für Programmfiles.

DELeted	$80
SEQential	$81
PRoGram	$82
USeR	$83
RELative	$84

Hinter dem Filenamen stehen 9 Bytes mit dem Inhalt $00 und werden nicht benutzt. Die folgenden 2 Bytes geben die Länge des Files in Blöcken an. Es folgen 2 „Trenn-Bytes", sie kennzeichnen das Ende des Eintrages.

Wichtig sind die Bytes $00 und $01 des Blockes. Da die Directory länger sein kann als nur 1 Block, muß sie irgendwo weitergehen. Die Bytes $00 und $01 geben die Spur und den Sektor des folgenden Directory-Eintrags wieder. In diesem Beispiel liegt der nächste Directory-Eintrag in Spur $12 und Sektor $04. Endet die Directory, d.h. es folgt kein weiterer Directory-Block, steht in Byte $00 der Wert $00. In Byte $01 steht dann die Anzahl der belegten Bytes von diesem Block. Wird die Directory eingelesen und das DOS trifft auf eine $00 in Byte $00, liest er nur noch die in Byte $01 angegebene Anzahl Bytes ein und beendet seine Arbeit.

D15.1 Die BAM

In der BAM (Block-Availability-Map) einer Disk (Spur 18, Sektor 0) kann es folgendermaßen aussehen:

```
>:00  12 01 41 00 15 ff ff 1f ..a..^^.
>:08  15 ff ff 1f 15 ff ff 1f .^^..^^.
>:10  15 ff ff 1f 15 ff ff 1f .^^..^^.
>:18  15 ff ff 1f 15 ff ff 1f .^^..^^.
>:20  15 ff ff 1f 15 ff ff 1f .^^..^^.
>:28  15 ff ff 1f 15 ff ff 1f .^^..^^.
>:30  15 ff ff 1f 15 ff ff 1f .^^..^^.
 :38  15 ff ff 1f 0a 0a af 14 .^^.../.
>:40  00 00 00 00 00 00 00 00 ........
>:48  0f 6c ff 07 00 00 00 00 .L^.....
>:50  0a 39 fc 00 13 ff ff 07 .9ß..^^.
>:58  13 ff ff 07 13 ff ff 07 .^^..^^.
>:60  13 ff ff 07 12 ff ff 03 .^^..^^.
>:68  12 ff ff 03 12 ff ff 03 .^^..^^.
>:70  12 ff ff 03 12 ff ff 03 .^^..^^.
>:78  12 ff ff 03 11 ff ff 01 .^^..^^.
>:80  11 ff ff 01 11 ff ff 01 .^^..^^.
>:88  11 ff ff 01 11 ff ff 01 .^^..^^.
>:90  44 49 53 4b 45 54 54 45 diskette
>:98  a0 a0 a0 a0 a0 a0 a0 a0
>:a0  a0 a0 44 4b a0 32 41 a0   dk 2a
>:a8  a0 a0 a0 00 00 00 00 00    .....
>:b0  00 00 00 00 00 00 00 00 ........
>:b8  00 00 00 00 00 00 00 00 ........
>:c0  00 00 00 00 00 00 00 00 ........
>:c8  00 00 00 00 00 00 00 00 ........
>:d0  00 00 00 00 00 00 00 00 ........
>:d8  00 00 00 00 00 00 00 00 ........
>:e0  00 00 00 00 00 00 00 00 ........
>:e8  00 00 00 00 00 00 00 00 ........
>:f0  00 00 00 00 00 00 00 00 ........
>:f8  00 00 00 00 00 00 00 00 ........
```

In den Bytes $00 und $01 steht Spur/Sektor des ersten Blockes der Directory ($12/$01). In Byte $02 folgt das Aufzeichnungsformat, bei der VC-1541 immer $41. Byte $03 ist ein Nullbyte (immer „0") und wird nicht benutzt.

Der Diskettenname steht in $90 bis $9f, aufgefüllt mit SHIFT/SPACE ($a0). Das ID (Identifizierungsmerkmal) befindet sich in den Bytes $a2 und $a3. Das

Aufzeichnungsformat (2A) folgt in Byte $a5 und $a6. Die Bytes $a7 bis $aa sind mit SHIFT/SPACE gefüllt. Der Rest ($ab bis $ff) wird nicht benutzt und ist mit $00 gefüllt.

Die eigentliche BAM (welche Blöcke sind verfügbar, frei) liegt in den Bytes $04 bis $8f. Jeweils 4 Bytes bilden den Eintrag für eine Spur. Die Bytes $04 bis $07 für Spur 1, Byte $08 bis $0b für Spur 2, die Bytes $8c bis 8f für Spur 35. Alle Einträge sind identisch aufgebaut, so daß sich die Beschreibung auf einen Eintrag beschränken kann. Im ersten der 4 Bytes steht die Anzahl der freien, nicht belegten Blöcke dieser Spur.

In Byte $04 der BAM steht der Wert $15, es sind also 21 Blöcke in der Spur 1 frei verfügbar (nicht belegt). Die Bits der folgenden 3 Bytes kennzeichnen jeden einzelnen Sektor dieser Spur als belegt oder als frei.

```
Byte $05 : Sektor  0- 7   der Spur 1
Byte $06 : Sektor  8-15   der Spur 1
Byte $07 : Sektor 16-23   der Spur 1
```

Ein gesetztes Bit eines Bytes kennzeichnet einen freien, nicht belegten Sektor. Ist das Bit gelöscht, ist der Sektor bereits belegt.

```
Byte $05 : $ff  = 255 = 1111 1111   :Sektor  0- 7 frei
Byte $06 : $ff  = 255 = 1111 1111   :Sektor  8-15 frei
Byte $07 : $1f  =  31 = 0001 1111   :Sektor 16-20 frei
```

Befinden sich weniger als 21 Sektoren auf einer Spur, werden die nicht vorhandenen Blöcke mit einer „0" gekennzeichnet.

D15.2 Ändern des Diskettennamens

Um etwas Übung mit dem Disketten-Monitor zu bekommen, soll der Disketten-Name geändert werden.

```
Laden der BAM     R1200
Listen            L90A0
```

Die Bytes $90 bis $a0 werden auf dem Bildschirm gelistet. Die ASCII-Werte des neuen Diskettennamens müssen in hexadezimale Zahlen umgewandelt werden. Die Diskette soll z.B. den Namen „Test“ erhalten (die ASCII-Werte der Buchstaben können Sie aus G2.2 entnehmen).

```
T  =  84 = $54
E  =  69 = $45
S  =  83 = $53
T  =  84 = $54
```

Die restlichen 12 Bytes des Namens müssen mit SHIFT/SPACE ($a0) aufgefüllt werden. Mit

```
A90A0
```

können Sie die Bytes entsprechend ändern und mit

```
W1200
```

den geänderten Block auf die Diskette zurückschreiben. Nach dem Laden der Directory besitzt die Diskette den Namen TEST.

D16.0 Relative Datei

Relative Datei - das Zauberwort der professionellen Datentechnik - bedeutet: schneller Zugriff auf große Datenmengen. Die Datenmenge ist nur abhängig von der Kapazität der Diskette.
Die Vorteile einer relativen Datei gegenüber einer sequentiellen Datei sind derart groß, daß fast immer die relative Datei vorgezogen wird. Ein Vorteil ist, daß nicht alle Daten in den Computer eingelesen werden müssen. So ist es z. B. mit einem VC-20 bei einer Speicherkapazität von 3.5 kByte möglich, Daten in der Größenordnung von 160 kByte zu verwalten.

In der Praxis bedeutet das, daß ohne weiteres eine Adressdatei mit 2000 Teilnehmern erstellt werden kann.

D16.1 Aufbau einer relativen Datei

Bei einer sequentiellen Datei dürfen Datensätze eine unterschiedliche Länge, bedingt durch das „hintereinander Ablegen", aufweisen. Die relative Datei hingegen sucht den Datensatz „relativ" zum Anfang der Datei. Deshalb müssen alle Datensätze eine exakt gleiche Datensatzlänge haben. Ein Beispiel soll das Prinzip der relativen Datei verdeutlichen.

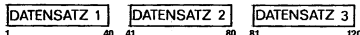


Abb. 26 3 Datensätze

Es sind 3 Datensätze mit je 40 Zeichen vorhanden, d.h. der Datensatz 1 liegt ab der Stelle 1, Datensatz 2 ab der Stelle 41 und der Datensatz 3 ab der Stelle 81. So hat jeder Datensatz eine eindeutig definierte Position. Der Anfang eines Datensatzes ist leicht zu bestimmen:

(Datensatznummer - 1) * LÄNGE DES DATENSATZ + 1

(3 - 1) * 40 + 1 = 81
(2 - 1) * 40 + 1 = 41
(1 - 1) * 40 + 1 = 1

Deutlich wird, und darauf möchte ich besonders hinweisen, daß die Datensatzlänge bei allen Datensätzen der Datei gleich sein muß !!!!! Ist der Anfang des Datensatzes bestimmt, werden die folgenden 40 Zeichen eingelesen und einer Variablen zugewiesen.
Die Datensätze werden zwar auch wieder hintereinander abgelegt, die Zugriffsreihenfolge beim Schreiben oder Lesen kann jedoch frei gewählt werden. Es ist nur möglich, Daten in einen Datensatz zu legen, wenn dieser vorhanden ist! Aus diesem Grund müssen zuerst alle Datensätze (z.B. 200 Stück) initialisiert werden. Diese Datensätze werden z.B. mit „SPACE" gefüllt. So sind dann die Datensätze 1 bis 200 vorhanden und können bearbeitet werden. Wird der Datensatz 9 eingelesen, wird ein „Zeiger" auf die Stelle ((9-1)*40+1) positioniert und die folgenden 40 Zeichen in eine Variable eingelesen. Das Schreiben läuft ebenso: „Zeiger" positionieren und dann 40 Zeichen in die Datei schreiben.

- Die Datensätze müssen alle gleich lang sein,
- alle Datensätze, die in der Datei genutzt werden sollen, müssen vorhanden sein.
- es muß darauf geachtet werden, daß nicht in Datensätze geschrieben wird, die noch nicht vorhanden sind.

Ein Record (Datensatz) darf beim C=64 eine Länge von maximal 254 Bytes haben.

D16.2 Vorbereiten einer relativen Datei

Auch relative Dateien müssen mit einem OPEN-Befehl eröffnet werden.

OPEN 1,8,2,"name,L,"+CHR$(LÄNGE)

Neben den bekannten Parametern (Kanalnummer, Gerätenummer, Sekundäradresse, Filename), folgt dem Filenamen ein „L". Dieses „L" teilt der Floppy (dem DOS in der Floppy) mit, daß eine relative Datei eröffnet werden soll. Die Recordlänge (Datensatzlänge) wird in einem CHR$(...)-Befehle dem DOS übermittelt. Bevor eine relative Datei eröffnet wird, müssen Sie sich Gedanken darüber machen,

a) wieviele Records (Datensätze) werden benötigt,
b) wie lang wird maximal ein Record.

Die Recordlänge einer relativen Datei kann später nicht mehr geändert werden! Records können mit einem INPUT# - Befehl gelesen werden, wenn der Record weniger als 88 Zeichen besitzt (siehe D10.0). Beachtet werden sollte auch, daß beim Schreiben mit PRINT# hinter dem Datensatz noch ein RETURN (CHR$(13)) gesetzt wird. Wenn ein Record 50 Zeichen aufnehmen soll, muß die Recordlänge 51 Zeichen betragen (1 Byte für das RETURN). Die Records erhalten eine Nummer - Recordnummer - (siehe D16.1), nach der dann „relativ" gesucht wird.
Als Beispiel soll eine Datei 50 Records mit einer Recordlänge von 51 Byte (1 Byte für RETURN) erhalten:

```
10 open 1,8,2,"relativ,l,"+chr$(51)
20 open 2,8,15
```

Mit OPEN 2,8,15 wird ein „Zeiger-Kanal" geöffnet, der später auf die Records zeigt. Dieser Zeiger muß positioniert werden:

PRINT # 2, "P" + CHR$(kanal) + CHR$(low) + CHR$(high) + CHR$(prüf)

Das "P" ist der eigentliche Positionier-Befehl. Das „low" und „high" ist die Recordnummer, umgerechnet in 2 Byte.

high = INT(record/256)
low = record-high*256

Der „Kanal" bezieht sich auf das OPEN - Statement (Zeile 20) und ist in diesem Fall „2".
„Prüf" gibt die Stelle innerhalb des „angewählten" Records an (Zeichen), auf welches positioniert werden soll. Hier das erste Zeichen, also „1".

```
30 rn = 50
40 hb = int(rn/256)
50 lb = rn - hb * 256
60 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
```

Um alle Records „freizugeben" (Erstellen, Initialisieren), wird der letzte Record (50.) mit einem CHR$(255) beschrieben und damit „freigegeben". Automatisch werden dann alle kleineren Record-Nummern ebenfalls freigegeben und mit CHR$(55) beschrieben.

```
70 print#1, chr$(255)
80 close1: close2
```

Das vollständige Programm erstellt eine relative Datei und gibt alle Records (die unterhalb der eingegeben Record-Nummer liegen) frei. Das Eröffnen der Datei kann je nach Recordanzahl mehrere Minuten dauern!

Es sind jetzt 50 Records mit den Nummern 1 bis 50 vorhanden, die jeweils eine Länge von 51 Zeichen aufweisen und als 1. Zeichen ein CHR$(255) tragen. Mit dem Disketten-Monitor können Sie sich dieses genauer ansehen.

D16.3 Eintrag in der Directory

```
>:78  00 00 00 00 00 00 01 00 ........
>:80  00 00 84 0f 03 52 45 4c .....rel
>:88  41 54 49 56 a0 a0 a0 a0 ativ
>:90  a0 a0 a0 a0 a0 0f 0d 33       ..3
>:98  00 00 00 00 00 00 0c 00 ........
>:a0  00 00 00 00 00 00 00 00 ........
>:a8  00 00 00 00 00 00 00 00 ........
```

Hier ist ein Auszug aus der Directory zu sehen. Er gibt den Eintrag der eröffneten, relativen Datei wieder. Neben dem Eintrag des Namens ist deutlich der Filetyp mit $84 (siehe D15.0) zu sehen. Der erste Block der Datei beginnt in der Spur $0f Sektor $03. Die Länge der Datei ist ebenfalls wie bei einem normalen Eintrag gekennzeichnet ($0c = 12 Blöcke). Es sind aber, im Gegensatz zu allen anderen Einträgen, 3 weitere Bytes benutzt.

Zuerst das Byte $97. Hier steht die Recordlänge ($33 = 51), davor die Spur ($0f) und Sektor ($0d) des sogenannten „Side-Sektor-Blocks" (siehe D16.4). Doch sehen Sie sich zunächst den ersten Block der relativen Datei an (R0F03).

```
>:00  0f 0f ff 00 00 00 00 00 ..^.....
>:08  00 00 00 00 00 00 00 00 ........
>:10  00 00 00 00 00 00 00 00 ........
>:18  00 00 00 00 00 00 00 00 ........
>:20  00 00 00 00 00 00 00 00 ........
>:28  00 00 00 00 00 00 00 00 ........
>:30  00 00 00 00 00 ff 00 00 .....^..
>:38  00 00 00 00 00 00 00 00 ........
>:40  00 00 00 00 00 00 00 00 ........
>:48  00 00 00 00 00 00 00 00 ........
>:50  00 00 00 00 00 00 00 00 ........
>:58  00 00 00 00 00 00 00 00 ........
>:60  00 00 00 00 00 00 00 00 ........
>:68  ff 00 00 00 00 00 00 00 ^.......
>:70  00 00 00 00 00 00 00 00 ........
>:78  00 00 00 00 00 00 00 00 ........
>:80  00 00 00 00 00 00 00 00 ........
>:88  00 00 00 00 00 00 00 00 ........
>:90  00 00 00 00 00 00 00 00 ........
>:98  00 00 00 ff 00 00 00 00 ...^....
```

Wieder ist in den ersten beiden Bytes ($00 und $01) die Spur und Sektor des nächsten Blockes zu erkennen. Danach folgt ein $FF (255) in Byte $02 und darauf 50 mal die $00. Hier ist ein Record mit der Länge von 51 Bytes zu sehen. Im ersten Byte der Records steht die 255, die mit CHR$(255) übermittelt worden ist (siehe D16.2). Der nächste Block beginnt wieder mit $FF usw.. Verfolgt man die relative Datei weiter, findet man ständig diesen Satzaufbau wieder.

D16.4 Die Side-Sektor-Blöcke

In den Bytes $95 und $96 der Directory (siehe D16.3) sind die Spur und der Sektor des ersten Side-Sektor-Blocks zu entnehmen. Im gezeigten Beispiel war es die Spur $0f und der Sektor $0d. Aus diesem Side-Sektor-Block holt sich die Floppy die Information, auf welcher Spur/Sektor die entsprechenden Recordnummern liegen.

```
>:00  00 25 00 33 0f 0d 00 00 .%.3....
>:08  00 00 00 00 00 00 00 00 ........
>:10  0f 03 0f 0f 0f 08 0f 12 ........
>:18  0f 09 0f 14 0f 0a 0f 0b ........
>:20  0e 00 0e 0a 0e 14 00 00 ........
>:28  00 00 00 00 00 00 00 00 ........
>:30  00 00 00 00 00 00 00 00 ........
>:38  00 00 00 00 00 00 00 00 ........
```

In Byte $00 und $01 steht die Adresse des nächsten Blockes des Side-Sektors. Byte $00 ist $00, wie üblich bedeutet dieses, daß keine weiteren Blöcke folgen und das aus diesem Block 37 Bytes ($25) benutzt sind. In Byte $02 liegt die laufende Nummer des Side-Sektor-Blocks. Dieses ist Block-Nummer „0“. Eine Datei kann maximal 6 solcher Side-Sektor-Blöcke enthalten (0 bis 5). In Byte $03 steht die Recordlänge ($33 = 51) und die nächsten 12 Bytes enthalten den Ort (Spur/Sektor) für die 6 möglichen Side-Sektor-Blöcke. So sind die Bytes $04/$05 für Block 0, die Bytes $06/$07 für Block 1 usw. verantwortlich. Ist ein Side-Sektor-Block nicht benutzt, steht für Spur und Sektor jeweils eine „Null“. Da nur der Side-Sektor-Block 0 genutzt ist, steht auch nur in den Bytes $04/$05 eine Information, alle anderen sind auf „Null“ gelegt.

Ab Byte $10 liegen die eigentlichen Zeiger für die Datenblöcke. Auch diese Zeiger bestehen wieder aus der Spur- und der nachfolgenden Sektorposition. Es sind in diesem Fall 11 Bytepaare, beginnend mit der Spur $0f und Sektor $03 für den 1. Datenblock. Wie nun ein Record gefunden wird, soll das folgende Beispiel verdeutlichen.

Aus der Datei soll der 25. Datensatz (Record) gefunden und eingelesen werden. In einem Block können 254 Bytes (die beiden Bytes $00 und $01, die den nächsten Block zeigen, sind abgezogen) gespeichert werden.

Blocknummer=(Recordnummer - 1) * Recordlänge / 254
Blocknummer=(25 - 1) * 51 / 254
Blocknummer= 4 REST 208

Das DOS hat nun die Information, daß sich der Datensatz im 4. Datenblock befindet. Aus dem REST + 2 (210 = $D2) wird die Position innerhalb des Datenblocks ermittelt. Der 4. Datenblock steht in Spur $0f und Sektor $09 (die Datenblöcke stehen in den Bytes $10 bis $25 des Side-Sektor-Blocks.

Mit diesem Verfahren ermittelt sich das DOS den genauen Ort eines Datensatzes (Record) innerhalb einer relativen Datei.

Anzumerken ist noch, daß in einem Side-Sektor-Block nur auf 120 Datenblöcke gezeigt werden kann. Wird bei der Rechnung ein Wert größer als 120 ermittelt, liegt der Datenblock in einem anderen Side-Sektor-Block. Mit einer Division durch 120 läßt sich der richtige Side-Sektor-Block ermitteln. Somit kann jeder Datensatz mit dem Lesen von maximal 2 Blöcken gefunden werden.

Side-Sektor-Block = Blocknummer / 120
Side-Sektor-Block = 400 / 120
Side-Sektor-Block = 3 (.3333)

Die relative Datei könnte theoretisch 6 * 120 * 254 = 182880 Bytes umfassen (6 Side-Sektor-Blocks a 120 Blocknummern mal der maximalen Speicherkapazität eines Records). Dieser Wert ist größer als die Speicherkapazität einer Diskette. Die maximale Kapazität der Diskette liegt bei 664 Blöcke * 254 Byte = 168656 Bytes.
Rechnet man noch die 6 möglichen Side-Sektor-Blöcke ab, so ist es möglich

(664 - 6) * 254 = 167132

Bytes in einer relativen Datei zu verwalten.
Nach soviel Therorie geht es jetzt langsam in die Praxis. Ein Record soll beschrieben werden.

D17.0 Beschreiben eines Records

Nachdem eine relative Datei erzeugt worden ist (siehe D16.2, D16.3), können Daten in sie geschrieben werden. Das folgende kleine Programm ermöglicht das Beschreiben eines Records:

```
10 print chr$(147);"beschreiben eines record"
20 open 1,8,2,"relativ,l,"+chr$(51)
30 open 2,8,15
40 input "welcher record (1-50) :";rn
50 if rn<1 or rn>50 then 40
60 input "inhalt (50 zeichen) :";rn$
70 if len(rn$)>50 then 60
80 print#2, "p"+chr$(2)+chr$(rn)+chr$(0)+chr$(1)
90 print#1, rn$
100 print: print "(w)eiter oder (e)nde ?"
110 get a$: if a$ = "" then 110
120 if a$ = "e" then close 1: close 2: end
130 if a$ = "w" then 40
140 goto 110
```

Nach dem Öffnen beider Kanäle (Zeile 20-30) wird nach der Recordnummer gefragt (Datensatznummer) und geprüft, ob sie im zulässigen Bereich (1-50) liegt. Nach der Eingabe des Inhaltes folgt die Prüfung, ob der String RN$ nicht mehr als 50 Zeichen beinhaltet. Der „Zeiger" (Zeile 80) wird auf den Record positioniert. Da eine zulässige Recordnummer nicht größer als 50 ist, kann die Umrechnung in High- und Lowbyte entfallen (diese Umrechnung ist nur bei relativen Dateien mit mehr als 255 Records notwendig). In den positionierten Record wird (Zeile 90) die Variable RNS geschrieben. Abschließend folgt die Abfrage, ob weiter eingegeben werden soll oder nicht.

D17.1 Lesen eines Records

Auch der Lesevorgang soll anhand eines Programmes erklärt werden.

```
10 print chr$(147);"lesen eines record"
20 open 1,8,2,"relativ,1,"+chr$(51)
30 open 2,8,15
40 input "welcher record (1-50) :";rn
50 if rn<1 or rn>50 then 40
60 print#2, "p"+chr$(2)+chr$(rn)+chr$(0)+chr$(1)
70 input#1, rn$
80 if rn$ = chr$(255) then print "leer": goto 100
90 print rn$
100 print: print"(w)eiter oder (e)nde ?"
110 get a$: if a$ = "" then 110
120 if a$ = "e" then close 1: close 2: end
130 if a$ = "w" then 40
140 goto 110
```

Der Programmaufbau ist fast identisch wie beim „Beschreiben“. Der Record wird ausgewählt und der Zeiger positioniert. Mit der INPUT#-Anweisung wird der Record eingelesen. Ist der Inhalt des Records gleich CHR$(255), so ist der Record unbeschrieben.

D17.2 Wie sieht ein beschriebener Record aus ?

```
>:50  00 00 00 00 00 00 00 00 ........
>:58  00 00 00 00 00 00 00 00 ........
>:60  00 00 00 00 00 00 00 00 ........
>:68  54 45 53 54 45 49 4e 47 testeing
>:70  41 42 45 20 44 45 53 20 abe des
>:78  33 2e 20 52 45 43 4f 52 3. recor
>:80  44 0d 00 00 00 00 00 00 d.......
>:88  00 00 00 00 00 00 00 00 ........
>:90  00 00 00 00 00 00 00 00 ........
>:98  00 00 00 ff 00 00 00 00 ...¯....
>:a0  00 00 00 00 00 00 00 00 ........
```

```
>:a8  00 00 00 00 00 00 00 00 ........
>:b0  00 00 00 00 00 00 00 00 ........
>:b8  00 00 00 00 00 00 00 00 ........
>:c0  00 00 00 00 00 00 00 00 ........
>:c8  00 00 00 00 00 00 ff 00 ......^.
>:d0  00 00 00 00 00 00 00 00 ........
>:d8  00 00 00 00 00 00 00 00 ........
```

Der Auszug aus dem 1. Block der relativen Datei zeigt deutlich den Inhalt des 3. Records. Das letzte Zeichen des Records ist $0D (13); das RETURN, das durch die PRINT # -Anweisung mitgesetzt worden ist. Alle anderen Records blieben unverändert.

D18.0 Record zerlegen in Datenfelder

Die Datensätze, die in den Beispielen auf die Records geschrieben wurden, beinhalteten immer nur ein (!) Datenfeld. Besteht ein Datensatz aus mehreren Datenfeldern, muß innerhalb des Records ein Bereich für jedes Datenfeld reserviert werden.

D18.1 Wieviele Datenfelder ?

Bevor eine relative Datei erstellt und auf der Diskette angelegt wird, sollte man sich über den Aufbau der Dateistruktur Gedanken machen. Die Telefon-Datei aus dem Beispiel D9.0 besitzt 4 Datenfelder (Name, Ort, Straße, Telefonnummer). Diese 4 Datenfelder (ein Datensatz) müssen nun in einem Record untergebracht werden, wobei die Länge des Datensatzes nicht die Länge des Records überschreiten darf. Aus diesem Grund müssen auch die Datenfelder eine maximale Länge aufweisen, deren Summe nicht größer als die Recordlänge wird.

Datenfeld 1 : Name	20 Zeichen
Datenfeld 2 : Ort	15 Zeichen
Datenfeld 3 : Straße	20 Zeichen
Datenfeld 4 : Telefon	10 Zeichen
Datensatz	65 Zeichen
	+ 1 RETURN
Recordlänge	66 Zeichen

Ein Datensatz benötigt in diesem Beispiel einen Platz von insgesamt 66 Zeichen, wobei den einzelnen Feldern wiederum eine bestimmte Größe zugeordnet wird. Das bedeutet, daß sichergestellt sein muß, daß der reservierte Platz für ein Datenfeld immer ausreichend ist. Der Aufbau des Records muß dann bei allen Records gleich sein. Also: die ersten 20 Zeichen eines Records beinhalten den Namen, die folgenden 15 Zeichen den Ort etc. Ist ein Datenfeld kleiner als der für ihn reservierte Platz, muß das Datenfeld mit Leerzeichen (SPACE) gefüllt werden, um damit die Datenfeldlänge einzuhalten.

D18.2 Datenfelder werden zum Datensatz

An dieser Stelle möchte ich Ihnen zwei Methoden vorstellen, um aus Datenfeldern einen Datensatz zu bilden. Die Datenfelder sehen in diesem Beispiel folgendermaßen aus:

Name	=	30 Zeichen
Vorwahl	=	8 Zeichen
Telefon	=	12 Zeichen
		50 Zeichen

Dieser Datensatz (50 Zeichen) kann in einen Record mit 51 Zeichen geschrieben werden (1 Zeichen für RETURN). Ist das Datenfeld „Name" kleiner als 30 Zeichen, wird das Datenfeld „Name" mit SPACE (CHR$(32)) aufgefüllt.

Methode 1

```
10 rem************ methode 1 *********
20 input" name (30):";na$
30 if len(na$) >30 then 20
40 if len(na$) =30 then 80
50 for i = len(na$)+1 to 30
60 na$ = na$+chr$(32)
70 next i
80 input" vorw. (8):";vw$
90 if len(vw$) > 8 then 80
100 if len(vw$) = 8 then 140
110 for i = len(vw$)+1 to 8
120 vw$ = vw$+chr$(32)
130 next i
140 input" tel.  (12):";te$
150 if len(te$) > 12 then 140
160 if len(te$) = 12 then 200
170 for i = len(te$)+1 to 12
180 te$ = te$+chr$(32)
190 next i
200 rc$ = na$+vw$+te$
210 print len(rc$)
220 print rc$
230 rem*****************************
```

Nach der Eingabe des Datenfeldes wird die Länge überprüft. Ist das Datenfeld größer als vorher vereinbart, muß die Eingabe wiederholt werden. Ist die Datenfeldlänge gleich der maximalen Länge, fährt das Programm mit der nächsten Eingabe fort.
Das Auffüllen der Datenfelder mit SPACE übernimmt jeweils die FOR-NEXT Schleife. Ab dem folgendem Zeichen (LEN(NA$)+1) bis zur maximalen Länge des Datenfeldes werden nun SPACE-Zeichen angefügt. Jedes der Datenfelder wird so behandelt und in der Zeile 200 werden die Datenfelder zu einem Datensatz (einer Variablen) zusammengefaßt. Der so entstandene Datensatz hat immer die Länge von 50 Zeichen. Um den entstandenen Datensatz rc$ in seiner ganzen Länge zu betrachten, wird er noch einmal ausgedruckt.

Methode 2

Diese Methode ist eine wesentlich elegantere Lösung. Doch zunächst das kurze Programm:

```
10 rem********* methode 2 ***********
20 x$="                              "
30 input" name (30):";na$
40 if len(na$) > 30 then 30
50 na$=na$+left$(x$,30-len(na$))
60 input" vorw. (8):";vw$
70 if len(vw$) > 8 then 60
80 vw$=vw$+left$(x$,8-len(vw$))
90 input" tel. (12):";te$
100 if len(te$) > 12 then 90
110 te$=te$+left$(x$,12-len(te$))
120 rc$=na$+vw$+te$
130 print len(rc$)
140 print rc$
150 rem******************************
```

In Zeile 20 sind einer Stringvariablen (x$) 30 SPACE-Zeichen zugewiesen worden. Dieser String beinhaltet soviele SPACE-Zeichen, wie die maximale Größe eines Datenfeldes (in diesem Falle 30 Zeichen für „Name"). Nach der Eingabe wird wiederum überprüft, ob die Datenfeldlänge überschritten worden ist. Das Auffüllen übernehmen dann die Zeilen 50, 80 und 110. Dem eingegebenen Datenfeld wird ein Teil des Strings x$ angehängt (dieser Teilstring besteht ausschließlich aus SPACE). Die Länge des anzuhängenden Strings errechnet sich aus der

maximalen Datenfeldlänge - tatsächlichen Länge

Ein Beispiel: in der Variablen für den Namen (na$) wurde MANFRED WALTER THOMA eingegeben. Die Länge der Variablen na$ beträgt dann 20 Zeichen (LEN(NA$)). Die Anzahl der benötigten SPACE-Zeichen ist 10 (30-20 = 10!! oder 30-len(na$) !!). Mit diesem Wert ist auch die Länge des Teilstrings bestimmt, der dann an die Variable na$ angehängt wird.

NA$ = NA$ + LEFT$ (X$, 30-LEN(NA$))

Der Datensatz ergibt sich wiederum aus dem Zusammenfügen der Datenfelder (Zeile 120). Das Datenfeld (rc$) hat auf jeden Fall die Länge von 50 Zeichen. Der Datensatz kann nun ohne Probleme in einen Record geschrieben werden.

D18.3 Datensatz in Datenfelder zerlegen

Wird ein Datensatz (Record) aus einer relativen Datei gelesen, so muß der Datensatz wieder in seine Datenfelder zerlegt werden. Die Datenfelder liegen in einem Datensatz immer an einer eindeutig bestimmten Stelle. Nach dem Einlesen des Datensatzes wird er mit Hilfe der folgenden Befehle in seine Datenfelder zerlegt:

```
NA$  =  MID$(RC$,   1,20)   oder   =   LEFT$(RC$,30)
VW$  =  MID$(RC$,  31,  8)
TE$  =  MID$(RC$,   39,12)  oder   =   RIGHR$(RC$,12)
```

```
300  rem***  zerlegen  in  datenfelder  ***
310  na$=left$(rc$,30)
320  vw$=mid$(rc$,31,8)
330  te$=right$(rc$,12)
340  print " * name :";na$
350  print " * vorw.:";vw$
360  print " * tel. :";te$
370  rem*****************************
```

Mit diesen Methoden können Datensätze in ihre Felder zerlegt und zusammengefügt werden.
Bisher besteht nur die Möglichkeit,über die Datensatznummern auf einen Record zuzugreifen. Das bedeutet, daß dem Benutzer immer bekannt sein muß, wo (in welchem Record) die Daten zu finden sind. Das ist natürlich bei ei großen Datei nicht möglich. Wie läßt sich dieses Problem lösen ?

D19.0 Index-Dateien – Hilfe beim Finden von Datensätzen

Um in relativen Dateien bestimmte Datensätze zu finden, ist es notwending, ihre Position innerhalb der Datei zu kennen. Diese Arbeit, zu wissen wo was

liegt, übernimmt eine sogenannte „Index-Datei" (Verzeichnis, Register). Innerhalb der Index-Datei befindet sich eine Liste, bestehend aus Suchbegriff und dem Ort, wo der Datensatz innerhalb der relativen Datei liegt. Das bedeutet, aus einer großen Datenmenge wird eine kleine Datenliste (Index-Datei) selektiert.
Soll innerhalb einer relativen Datei nach dem „Namen" gesucht werden, so befinden sich in der Index-Datei einmal der Name und (dieses ist das Wesentliche) die Recordnummer, wo der gesuchte Datensatz innerhalb der relativen Datei liegt.

	Name	Record
(1)	Meyer	21
(2)	Huber	2
(9)	Kalli	120

Um die Daten für „Meyer" zu finden, wird nur die Index-Datei durchsucht, bis „Meyer" gefunden wird. Aus der Index-Datei wird dann die Recordnummer entnommen und damit kann der gesamte Datensatz aus der relativen Datei gelesen werden.
Das bedeutet, daß das eigentliche Suchen nach einem Suchbegriff nur in der Index-Datei durchgeführt wird. Für jedes Suchkriterium ist daher eine Index-Datei notwendig. Eine Index-Datei befindet sich in der Regel im Computer, um auf die Daten möglichst schnell zugreifen zu können. Der Aufbau einer solchen Index-Datei kann sehr unterschiedlich sein.

D19.1 Index-Datei mit zwei Feldern

In einer Variablen befindet sich das Suchkriterium, z.B. der „Name" und eine zugehörige Variable enthält dann die Recordnummer, wo sich der Datensatz befindet.

```
10 dim in$(99),nr(99)
20 input "namen :"; n$
30 for i = 1 to 99
40 if n$ = in$(i) then 80
50 next i
```

```
60 print "nicht vorhanden"
70 goto 20
80 print "recordnummer :";nr(i)
90 ...
```

Im Varibelenfeld in$(..) befindet sich der Name und in der zugehörigen Variablen nr(..) die Recordnummer.

D19.2 Index-Datei mit einem Feld

In einer Variablen wird sowohl das Suchkriterium als auch die Recordnummer untergebracht. Es werden z.B. die ersten vier Zeichen für die Recordnummer reserviert.

```
0001Meyer
0002Huber
```

```
10 input"name :";n$
20 n = len(n$)
30 for i = 1 to 99
40 if n$ = mid$(in$(i),5,n) then 80
50 next i
60 print"nicht gefunden"
70 goto 10
80 rn = val(left$(in$(i),4))
90 print"recordnummer :";rn
```

D19.3 Index-Nummer gleich Recordnummer

Eine weitere Möglichkeit besteht in der Zuordnung der Index-Nummer zur Recordnummer. Für jeden Datensatz (Record) ist ein Index-Feld notwendig. Für 100 Records benötigt man ein Array mit 100 Feldern - dim in$(100) -. Der Index wird der Recordnummer zugeordnet. in$(49) = Record 49, in$(12) = Record 12 usw.

```
10 input"name :";n$
20 for i = 1 to 99
30 if n$ = in$(i) then 70
40 next i
50 print"name nicht vorhanden"
60 goto 10
70 print"recordnummer :"; i
```

Es bietet sich an, Index-Dateien als sequentielle Dateien zu verwalten. Man spricht auch von einem sequentiell-indizierten Dateienprinzip. Alle Änderungen, die in der relativen Datei durchgeführt werden, müssen selbstverständlich auch in der Index-Datei durchgeführt werden. Wird ein Datensatz gelöscht, so ist auch in der Index-Datei zu kennzeichnen, daß dieser Datensatz wieder frei (nicht belegt) ist. Das Index-Feld kann z.B. mit chr$(255) (PI) als nicht belegt gekennzeichnet werden. Bei Eingabe von neuen Daten kann dann die Index-Datei durchsucht werden, bis ein „leerer" Record gefunden ist. Der neue Eintrag wird dann in diesen Record geschrieben.

D20.0 Adress-Datei

Auf den folgenden Seiten finden Sie ein Programm zur Bearbeitung einer Adress-Datei. Darin werden alle bisher behandelten Punkte zusammengefaßt. Die Index-Datei besteht aus den Einträgen:

- (1) Anzahl der Records (DS)
- (2) letzter belegter Record (AN)
- (3) Anzahl der gelöschten Records bis zum letzten belegten Record (LO).
- (n) es folgt dann die eigentliche Index-Datei mit den Suchbegriffen (Name).

Wurde ein Datensatz gelöscht, der in der Mitte der belegten Datensätze liegt, so wird die Varibale „LO" um eins erhöht. Das bedeutet, daß sich ein freier Platz zwischen den Datensätzen befindet. Ist „LO" größer als Null, wird die Index-Datei nach einem freien Platz durchsucht! Ist „LO" gleich Null (es ist kein gelöschter Platz vorhanden), wird der neue Eintrag hinter dem letzten Record gespeichert (AN). Der Vorteil liegt darin, daß nur nach einem gelöschten Platz gesucht wird, wenn auch ein gelöschter Platz vorhanden ist.

Ich hoffe, daß Ihnen dieses Programm einige Anstöße gibt, damit Sie sich eine Datei erstellen können, die Ihren ganz speziellen Vorstellungen entspricht.

```
10000 rem**********************************************
10010 rem*       adress-datei . suchen nach namen       *
10020 rem**********************************************
10030 :
10040 rem*************** farben setzen ******************
10050 poke 53280,0
10060 rem**********************************************
10070 :
10080 rem******** fehlerkanal oeffnen #3 up 50000 ********
10090 open 3,8,15
10100 rem**********************************************
10110 :
10120 rem*********** neue datei eroeffnen ? *************
10130 print chr$(147)
10140 print tab(13);"adress-datei"
10150 print: print
10160 print " * neue datei anlegen (j/n)"
10170 get a$: if a$ = "" then 10170
10180 if a$ = "j" then gosub 12880: goto 10130
10190 if a$ = "n" then gosub 13300: goto 10230
10200 goto 10170
10210 rem**********************************************
10220 :
10230 rem************* menue -hauptprogramm- ***********
10240 print chr$(147)
10250 print " ------------- m e n u e ------------"
10260 print: print
10270 print " *  (1)  suchen nach datensatznummer .."
10280 print " *  (2)  suchen nach namen ..........."
10290 print " *  (3)  eingabe neuer daten ........."
10300 print " *  (4)  loeschen von daten..........."
10310 print " *  (5)  aendern von daten ..........."
10320 print " *  (6)  programm-ende ..............."
10330 print
10340 print " *        bitte waehlen sie ..........."
10350 print: print: print
```

```
10360 print " ++++++ file-info ++++++"
10370 print " + name     : ";dn$;tab(24);"+"
10380 print " + datens.  :";ds;tab(24);"+"
10390 print " + laenge   : 88          +"
10400 print " + belegt   :";an-lo;tab(24);"+"
10410 print " +++++++++++++++++++++++"
10420 get a$: if a$ = "" then 10420
10430 a=val(a$)
10440 if a<1 or a>6 then 10420
10450 on a gosub 10490,10710,11130,11500,11740,12120
10460 goto 10240
10470 rem************************************************
10480 :
10490 rem********** suchen nach datensatznummer **********
10500 print chr$(147)
10510 print " --- suchen nach datensatznummer ---"
10520 print: print:rc=0
10530 print " * datensatznr. (";ds;") :";: input rc
10540 if rc<1 or rc>an or in$(rc)=chr$(255) then 10590
10550 gosub 12410
10560 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
10570 input#1, rc$
10580 print: print : goto  10620
10590 print: print
10600 print " * datensatz ist nicht belegt *"
10610 print:goto 10640
10620 gosub 12510
10630 gosub 12750
10640 print " * (w)eiter suchen oder (m)enue"
10650 get a$: if a$ = "" then 10650
10660 if a$ = "m" then return
10670 if a$ = "w" then 10490
10680 goto 10650
10690 rem************************************************
10700 :
10710 rem*************** suchen nach namen ***************
10720 print chr$(147)
10730 print " --- suchen nach namen ---"
10740 print: print:n1$="":n2$=""
```

```
10750 input " * namen   :";n1$: if len(n1$)=0 then return
10760 input " * vorname :";n2$
10770 if len(n2$)=0 then 10850
10780 n3$=n1$+left$(n2$,2)
10790 for i = 1 to an
10800 if n3$ = in$(i) then x=1:goto 10930
10810 next i
10820 print: print
10830 print " * nicht gefunden !! *"
10840 print
10850 print: print " * suche nur nach ";chr$(18);n1$:
      l=len(n1$)
10860 for i = 1 to an
10870 if n1$ = left$(in$(i),l) then 10930
10880 next i
10890 print: print
10900 print " * nicht gefunden !! *"
10910 print: print
10920 goto 11060
10930 rem*************** namen gefunden *****************
10940 rc=i: gosub 12410
10950 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
10960 input#1, rc$
10970 gosub 12510: print chr$(147):print:print
10980 gosub 12750
10990 print " * datens.  :";i: if x=1 then x=0: goto 11050
11000 print: print " * weitere ";n1$;" suchen (j/n)"
11010 get a$: if a$ = "" then 11010
11020 if a$ = "j" then 10880
11030 if a$ = "n" then return
11040 goto 11010
11050 rem************ richtig gefunden *****************
11060 print: print " * (w)eiter suchen oder (m)enue"
11070 get a$ : if a$ = "" then 11070
11080 if a$ = "m" then return
11090 if a$ = "w" then 10710
11100 goto 11070
11110 rem*********************************************
11120 :
```

```
11130 rem************** eingabe neuer daten **************
11140 print chr$(147)
11150 print " --- eingabe neuer daten ---"
11160 print: print:nn$="" :if an=ds and lo=0 then return
11170 input " * name     :";nn$: if len(nn$)=0 then return
11180 if len(nn$)>15 then print chr$(145);:goto 11170
11190 input " * vorname :";vn$
11200 if len(vn$)>15 then print chr$(145);:goto 11190
12100 input " * plz      :";pl$
11220 if len(pl$)> 4 then print chr$(145);:goto 11210
11230 input " * ort      :";ot$
11240 if len(ot$)>15 then print chr$(145);:goto 11230
11250 input " * strasse :";sr$
11260 if len(sr$)>15 then print chr$(145);:goto 11250
11270 input " * beruf    :";be$
11280 if len(be$)>10 then print chr$(145);:goto 11270
11290 input " * telefon :";te$
11300 if len(te$)>13 then print chr$(145);:goto 11290
11310 print: print
11320 print " * alle eingaben richtig (j/n)"
11330 get a$ : if a$ = "" then 11330
11340 if a$ = "n" then print chr$(19):goto 11150
11350 if a$ <>"j" then 11330
11360 if lo = 0 then 11410
11370 for i = 1 to an
11380 if in$(i)=chr$(255) then rc = i: lo=lo-1: goto 11400
11390 next i
11400 in$(rc)=nn$+left$(vn$,2):goto 11430
11410 an=an+1: rc=an
11420 in$(an)=nn$+left$(vn$,2)
11430 gosub 12620
11440 gosub 12410
11450 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
11460 print#1, rc$
11470 return
11480 rem************************************************
11490 :
11500 rem********** loeschen eines eintrages **************
11510 print chr$(147)
```

```
11520 print " --- loeschen von daten ---"
11530 print: print:rc=0
11540 input " * loeschen des datensatz nr. :";
11550 if rc<1 or rc>an then return
11560 if in$(rc)=chr$(255) then return
11570 gosub 12410
11580 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
11590 print#1, rc$
11600 gosub 12510
11610 gosub 12750
11620 print " * soll geloescht werden (j/n)"
11630 get a$: if a$ = "" then 11630
11640 if a$ = "n" then return
11650 if a$ <>"j" then 11630
11660 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
11670 print#1, chr$(255)
11680 rem********* indexdatei aufbereiten *****************
11690 if rc = an then an=an-1:goto 11710
11700 in$(rc)=chr$(255):lo=lo+1
11710 return
11720 rem**********************************************************
11730 :
11740 rem*************** aendern von eintraegen ***********
11750 print chr$(147)
11760 print " --- aendern von daten ---"
11770 print: print:rc=0
11780 input " * datensatznummer :";rc
11790 if rc<1 or rc>an or in$(rc)=chr$(255) then return
11800 gosub 12410
11810 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
11820 input#1, rc$
11830 gosub 12510
11840 gosub 12750
11850 print chr$(19):for i = 1 to 6: print: next
11860 print tab(12);:input nn$
11870 if len(nn$)>15 then print chr$(145);:goto 11860
11880 print tab(12);:input vn$
11890 if len(vn$)>15 then print chr$(145);:goto 11880
11900 print tab(12);:input pl$
```

```
11910 if len(pl$)> 4 then print chr$(145);:goto 11900
11920 print tab(12);:input ot$
11930 if len(ot$)>15 then print chr$(145);:goto 11920
11940 print tab(12);:input sr$
11950 if len(sr$)>15 then print chr$(145);:goto 11940
11960 print tab(12);:input be$
11970 if len(be$)>10 then print chr$(145);:goto 11960
11980 print tab(12);:input te$
11990 if len(te$)>13 then print chr$(145);:goto 11980
12000 print
12010 print " * alle aenderungen richtig (j/n)"
12020 get a$: if a$ = "" then 12020
12030 if a$ = "n" then 11850
12040 if a$ <>"j" then 12020
12050 in$(rc)=nn$+left$(vn$,2)
12060 gosub 12620
12070 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
12080 print#1, rc$
12090 return
12100 rem*************************************************
12110 :
12120 rem**************** programm - ende ****************
12130 print chr$(147)
12140 print " --- programm ende ---"
12150 print: print
12160 close 1: close 2
12170 print " * lege indexdatei ab"
12180 open 4,8,2,"@:"+dn$+".index,s,w"
12190 gosub 12350
12200 if fe <>0 then stop
12210 print#4, ds
12220 print#4, an
12230 print#4, lo
12240 if an = 0 then 12290
12250 for i = 1 to an
12260 print#4, in$(i)
12280 next i
12290 close 4: close 3
12300 print:print
```

```
12310 print " * auf wiedersehen *"
12320 end
12330 rem**********************************************
12340 :
12350 rem************** fehlerkanal lesen ***************
12360 input#3, fe,fe$
12370 print " * disk-status :";fe$
12380 return
12390 rem**********************************************
12400 :
12410 rem************** umwandlung hb und lb ************
12420 hb = int(rc/256): lb = rc-hb*256
12430 return
12440 rem**********************************************
12450 :
12460 rem***************** zeitschleife *****************
12470 for ze = 1 to 3000: next ze
12480 return
12490 rem**********************************************
12500 :
12510 rem************ zerlegen in datenfelder ***********
12520 nn$ = left$(rc$,15)
12530 vn$ = mid$(rc$,16,15)
12540 pl$ = mid$(rc$,31,4)
12550 ot$ = mid$(rc$,35,15)
12560 sr$ = mid$(rc$,50,15)
12570 be$ = mid$(rc$,65,10)
12580 te$ = right$(rc$,13)
12590 return
12600 rem**********************************************
12610 :
12620 rem**************** datensatz bilden **************
12630 sp$ = "               ": rem  15 space
12640 nn$ = nn$+left$(sp$,15-len(nn$))
12650 vn$ = vn$+left$(sp$,15-len(vn$))
12660 pl$ = pl$+left$(sp$, 4-len(pl$))
12670 ot$ = ot$+left$(sp$,15-len(ot$))
12680 sr$ = sr$+left$(sp$,15-len(sr$))
12690 be$ = be$+left$(sp$,10-len(be$))
```

```
12700 te$ = te$+left$(sp$,13-len(te$))
12710 rc$ = nn$+vn$+pl$+ot$+sr$+be$+te$
12720 return
12730 rem*************************************************
12740 :
12750 rem**************** datenausgeabe *******************
12760 print: print
12770 print " * name     :  ";nn$
12780 print " * vorname  :  ";vn$
12790 print " * plz      :  ";pl$
12800 print " * ort      :  ";ot$
12810 print " * strasse  :  ";sr$
12820 print " * beruf    :  ";be$
12830 print " * telefon  :  ";te$
12840 print
12850 return
12860 rem*************************************************
12870 :
12880 rem**************** neue datei anlegen **************
12890 print
12900 input " * name der neuen datei  :";dn$
12910 if len(dn$)>10 then print chr$(145);: goto 12900
12920 input " * anzahl der datensaetze :";rc
12930 if rc <1 then print chr$(145);: goto 12920
12940 print " * datensatzlaenge : 88 zeichen"
12950 print
12960 print " * speicherbedarf  :";rc*88;"byte"
12970 print
12980 if rc*88 <= 150000 then 13020
12990 print " * fehler !! speicherbedarf zu gross !!"
13000 gosub 12460
13010 return
13020 print " * arbeitsdiskette einlegen"
13030 print
13040 get a$: if a$ = "" then 13040
13050 print " * lege indexdatei an"
13060 open 4,8,2,dn$+".index,s,w"
13070 gosub 12350
13080 if fe <> 0 then gosub 12460: close 4: return
13090 print#4, rc
```

```
13100 print#4, 0
13110 print#4, 0
13120 close 4
13130 print
13140 print " * lege relative datei an"
13150 open 1,8,2,dn$+",1,"+chr$(88)
13160 gosub 12350
13170 if fe <> 0 then close 1: gosub 12460: return
13180 open 2,8,15
13190 gosub 12350
13200 if fe <> 0 then close 1: close 2: gosub 12460: return
13210 print: gosub 12410
13220 print#2, "p"+chr$(2)+chr$(lb)+chr$(hb)+chr$(1)
13230 print#1, chr$(255)
13240 close 1: close 2
13250 gosub 12350
13260 gosub 12460
13270 return
13280 rem*********************************************
13290 :
13300 rem**************** alte datei laden ***************
13310 print: print:dn$=""
13320 input " * namen alte datei :";dn$: if len(dn$)=0 then
      10130
13330 open 4,8,2,dn$+".index,s,r"
13340 gosub 12350
13350 if fe <> 0 then close 4: gosub 12460: goto 13310
13360 input#4, ds
13370 input#4, an
13380 input#4, lo
13390 dim in$(ds)
13400 if an = 0 then 13440
13410 for i = 1 to an
13420 input#4, in$(i)
13430 next i
13440 close 4
13450 open 1,8,2,dn$+", ,"+chr$(88)
13460 open2,8,15
13470 return
13480 rem*********************************************
```

D21.0 Sortieren und Suchen in einer sortierten Datei

Daten lassen sich wesentlich schneller in einer sortierten Datei finden, da sie immer in einer bestimmten Reihenfolge abgelegt sind. Zahlen liegen z.B. von der kleinsten zur größten Zahl und Zeichensätze alphabetisch geordnet vor. Jeder neue Eintrag muß in diese Datei wieder neu einsortiert werden.
Das Sortieren von Daten ist in der Datenverarbeitung ein altes Problem. Es gibt heute unzählige Sortiermethoden, die sich in Programmaufwand und Verarbeitungsgeschwindigkeit erheblich unterscheiden. Eine verbreitete Suchroutine ist das sogenannte „Bubble-Sort" (Sort=Sortieren, bubble=Blase), da es leicht zu verstehen und programmtechnisch einfach aufgebaut ist.
An einem kleinen Beispiel möchte ich Ihnen das Sortierverfahren erläutern. In den Feldern A(1) - A(3) liegen drei (unsortierte) Zahlen:

A(1) = 9
A(2) = 1
A(3) = 5

Im 1. Durchgang wird die kleinste Zahl der Liste gesucht und in das 1. Feld A(1) gelegt:

- Vergleiche A(1) mit A(2).
- Ist A(2) kleiner als A(1), dann vertausche beide Inhalte.
- Anschließend vergleichen Sie A(1) mit A(3) und tauschen gegebenenfalls die Inhalte.

Nach dem 1. Durchgang sieht die Liste jetzt folgendermaßen aus:

A(1) = 1
A(2) = 9
A(3) = 5

Das Vertauschen zweier Variableninhalte ist nicht direkt, sondern nur über eine Hilfsvariable möglich:

H = A(1)
A(2) = A(1)
A(1) = H

Der 2. Durchgang hat die Aufgabe, die zweitkleinste Zahl aus der Liste in das Feld A(2) zu legen. Da sich im Feld A(1) schon die kleinste Zahl befindet, muß nur noch ab Feld 2 (A(2)) geprüft werden!

- Vergleiche A(2) mit A(3).
- Ist A(3) kleiner als A(2), vertausche beide Inhalte

Um eine Liste von 100 Daten zu sortieren sind immerhin 99 solcher Durchgänge notwendig. Das nachfolgende, kurze Unterprogramm sortiert nummerische Felder in aufsteigender Reihenfolge:

```
1000 rem ***********************************************
1010 rem Anzahl. der Daten = m
1020 rem Daten = a(1) bis a(m)
1030 :
1040 for i = 1 to m-1
1050 for j = i+1 to m
1060 if a(i) <= a(j) then 1080
1070 h=a(i): a(i)=a(j): a(j)=h
1080 next j
1090 next i
1100 return
1110 rem ***********************************************
```

Probieren Sie das Sortierprogramm einmal aus:

```
10 m=100
20 dim a(m)
30 for z=1 to m
40 a(z)=rnd(1)*1000
50 next z
60 print "unsortierte liste"
70 for i= 1 to m
80 print a(i);
90 next i
100 print chr$(147)
110 print "bitte warten ... sortiere !"
120 gosub 1000
```

```
130 print "sortierte liste"
140 for i= 1 to m
150 print a(i);
160 next i
170 end
```

In den Programmzeilen 30-50 werden zufällige, reale Zahlen dem Array A(..) zugeordnet und anschließend ausgedruckt. Nach dem Sortiervorgang, der schon bei 100 Daten rund 1 Minute und 15 Sekunden dauert, wird die sortierte Liste ausgegeben. Das Sortieren von Zeichenketten (Strings) arbeitet nach dem gleichen Verfahren, da für den Rechner Zeichen auch nur Zahlen sind. Wie sucht man nun in einer sortierten Liste ?

Erinnern Sie sich noch an das alte Zahlenratespiel? Sie sollten eine Zahl zwischen 1 und 100 raten. Anschließend wurde Ihnen gesagt, ob Ihre geratene Zahl zu groß oder zu klein ist. Sicherlich! Um möglichst schnell die richtige Zahl zu erraten, verfahren Sie nach dem Prinzip, immer die Mitte der zur Verfügung stehenden Zahlen zu raten. Also zuerst die 50. Ist die gesuchte Zahl kleiner, so liegt sie zwischen 1 und 50. Sie fragen nun nach der 25 (wieder die Mitte). Ist die gesuchte Zahl nun wieder größer, so liegt sie zwischen 25 und 50. Und so weiter.....
Auf diese Weise ist es möglich, die richtige Zahl zwischen 1 und 100 in 7 Versuchen zu raten. Mit der Gleichung

VERSUCHE = INT(LOG(MÖGLICHKEITEN) / LOG(2)+1)

läßt sich die maximale Anzahl der Versuche berechnen. Bei einer Zahl zwischen 1 und 1000 sind es

INT(LOG(1000) / LOG(2)+1) = 10

10 Versuche. Ein solches Suchverfahren ist nur dann möglich, wenn die zu ratenden Zahlen in einer sortierten Form (geordnet) vorliegen. Grafisch sieht diese Suchmethode aus wie ein Baum: man spricht auch von einem binären Baum. Hier das Suchen nach einer Zahl zwischen 1 und 31.

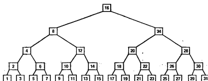


Abb. 27 Binärer Baum

Nach maximal fünfmaligem Suchen ist die richtige Zahl gefunden. Wie ist nun dieses Prinzip auf eine Datei anzuwenden, wenn in einer relativen Datei Datensätze alphabetisch geordnet, z.B. nach Namen vorliegen.

1. Record: Andresen/Herbert
2. Record: Berger/Manfred
3. Record: Carlsen/Thomas

usw ...

Als Beispiel soll eine alphabetisch geordnete, relative Datei mit 31 Records dienen. Um den Datensatz für „Carlsen" zu finden, wird entsprechend dem binären Baum zuerst Record 16 eingelesen und geprüft, ob der Name mit dem gesuchten Namen übereinstimmt.

IF NAMES = RECORD$ THEN .... gefunden !

Wurde der richtige Record nicht gefunden, so wird geprüft, ob der in RECORDS befindliche Name „kleiner" oder „größer" als der gesuchte Name ist (für den Computer sind Zeichen auch nur „Zahlen", deren Wertigkeit von 65 (für A) bis 90 (für Z) steigend ist). Daraufhin kann entschieden werden, in welchem Teil des Baumes weiter gesucht werden muß. Ein kleines Programm soll Ihnen die Berechnung verdeutlichen.

```
10 open 1,8,2,"test,1,"+chr$(..)
20 open 2,8,15
30 an = 31
40 c = int(log(an)/log(2)+1)
50 s = c - 1: c = 2^c. z = c/2
60 input "name :";n$
70 if s<0 then print "nicht vorhanden":goto 30
80 s = s - 1
90 print#2, "p"+chr$(2)+chr$(z)+chr$(0)+chr$(1)
100 input#1, rc$
110 na$ = left$(rc$,15)
120 if n$ = na$ then 170
130 if n$ < na$ then z = z-2^s: goto 70
140 z = z+2^s
150 if z>c then print "nicht vorhanden": goto 30
160 goto 70
170 print rc$
180 goto 30
```

Nach Errechnen der maximalen Anzahl der Suchvorgänge (c) wird der Zeiger (z) berechnet, der in diesem Fall auf Record 16 zeigt. Der Record wird dann eingelesen und der Name (n$) mit dem Namen im Datensatz (na$) verglichen. Stimmen sie überein, wird der gesamte Datensatz (rc$) ausgedruckt. Ist er kleiner, wird der Zeiger (z) um den Faktor 2^s erniedrigt. Ist das gesuchte Wort größer als na$, wird er um den Faktor 2^s erhöht. Ist s kleiner als Null, so ist das gesuchte Wort nicht vorhanden (alle Möglichkeiten durchgesucht).
An dieser Stelle werden Sie sicher bemerkt haben, daß der Name binärer Baum nicht von ungefähr kommt. Die Anzahl der verwendeten Records ist immer eine Zahl, die binär nur aus „1" besteht.

1111	=	15
11111	=	31
111111	=	63
1111111	=	127

Probleme treten auf, wenn die Anzahl der Records eine hiervon abweichende Zahl ist. Zur Lösung des Problems wird die nächsthöhere Zweier-Potenz benutzt. Sind es z.B. 43 Records, so erhöhen Sie die Recordzahl auf 63. Nun haben

Sie aber tatsächlich nur 43 und nicht 63 Records und ein nicht vorhandener Record kann nicht eingelesen werden (Fehlermeldung). Soll z.B. der Record 43 gesucht werden, so müßte nach dem Baumprinzip zuerst Record 32 und dann Record 48 (!!) eingelesen und geprüft werden. Da diese Rocordnummer aber größer als die vorhandene Anzahl der Records ist, liegt die gesuchte Recordnummer auf jeden Fall niedriger! Das bedeutet, vor Einlesen eines Records ist zu prüfen, ob dieser Record vorhanden ist.

```
95 it z > an then z=z-2^s
```

Mit dieser Methode lassen sich aus jeder geordnet vorliegenden Datei sehr schnell Daten finden. Eine alphabetisch geordnete Index-Datei, könnte z.B. so verwaltet werden
Auch ein Vokabelprogramm könnte so aufgebaut sein. In einer relativen Datei liegen die Vokabeln mit ihren Übersetzungen alphabetisch geordnet vor. Nach der binären Suchmethode kann direkt in der relativen Datei gesucht werden.

Durch Verwendung von relativen Dateien lassen sich mit dem C=64 semiprofessionelle Anwendungen realisieren, mit denen auch Probleme im „small-business"-Bereich gelöst werden können. Neben den bisher behandelten sequentiellen und relativen Dateien besteht weiterhin die Möglichkeit, Dateiprinzipien selber zu entwickeln.

D22.0 Direkter Zugriff auf die Diskette

Auf einen Block der Diskette kann auch direkt zugegriffen werden. Der Inhalt eines Blocks wird in einen der fünf internen Datenspeicher der Diskettenstation eingelesen. Jeder dieser Datenspeicher kann 256 Bytes aufnehmen. Die Datenspeicher werden u.a. für die Speicherung der BAM benutzt. Beim direkten Zugriff wird ein Block komplett in einen der Datenspeicher geladen. Innerhalb des Datenspeichers können die Daten geändert und in einen beliebigen Block zurückgeschrieben werden.

D22.1 Welcher Datenspeicher?

Ein Kanal muß einem der 5 Datenspeicher zugeordnet werden, in dem die Daten abgelegt werden sollen. Hierzu wird ein spezieller Befehl benutzt: mit dem Nummernzeichen (#) wird ein Datenspeicher ausgewählt. Ein Index hinter

dem Nummernzeichen benennt einen der 5 Datenspeicher (# 0- # 4). Wird nur das Nummernzeichen (#) eingegeben, sucht sich das DOS selber einen freien, nicht benutzten Datenspeicher. Da auch das DOS Datenspeicher benutzt, bietet es sich an, immer durch das DOS den Datenspeicher auswählen zu lassen.

```
OPEN 1,8,2, "#"
```

Um festzustellen, welcher der 5 Speicher benutzt worden ist, lesen Sie das 1.Byte aus dem Datenspeicher ein.

```
GET#1, A$
```

Bitte beachten Sie, daß eine GET # -Anweisung kein Nullbyte einlesen kann (A$ wäre dann ein Leerstring)!! Durch „Hinzuaddieren" von CHR$(0) wird dieser Mangel umgangen.

```
PRINT ASC(A$+CHR$(0))
```

Die erhaltene Zahl ist identisch mit dem belegten Datenspeicher. Wird ein spezieller Datenspeicher ausgewählt, z.B. Datenspeicher 3 (#3), und dieser wird bereits benutzt, so wird die Fehlermeldung NO CANNEL ausgegeben.

D22.2 Block in Datenspeicher laden

Nachdem ein Datenspeicher ausgewählt wurde, kann der Inhalt eines Blocks von der Diskette in den Speicher kopiert werden. Dazu wird ein weiterer Kanal benötigt, der die Informationen an das DOS weiterleitet.

```
10 open 1,8,15
20 open 2,8,2, "#"
```

Der Kanal 2 führt in den Datenspeicher und Kanal 1 (Befehlskanal) in das DOS. Dem DOS wird nun mitgeteilt, daß ein Block gelesen (Block-Read) und über welchen Kanal die Daten geleitet werden sollen (in den Datenspeicher über Kanal 2). Als weitere Information ist natürlich wichtig, welcher Block einzulesen ist.

```
30 print#1, "U1 2 0 18 0"
```

Das Befehlsformat des Block-Read Befehls kann entweder mit „U1" oder „B-R" bezeichnet werden. Die „2" ist die Kanalnummer zum Datenspeicher und mit der „0" wird die Laufwerkseite bei einem doppelseitigen Diskettenlaufwerk ausgewählt. „18 0" geben Spur und Sektor (zeigt hier auf die BAM) des einzulesenden Blocks an. Bei der nur einseitig betriebenen Diskettenstation ist immer eine „0" anzugeben.
Mit diesem Befehl ist der Block (18/0) in den Datenspeicher kopiert worden. Alle 256 Daten können nun mit GET# ausgelesen werden.

D22.3 Positionieren im Datenspeicher

Innerhalb des Datenspeichers kann auf ein bestimmtes Byte gezeigt werden, das dann mit der nächsten GET#-Anweisung eingelesen wird. Mit dem „Buffer-Pointer" (B-P)-Befehl wird auf ein bestimmtes Byte gezeigt.

```
40 print#1, "b-p 2144"
```

Hier wird auf das 144. Byte im Datenspeicher, der über den Kanal 2 erreicht wird, positioniert. Das folgende Beispiel liest den Namen der Diskette ein, der ab Byte 144 in der BAM (18/0) liegt. Der Name umfaßt maximal 16 Zeichen (siehe D15.1)

```
10 open1,8,15
20 open2,8,2,"#"
30 print#1,"u1 2 0 18 0"
40 print#1,"b-p 2 144"
50 for i = 1 to 16
60 get#2, a$
70 printa$+chr$(0);
80 next i
90 close1:close2
```

Daten können aus dem Datenspeicher nicht nur gelesen, sondern es können auch Daten in den Speicher geschrieben werden. Die neuen Daten werden ab der Stelle in den Puffer geschrieben, auf den der Pointer zeigt.

```
PRINT#1, "B-P 2 24"
```

PRINT#2, "HALLO"

„HALLO" steht dann in den Bytes 24 bis 28 des Datenspeichers. Alle Änderungen erfolgten bisher nur im Datenspeicher. Jetzt muß der geänderte Block wieder zurück auf die Diskette geschrieben werden.

D22.4 Datenspeicher zurückschreiben

Mit dem „Block-Write"-Befehl wird der Inhalt eines Datenspeichers in definierten Block (Sektor) der Diskette geschrieben.

PRINT#1, "U2 2 0 18 0"

Das Befehlsformat ist identisch mit dem des Block-Read-Befehls. "U2" kann auch durch „B-W" ersetzt werden. Somit ist es auch möglich, den Diskettennamen zu ändern.

```
10 open1,8,15
20 open2,8,2,"#"
30 print#1,"u1 2 0 18 0"
40 print#1,"b-p 2 144"
50 for i = 1 to 16
60 get#2, a$
70 printa$+chr$(0);
80 next i
90 print
100 input "disk-name :";dn$
110 if len(dn$)>16 then printchr$(145);:goto100
120 if len(dn$)<16 thendn$=dn$+chr$(160):goto 120
130 print#1,"b-p 2 144"
140 print#2,dn$
150 print#1,"u2 2 0 18 0"
160 close2:close1
```

Nach Eingabe des neuen Diskettennamens wird die Länge überprüft (16 Zeichen) und gegebenenfalls mit SHIFT/SPACE (chr$(160)) aufgefüllt. Der Pointer muß neu positioniert werden, damit auch tatsächlich ab Byte 144 geschrieben wird. Abschließend wird der gesamte Block zurückgeschrieben.

Mit Hilfe dieser Befehle lassen sich eigene Dateistrukturen entwickeln, die ganz auf Ihre Belange zugeschnitten werden können.

Mit den Befehlen Block-Read, Block-Write und Buffer-Pointer stehen wirklich Tür und Tor für Dateien jeder nur denkbaren Art offen. Alleine durch die Verwendung des Buffer-Pointers (zeigen auf ein bestimmtes Byte) lassen sich auch Teile eines Datensatzes einlesen, ohne den gesamten Datensatz in den Rechner zu laden. Ebenso können bestimmte Teile eines Datensatzes geändert werden. Die Länge eines Datensatzes ist so flexibel, daß Längen zwischen einem und 169984 Byte denkbar sind. Ganz wie Sie wollen. Es ist wirklich eine sehr „starke" Sache.

Das folgende kleine und einfache Beispielprogramm zeigt die Anwendung einer sogenannten Direkt-Zugriff-Datei. Benutzt werden die ersten 17 Spuren (je 21 Sektoren) als Speicher, wobei ein Datensatz die Länge eines Blockes (256 Byte) besitzt. Es stehen insgesamt 17*21 = 357 Blöcke für Daten zur Verfügung. Ab Zeile 1000 ist ein Umrechnungsprogramm zu finden, welches die Block-Nummer in Spur und Sektor umrechnet. Ein (Daten-) Block ist immer dann nicht belegt, wenn sich im 1. Byte ein CHR$(1) befindet. Aus diesem Grunde werden bei der Eröffnung einer neuen Datei alle Blöcke der ersten 17 Spuren mit einem CHR$(1) beschrieben (ab Zeile 300). Verwenden Sie deshalb bitte eine nicht benutzte (formatierte) Diskette, da die Inhalte der ersten 17 Spuren zerstört werden.

```
10 rem*************** m e n u e **********************
20 poke 53280,0
30 open 1,8,15
40 open 2,8,2,"#"
50 print chr$(147)
60 print " --- datei in direktzugriff ---"
70 print: print
80 print " *  (1)  neue datei erstellen ......"
90 print " *  (2)  datensatz einlesen ........"
100 print " *  (3)  datensatz schreiben......."
110 print " *  (4)  datensatz loeschen ........"
120 print " *  (5)  programm-ende ............"
130 print: print
140 print " *       bitte waehlen sie ........"
150 get b$: if b$ = "" then 150
```

```
160 b=val (b$)
170 if b<1 or b>5 then 150
180 on b gosub 300,400,600,800,950
190 goto 50
200 rem***********************************************
210 :
300 rem*************** neue datei *********************
310 for nr=1 to 357
320 gosub 1000:printchr$(145);nr
330 print#1, "u1 2 0";sp;se
340 print#2, chr$(1)
350 print#1, "u2 2 0";sp;se
360 next nr
370 return
380 rem***********************************************
390 :
400 rem**************** lesen ************************
410 print chr$(147)
420 print " --- datensatz einlesen ---"
430 print: print
440 input "datensatznr. (1-357):";nr
450 if nr<1 or nr>357 then print chr$(145);:goto 440
460 print: print
470 gosub 1000
480 print#1, "u1 2 0";sp;se
490 get#2, a$
500 if a$ = chr$(1) then print " * l e e r *": goto 540
510 printa$;
520 get#2,a$:if a$ = "" chr$(255) then 540
530 printa$;:goto 520
540 get b$: if b$= "" then 540
550 return
560 rem***********************************************
570 :
600 rem**************** erstellen ********************
610 print chr$(147)
620 print " --- datensatz erstellen ---"
630 print: print
640 input "datensatznr. (1-357):";nr
```

```
650 if nr<1 or nr>357 then print chr$(145);:goto 640
660 print: print
670 gosub 1000
680 input " * info 1";i1$
690 input " * info 2";i2$
700 input " * info 3";i3$
710 ig$=i1$+chr$(13)+i2$+chr$(13)+i3$+chr$(255)
720 print#1, "u1 2 0";sp;se
730 print#2, ig$
740 print#1, "u2 2 0";sp;se
750 return
760 rem*********************************************************
770 :
800 rem**************** loeschen ***********************
810 print chr$(147)
820 print " --- datensatz loeschen ---"
830 print: print
840 input "datensatznr. (1-357):";nr
850 if nr<1 or nr>357 then print chr$(145);:goto 840
860 gosub 1000
870 print#1, "u1 2 0";sp;se
880 print#2, chr$(1)
890 print#1, "u2 2 0";sp;se
900 return
910 rem*********************************************************
920 :
940 rem****************** e n d e **********************
950 rem
960 print chr$(147)
970 close 1: close 2: end
980 rem*********************************************************
990 :
1000 rem************ umrechnung in sp/se **************
1010 sp=int(nr/21)
1020 if nr/21 <> sp then sp=sp+1:goto 1040
1030 se=20:goto 1050
1040 se=int((nr/21-int(nr/21))*21+0.5)-1
1050 return
1060 rem*********************************************************
```

D22.5 Block als „belegt“ oder „frei“ kennzeichnen

Alle Blöcke, die nicht einem File zugeordnet sind, werden in der BAM (siehe D15.1) als „frei“ bezeichnet. Um einen Block in der BAM als belegt zu kennzeichnen, steht der „Block-Allocate“ Befehl zur Verfügung.

```
PRINT#1, "B-A 0 1 1"
```

Der Block in Spur 1 und Sektor 1 wird damit in der BAM als belegt gekennzeichnet (Die „0“ bezieht sich wieder auf die Laufwerkseite). Ist dieser Block bereits belegt, wird die Fehlermeldung „NO BLOCK Spur Sektor“ ausgegeben. Die Angabe Spur/Sektor zeigt dann auf den nächsthöheren, freien Block. Ist kein weiterer Block frei, steht bei Spur und Sektor jeweils der Wert Null.

Damit alle im obigen Programm verwendeten Blöcke in der BAM als belegt gekennzeichnet werden, geben Sie bitte folgendes Programm ein:

```
10 rem*** blocke in der bam als belegt kennzeichnen ****
20 open 1,8,15
30 for nr =1 to 357
40 gosub 1000
50 print#1, "b-a 0";sp;se
60 input#1, a$,b$,c,d, :print a$;" "b$;c;d;nr
70 if a$=65 then print sp;se;;"belegt": goto 90
80 next nr
90 close1: end
1000 rem************* umrechnung in sp/se ***************
1010 sp=int (nr/21)
1020 if nr/21<> sp then sp=sp+1:goto 1040
1030 se=20:goto 1050
1040 se=int((nr/21-int(nr/21))*21+0.5)-1
1050 return
1060 rem*********************************************
```

Alle 357 Blöcke werden in der BAM als belegt gekennzeichnet. Gleichzeitig wird der Fehlerkanal ausgelesen und das Programm wird unterbrochen, wenn ein Block schon belegt ist. Aus der Angabe von Spur/Sektor ist der nächste freie Block zu entnehmen.

Analog zum „Block-Allocate" Befehl gibt es den „Block-Free"-Befehl, der einen Block in der BAM als frei kennzeichnet.

PRINT#1, "B-F 0 1 1"

Anzumerken ist noch, daß die mit „B-A" als belegt gekennzeichneten Blöcke durch den „Validate"-Befehl wieder freigegeben werden, da sie nicht mit einem File verkettet sind.

D22.6 Einlesen der Directory in ein Programm

Es ist sehr lästig, daß nach Laden der Directory ein BASIC-Programm gelöscht wird. Durch die Möglichkeit, direkt auf die Directory-Blöcke zuzugreifen, kann die Directory eingelesen werden, ohne ein Programm zu zerstören. Der Aufbau der Directory ist in D15.0 beschrieben.

```
60000 rem ** einlesen der directory ***
60010 open 1,8,15
60020 open 2,8,2," "
60030 sp=18:se=1
60040 print#1, "u1 2 0"sp;se
60050 get#2, a$:sp=asc(a$+chr$(0))
60060 get#2, a$:se-asc(a$+chr$(0))
60070 for z=0to7
60080 print#1, "b-p 2";z*32+5
60090 ff$=""
60100 for i=1 to 16
60110 get#2, a$:if a$= "" then a$=chr$(0)
60120 ff$=ff$+a$
60130 next i
60140 printff$,
60150 next z
60160 if sp<>0 then 60040
60170 close1:close2
60180 rem******************************
```

D23.0 Memory- und User-Befehle

Eine Reihe weiterer Befehle, die eine umfassende Kenntnis der Maschinensprache voraussetzen, stehen dem Programmierer zur Verfügung. Sie sollen an dieser Stelle nur der Vollständigkeithalber erwähnt werden. Es sind Befehle, die direkt in das DOS greifen, um dort Veränderungen durchzuführen und um Befehle, die im Datenspeicher abgelegte Maschinenprogramme dort ausführen.

"Block-Execute"	(B-E)	: PRINT#1, "B-E 2 0 10 1" Ein Maschinenprogramm, das in Spur 10/Sektor 1 liegt, wird in einen Datenspeicher geladen und ausgeführt.
"Memory-Read"	(M-R)	: PRINT#1, "M-R";CHR$(LB);CHR$(HB) Jede Adresse des DOS kann ausgelesen werden. Die Adresse wird in Low- und Highbyte angegeben.
"Memory-Write"	(M-W)	: PRINT#1, "M-W";CHR$(LB);CHR$(HB);CHR$(Menge);CHR$(DATA1).... In den DOS-(RAM) Speicher können Daten geschrieben werden. LB/HB ist die Adresse, ab der die Daten im Speicher abgelegt werden sollen. „Menge" ist die Anzahl der folgenden Daten (DATAn)..
"Memory-Execute"	(M-E)	: PRINT#1, "M-E";CHR$(LB);CHR$(HB) Ein Maschinenprogramm, das ab der Adresse (LB/HB) liegt, wird ausgeführt.

"User"-Befehle

User-Befehle „springen" in das DOS- oder den Datenspeicher 2 und führen dort liegende Programme aus. Zwei der 10 User-Befehle sind bisher schon benutzt worden:

U1 entspricht dem "Block-Read"-Befehl
U2 entspricht dem "Block-Write"-Befehl
U3-U9 springen in Datenspeicher 2
U: setzt die Diskettenstation in den Einschaltzustand zurück (RESET).

D23.1 Ändern der Gerätenummer

Die Gerätenummer der Diskettenstation ist vom Werk aus auf die Nummer 8 festgelegt worden. Zwei Möglichkeiten bestehen, um die Gerätenummer zu ändern: hardware- oder softwaremäßig.

Hardware:

Bei der Hardwareumstellung der Gerätenummer ist es notwendig, einen Eingriff in der Diskettenstation durchzuführen (Garantieverlust!!). Auf der Platine der Diskettenstation befinden sich oberhalb der Widerstände R32 und R36 zwei Lötbrücken. Durch Zertrennen der vorderen Lötbrücke wird die Gerätenummer fest auf Nummer 9 umgestellt.

Software:

```
10 open 1,8,15
20 input " * geraetenummer :";n
30 print#1, "m-w";chr$(119)chr$(0);chr$(2)chr$(n+32);
   chr$(n+64)
40 close 1
```

Die neue Gerätenummer bleibt bis zu einem Reset erhalten.

KASSETTE

E1.0 Ein billiger Massenspeicher

Der Kassettenrecorder ist das einfachste Mittel zur Programm- und Datensicherung. Die Daten werden auf handelsübliche Tonbandkassetten gespeichert. Von hier können sie jederzeit wieder in den Speicher des C=64 geladen werden. Die Übertragungsgeschwindigkeit zwischen Datasette und Computer ist sehr gering, so daß sich große Lade- und Speicherzeiten ergeben. Dafür ist der Anschaffungspreis mit ca. 120.- DM sehr niedrig.

E2.0 Bits werden zur Frequenz

Das Band der Kassette ist mit einer magnetisierbaren Schicht belegt, deren Teilchen unterschiedlich magnetisiert werden können. Je nach Art der Magnetisierung kann festgestellt werden, um welchen Ton es sich handelt. Für die Programm- und Datenspeicherung werden nur zwei Töne (Zustände) benötigt: einer für den Zustand „0" und ein anderer für den Zustand „1". Mit einer Frequenz von 2000 Hz ('0') und 4000 Hz ('1') werden die Bits verschlüsselt und auf der Kassette abgelegt. Hören Sie sich einmal eine Programmkassette auf einem Recorder an. Das „Gepiepse" sind die Bits, verschlüsselt in Frequenzen. Beim Laden von der Kassette findet eine Decodierung von Frequenz in Ladungszustand statt.

Alle Daten werden, bevor sie auf das Band geschrieben werden, in einen Zwischenspeicher abgelegt. Erst wenn der gesamte Speicher mit Daten gefüllt ist,

werden sie „vertont" und zur Datasette übertragen. Der sogenannte Kassettenpuffer befindet sich in den Adressen 828 - 1019 (192 Byte). Beim Lesen von der Datasette werden ebenfalls erst die Daten im Kassettenpuffer gesammelt und dann zum Computer weitergeleitet.

E3.0 Laden und Speichern

Zum Laden und Speichern von Programmen besitzt der C=64 die beiden Befehle LOAD (Laden) und SAVE (Sichern). Als Gerätenummer ist die Datasette fest auf die Nummer 1 gelegt. Anhand dieser Gerätenummer kann der Computer unterscheiden, welches externe Gerät angesprochen werden soll.

```
LOAD "name", 1
SAVE "name", 1
```

Der Programmname kann maximal 16 Zeichen beinhalten. Nach diesem Namen sucht der Computer, bis er ihn gefunden hat und dann lädt er das folgende Programm ein. Beim Laden und Speichern von Programmen gibt es zwei generelle Unterschiede.

1) LOAD/SAVE von BASIC-Programmen.

   BASIC-Programme sind bestimmt durch die Zeiger 43/44 (BASIC-Anfang) und 45/46 (BASIC-Ende) (siehe A4.1). Mit dem Befehl SAVE „name", 1 wird dieser Bereich als Programm geSAVEt. Beim Laden mit LOAD „name", 1 wird das Programm wieder direkt an den BASIC-Anfang abgelegt.

2) LOAD/SAVE von Maschinenprogrammen

   Maschinenprogramme werden immer an den Ort abgelegt, von wo aus sie aufrufbar sind (z.B SYS 49152). Sie dürfen nicht an den BASIC-Anfang gelegt werden und müssen darum besonders gekennzeichnet sein. Mit LOAD „name", 1,1 sind Maschinenprogramme absolut ladbar. Damit das Maschinenprogramm tatsächlich in seinen richtigen Bereich geladen wird (z.B. 49152), ist der BASIC-Anfang-Zeiger auf 49152 verstellt worden und der Computer hat scheinbar keinen Speicherplatz mehr zur Verfügung! Deshalb ist es notwendig, den BASIC-Anfang nach dem Laden eines Maschinenprogramms mit NEW zu normalisieren.

Beim Laden und Speichern von Programmen auf der Datasette kann die Gerätenummer entfallen. Sowie keine Gerätenummer angegeben wird, setzt das Betriebssystem voraus, daß von der Datasette geladen oder gespeichert werden soll. Wird kein Programmname angegeben, so wird das nächste, sich auf der Kassette befindliche Programm (an den BASIC-Anfang) geladen.

Um sicher zu gehen, daß alle Daten richtig auf das Band gekommen sind, bietet der C=64 die Möglichkeit, das Programm innerhalb des Computers mit einem auf der Kassette befindlichen Programm zu vergleichen.

```
VERIFY "name",1
```

Sind beide Programme identisch (es liegen keine SAVE-Fehler vor), meldet sich der C=64 mit einem „OK". Liegen Fehler vor, wird ein „VERIFY-ERROR" ausgegeben.

E3.1 Der Programmkopf

Vor dem eigentlichen Programm wird auf der Kassette ein Programmkopf (Header) abgelegt. Im Header sind folgende Informationen enthalten:

Byte 0	: Programmtyp
Byte 1	: Startadresse (Lowbyte)
Byte 2	: Startadresse (Highbyte)
Byte 3	: Endadresse (Lowbyte)
Byte 4	: Endadresse (Highbyte)
Byte 5-20	: Filename (16 Zeichen)

Anhand des Headers kann nach dem Filenamen (Programmname) gesucht werden. Der Header wird, wie alle Daten von der Kassette, erst im Kassettenpuffer zwischengespeichert. Stimmt der gesuchte Programmname mit dem Namen aus dem Header überein, so wird das Programm geladen. Andernfalls wird weiter gesucht, bis der nächste Header auftritt und entsprechend verglichen wird.
Aus dem Programmtyp (Byte 0) geht im wesentlichen hervor, um welche Art File es sich handelt. Das Byte 0 kann fünf verschiedene Werte enthalten.

Byte 0 = 1 : BASIC-Programm
Byte 0 = 2 : Datenblock
Byte 0 = 3 : Maschinenprogramm
Byte 0 = 4 : Sequentielle Datei
Byte 0 = 5 : Bandende (End of Tape)

Ein BASIC-Programm (1) wird immer an den BASIC-Anfang geladen, Maschinenprogramme (3) immer absolut an die in Byte 1 und Byte 2 angegebene Adresse. Mit einer 4 ist gekennzeichnet, daß eine Datei folgt.
Der Header wird, wie schon erwähnt, komplett in den Kassettenpuffer geladen. In Adresse 828 (Anfang Kassettenpuffer) liegt der Programmtyp usw.. Wird der Ladevorgang nach der FOUND-Meldung mit RUN/STOP unterbrochen (der Header ist in den Kassettenpuffer eingelesen), können alle Informationen des Headers aus dem Kassettenpuffer gelesen und angezeigt werden. Ein Programm übernimmt diese Arbeit.

```
100 rem************ cassetten header auslesen ***********
110 poke 53280,0
120 print chr$(147)
130 print " ----- cassetten header auslesen -----"
140 dim a(20)
150 print: print
160 :
170 rem************ header suchen und einlesen **********
180 sys 63276
190 rem**********************************************
200 :
210 for i = 0 to 20
220 a(i)=peek(828+i)
230 next i
240 :
250 rem********** bestimmen des programmtyps ************
260 ty$(1)="basic-programm"
270 ty$(2)="datenblock"
280 ty$(3)="maschinenprogramm"
290 ty$(4)="sequentielle datei"
300 ty$(5)="end of tape block"
310 print: print
320 print " * programmtype   : ";ty$(a(0))
```

```
330 rem*************************************************
340 :
350 rem************* start- und endadresse ************
360 sa=a(2)*256+a(1)
370 ea=a(4)*256+a(3)
380 pl=ea-sa
390 print
400 print " * startadresse   :";sa
410 print " * endadresse     :";ea
420 print " * programmlaenge :";pl
430 rem*************************************************
440 :
450 rem******************* programmname ***************
460 for i = 5 to 20
470 na$=na$+chr$(a(i))
480 next i
490 print
500 print " * programmname   : ";chr$(18);na$: na$=""
510 print: print
520 print " * naechsten header suchen (j/n) ? *"
530 get a$: if a$ = "" then 530
540 if a$ = "j" then 180
550 if a$ <>"n" then 530
560 end
```

Damit nicht nach dem Laden des Headers in den Kassettenpuffer die RUN/STOP-Taste gedrückt werden muß (vergessen Sie es, wird das Programm überschrieben), wird in Zeile 180 (SYS 63276) der Teil des Betriebssystems aufgerufen, der den Header einliest. Ist der Header gefunden, wird zurück in das BASIC-Programm gesprungen (die Datasette stoppt). Alle Informationen des Headers werden angezeigt und es kann entschieden werden, ob weitere Header gesucht werden sollen.

E4.0 Motorsteuerung

Sie werden sicherlich bemerkt haben, daß trotz gedrückter PLAY-Taste die DATASETTE bei der FOUND-Meldung anhält. Daraus ist zu schließen, daß der Recordermotor ein- und ausschaltbar ist. Ebenso werden die Tasten der

Datasette abgefragt und auf ihren Zustand (Taste gedrückt oder nicht) geprüft. Für beide Funktionen ist einmal mehr das Datenrichtungsregister (1) zuständig (siehe A5.0). Im Normalzustand besitzt Adresse 1 den Inhalt 55.

Adresse 1 : 0011 0111 = 55

Bit 4 ist für die Datasetten-Tastenabfrage verantwortlich. Ist keine Taste gedrückt, ist das Bit 4 gesetzt ('1'). Ist eine Taste betätigt, wird es gelöscht ('0'). Wird eine Taste gedrückt, soll die Datasette natürlich laufen. Das Bit 5 wird ebenfalls gelöscht ('0') und der Motor ist in Betrieb. Durch Setzen von Bit 5 kann der Motor angehalten werden. Leider ist das Ein- und Ausschalten des Motors nicht ganz so einfach, denn auch die Tasten der Datasette werden alle 1/60 Sekunde abgefragt. Ist keine Taste gedrückt, setzt das Betriebssystem das Bit 5 der Adresse 1 auf „1" und gleichzeitig die Adresse 192 (Interlock-Register) auf Null. Nur wenn eine Taste gedrückt und das Interlock-Register mit einem Wert ungleich Null geladen ist, kann der Motor ein- bzw. ausgeschaltet werden.

```
POKE 192, 255
POKF 1, PEEK (1) OR 32    setzt Bit 5/Motor aus
```

Der Motor kann nun auch wieder mit

```
POKE 1, PEEK (1) AND 31   löschen Bit 5
```

eingeschaltet werden.
Ist eine der Tasten der Datasette gedrückt, ist Bit 5 und Bit 6 gelöscht (0000 0111 = 7).

```
IF PEEK(1) = 7 THEN PRINT "TASTE GEDRUECKT"
```

Mit der Möglichkeit, den Motor an- und auszuschalten, läßt sich u. a. eine „Kassetten-Directory" realisieren. Es ist schon sehr mühselig, ein Programm auf der Kassette zu suchen. Warum soll dieses nicht der Rechner übernehmen? Dabei wird vereinbart, daß pro Seite einer C-90 Kassette 13 Programme Platz finden sollen (á 3 Minuten). Jedes Programm beginnt an einer bestimmten Stelle der Kassette. Die Zeit, bis die Datasette im schnellen Vorlauf diesen Ort erreicht, wird bestimmt und im Programm festgelegt. Nach Betätigen der schnellen Vorlauftaste wird die vergangene Zeit mit der Suchzeit verglichen. Ist die Zeit erreicht, wird der Motor gestoppt und das Programm kann geladen werden.

```
1000 rem******** kassetten directory (c-90 tape) **********
1010 poke 53280,0: poke 53281,0
1020 print chr$(147);
1030 dim na$(13),in$(13),tm(13)
1040 print "********** kassetten directory ********"
1050 print:print
1060 for i=1 to 13
1070 read na$(i),tm(i),in$(i)
1080 j$=mid$(str$(i),2,3):if len(j$)<2 then j$="0"+j$
1090 print chr$(18);j$;"  >"na$(i);"<";chr$(146);
     "  ";in$(i)
1100 next i
1110 print: print
1120 input "   * welches programm laden ";ln
1130 if ln<1 or ln>13 then print chr$(145);: goto 1120
1140 print "    * suche : ";chr$(18);na$(ln)
1150 if peek(1) = 55 then 1190
1160 print:print "    * press stop on tape  !! *"
1170 if peek(1) <> 55 then 1170
1180 print chr$(145);chr$(145);
1190 print:print "    * press f.fwd on tape !!
1200 if peek(1) <> 7 then 1200
1210 print chr$(145);"    * ok ... bitte warten !!
1220 poke192,255
1230 ti$="000000"
1240 if val(ti$)<=tm(ln) then 1240
1250 poke 1, peek(1) or 32
1260 print chr$(145);"    * press stop on tape   "
1270 if peek(1) <> 55 then 1270
1280 print chr$(145);"    * press play on tape   "
1290 if peek(1) <> 7 then 1290
1300 clr:print chr$(147);:load
1310 :
1320 rem******* daten fuer directory (name/zeit/info) ****
1330 data "                ",0  ,"...............""
1340 data "                ",15 ,"..............."
1350 data "                ",30 ,"..............."
1360 data "                ",43 ,"..............."
1370 data "                ",55 ,"..............."
```

```
1380 data "                ",105,"...............  "
1390 data '                ",115,"..............."
1400 data "                ",124,"..............."
1410 data "                ",133,"..............."
1420 data "                ",142,"..............."
1430 data "                ",151,"..............."
1440 data "                ",200,"..............."
1450 data "                ",209,"..............."
```

In den DATA-Zeilen ab Zeile 1330 befindet sich Platz für 13 Programmeinträge. Hinter dem Namen des Programms (hier noch 16 SPACE) folgt die schnelle Vorlaufzeit in Sekunden, dann ist noch Platz für eine Information zum Programm (z.B. Spiel, Tool). Wollen Sie ein neues Programm auf die Kassette bringen, verfahren Sie nach folgendem Prinzip:

1) Speichern Sie das neue, fertige Programm auf einer extra Kassette ab.
2) Laden Sie die Kassetten-Directory, wählen Sie den nächsten freien Platz an und lassen die Kassette im schnellen Vorlauf dorthin fahren.
3) Unterbrechen Sie hier das Programm, laden das neue Programm in den Rechner und SAVEn es auf der vorbereiteten Kassette ab. Jetzt befindet sich das Programm an der richtigen Stelle
4) Laden Sie abschließend noch einmal die Kassetten-Directory ein und geben in der entsprechenden DATA-Zeile den Programmnamen und eine Information ein (Speichern nicht vergessen).

E5.0 Sequentielle Datei

Die sequentielle Datei ist für die DATASETTE die einzige Form, Daten (Zahlen, Texte) extern abzulegen. Bedingt durch die Form des Bandes können Daten nur hintereinander gespeichert und in derselben Reihenfolge wieder in den Computer eingelesen werden. Das Prinzip und die Funktion einer sequentiellen Datei sind ausführlich in den Abschnitten D11.0 - D11.3, D11.5, D12.0 und D14.0 beschrieben. Bitte schlagen Sie dort nach, falls Ihnen die Zusammenhänge nicht mehr ganz klar sein sollten.

Um Daten vom Computer zu einem Peripheriegerät oder umgekehrt zu senden, muß ein Weg bestimmt und eröffnet werden. Der „Name" des Weges (laufende Nummer) kann eine Zahl zwischen 1 und 127 sein.

OPEN lfd Nr, Gerätenr.
OPEN 1, 1

Mit der Gerätenummer wird ausgewählt, welches externe Gerät angesprochen werden soll. Die Datasette ist fest auf die Gerätenummer 1 gelegt.
Zwei „Senderichtungen" sind mit der Datasette möglich: vom Computer zur Datasette und umgekehrt. Mit der Sekundäradresse wird bestimmt, ob der Computer von der Datasette lesen oder schreiben soll.

OPEN 1, 1, 0 öffnet Datei zum Lesen
OPEN 1, 1, 1 öffnet Datei zum Schreiben

Nach der Sekundäradresse (0=lesen, 1=schreiben) folgt der Name der zu bearbeitenden Datei.

OPEN 1, 1, 0, "dateiname"
OPEN 1, 1, 1, "dateiname"

Beschrieben werden die Dateien mit dem PRINT#- und gelesen mit dem INPUT#- oder GET#-Befehl. Die laufende Nummer (lfd Nr) ist jeweils die Nummer, mit der der Weg bezeichnet ist (hier 1).

E5.1 Eröffnen einer Datei (Schreiben)

```
100 open 1,1,1,"testdatei"
110 for i = 1 to 100
120 print#1, i
130 next i
140 close 1
```

Die Zahlen (i) 1 bis 100 werden in die „Testdatei" geschrieben. Nachdem alle Daten übermittelt worden sind, wird der Weg mit CLOSE wieder geschlossen.

Dieses ist besonders wichtig, da sich noch Daten im Kassettenpuffer befinden können, die noch nicht übermittelt wurden. Durch CLOSE wird der letzte Inhalt des Kassettenpuffers zur Datasette übermittelt (siehe D16.0).

E5.2 Lesen einer Datei

Auf dem umgekehrten Weg lassen sich Daten aus der Datei in den Computer lesen.

```
200 dim x(100)
210 open 1,1,0,"testdatei"
220 for i = 1 to 100
230 input#1, x(i)
240 next i
250 close 1
260 for i = 1 to 100
270 print x(i),
280 next i
```

Alle Daten sind in das Array x(..) eingelesen und werden abschließend (Zeile 260-280) noch ausgedruckt.

Das Ende einer Datei kann ebenso wie bei der Diskette mit Hilfe der Statusvariablen „ST" abgefragt werden (siehe D11.5).

Die Übertragungsgeschwindigkeit ist sehr gering, so daß auf größere Datenmengen schon eine ganze Zeit gewartet werden muß. Die Datenmenge sollte immer eine Größe besitzen, die ohne weiteres in den C=64 eingelesen werden kann (in ein Array). Bearbeiten Sie dann ihre Daten und schreiben Sie sie komplett wieder auf die Kassette. Müssen Sie mit größeren Datenmengen arbeiten, sollten Sie sich eine Diskettenstation schenken lassen (wäre doch schön, oder?).

EIN-/AUSGABE

F1.0 Die Tastatur

Die Tastatur ist das zentrale Instrument, um Daten (Programme, Eingaben) von außen in den Rechner einzugeben. Der C=64 besitzt insgesamt 66 Tasten. Zwei dieser Tasten sind „Sondertasten": die „RESTORE"- und die „SHIFT/LOCK"-Taste (siehe weiter unten).
Ohne die beiden „Sondertasten" sind es 64 Tasten, die in einer 8*8 Matrix (8*8=64) aufgebaut sind. Durch das Betätigen einer Taste werden jeweils 2 der 16 Leitungen miteinander verbunden. Hierdurch erkennt der Rechner, welche der 64 (66) Tasten betätigt worden ist. Mechanisch ist dieses sehr einfach gelöst worden:
beim Niederdrücken einer Taste wird ein Metallplättchen auf zwei Kontaktstellen gedrückt und die Verbindung ist hergestellt. Die Leitungen, mit Sonderleitungen 20 an der Zahl, sind auf einen Stecker geführt und auf die Platine gesteckt.

F1.1 Die Matrix

DEL	RETURN	=	CRSR1	F1	F3	F5	F7
3	W	A	4	Z	S	E	SHIFT1
5	R	D	6	C	F	T	X
7	Y	G	8	B	H	U	V
9	I	J	0	M	K	O	N
+	P	L	-	.	:	ü	,
ß	*	;	HOME	SHIFTr	=	ß	/
1	-	CTRL	2	SPACE	C=	Q	RUNSTOP

Die 8 Zeilen (waagerecht) und die 8 Spalten (senkrecht) sind deutlich zu erkennen. Die „2" befindet sich in Zeile 8 und Spalte 4. Das „A" in Zeile 2 und Spalte 3. Die 8 Zeilen-und Spaltenleitungen entsprechen je einem Byte.

F1.2 Tastaturabfrage

Über das Zeilen- und Spaltenregister (56320 und 56321) wird abgefragt, welche Taste gedrückt ist. Nacheinander werden die 8 Leitungen des Spaltenregisters auf 0 gesetzt. Ist eine Taste gedrückt, wird die entsprechende Leitung des Zeilenregisters ebenfalls auf „0" gesetzt. Diese Veränderung ist dann im Register 56321 festzustellen (das jeweilige Bit wird gelöscht).

F1.3 RESTORE und SHIFT/LOCK

Diese beiden Tasten sind nicht in der 8*8 Matrix enthalten.

Die SHIFT/LOCK-Taste ist nur ein Schalter, der parallel zu den SHIFT-Tasten liegt.
Die RESTORE-Taste legt ein IC im C=64 (U20) auf Masse und erzeugt ein „Rücksetzen des Betriebssystems" in den ursprünglichen Zustand (NMI).

F1.4 Tastaturpuffer

Bei Betätigung einer Taste wird im Tastaturpuffer (Adresse 631 - 640) das Zeichen zwischengespeichert. In der zugehörigen Adresse 198 steht die Anzahl der im Tastaturpuffer liegenden Zeichen. Der Tastaturpuffer läßt sich mit „Zeichen" laden, die dann auf dem Bildschirm ausgedruckt oder ausgeführt werden. Als Beispiel möchte ich das programmierte Laden innerhalb eines Programms erläutern.

```
10 rem****** nachladen per programm *******
20 input " * prog.-name :";pg$
30 print "load";chr$(34);pg$;chr$(34);",8"
40 poke 631,145:poke632,145:poke633,145
50 poke 634, 13:poke 198,4
60 rem**********************************
```

In der Zeile 20 wird nach dem Programm gefragt, welches geladen werden soll (pg$). Zeile 30 PRINTet auf den Bildschirm die Zeile

load "name",8

CHR$(34) ist der ASCII-Code für die Anführungsstriche und in pg$ steht der Programmname.
Zeile 40 lädt die ersten drei Speicherstellen des Tastaturpuffers mit dem ASCII-Code 145. Dies entspricht dem „Cursor up" und bewirkt, daß der Cursor sich drei Zeilen nach oben bewegt und damit direkt auf „load"name",8" steht. Abschließend wird ein CHR$(13), das entspricht dem RETURN, in den Tastaturpuffer gelegt und der Adresse 198 mitgeteilt, daß 4 Eingaben im Tastaturpuffer liegen. Das RETURN (13) bewirkt, daß die Zeile „load"name",8" direkt ausgeführt und damit das Programm geladen wird.

F2.0 Joystick

Der Joystick ist neben dem Paddle das sinnvollste Mittel, um Spiele (spez. Aktionspiele) einfach und rasant durchzuführen. Der Joystick ist bei fast jedem Spiel notwendig. Um die Arbeitsweise eines Joystick's und seine Programmierung zu verstehen, wird nachfolgend auf die Joysticks eingegangen.

Der C=64 besitzt 2 Joystick-Eingänge (JOY1 und JOY2). Die Anschlüsse befinden sich an der rechten Seite beim Ein- Ausschalter (CONTROL PORT 1, CONTROL PORT 2).

F2.1 Aktivieren der Control Ports

Da der C=64 die Abfrage der Tastatur und der Joysticks über einen Baustein abwickelt, kann nur das Keyboard oder die Joysticks abgefragt werden. Von daher muß dem Computer mitgeteilt werden, wenn er seine Informationen von den Joysticks erhalten soll. Mit

POKE 56322, 244

werden die Joysticks eingeschaltet und sofort ist eine Nutzung der Tastatur nicht mehr möglich. Um die Tastatur wieder einzuschalten, muß

POKE 56322, 255

eingegeben werden (durch RUN/STOP + RESTORE wird die Adresse 56322 ebenfalls auf 255 gesetzt). An Ende eines Programms mit Joystick sollte immer darauf geachtet werden, daß die Tastatur wieder aktiviert wird, da sonst keine Eingaben von der Tastatur möglich sind.

F2.2 JOY1 und JOY2

Die Informationen vom JOY1 stehen in dem Register 56321. Von hier können sie mit

J1 = PEEK (56321)

gelesen werden. Für den JOY2 steht ebenfalls ein Register zur Verfügung. Dieses hat die Adresse 56320 und kann mit

J2 = PEEK (56320)

entsprechend ausgelesen werden.

F2.3 Werte der Register 56320 und 56321

Alle weiteren Beschreibungen beziehen sich sowohl auf JOY1 als auch auf JOY2!!
Befindet sich der Joystick in Ruhestellung, d.h. es ist weder die Taste gedrückt noch wird der Stick bewegt, steht im Register 56320 (56321) der Wert 127.

01111111 = 127 keine Funktion

Welche Bits sind nun für den Joystick wichtig? Der Joystick ist im Prinzip ein Kasten mit 5 Schaltern. Einer für die Feuertaste und je einer für oben, unten,

links und rechts. Aus diesem Grunde werden auch nur die ersten 5 Bits der Register benötigt (Bit 0 bis 4). In Ruhestellung sind die 4 Bits gesetzt ('1'). Durch Betätigen des Sticks oder der Taste wird eines der 4 Bits gelöscht ('0').

BIT 0	x1111110	für OBEN
BIT 1	x1111101	für UNTEN
BIT 2	x1111011	für LINKS
BIT 3	x1110111	für RECHTS
BIT 4	x1101111	für FEUER

F2.4 Kombinationen

Ist der Stick in die Richtung OBEN/LINKS gedrückt, werden die Schalter für oben und links geschlossen und damit die beiden Bits auf Null gelegt.

x1111010= 122OBEN/LINKS

Alle anderen möglichen Kombinationen entstehen entsprechend:

x1110101= 117 UNTEN/RECHTS

x1100101= 101 UNTEN/RECHTS+FEUER

F2.5 Abfrage der Joysticks im BASIC-Programm

```
10 poke 56322, 224
20 j2 = peek (56320)
30 if (j2 and 1) = 0 then print "oben"
40 if (j2 and 2) = 0 then print "unten"
50 if (j2 and 4) = 0 then print "links"
60 if (j2 and 8) = 0 then print "rechts"
70 if (j2 and16) = 0 then print "feuer"
80 goto 20
```

Mit einer AND-Verknüpfung (siehe G1.2) läßt sich leicht ermitteln, in welcher Stellung sich der Joystick befindet.
Zwei Beispiele sollen noch einmal die Arbeitsweise erklären.

Prüfen nach der Richtung „oben":

```
     01111110   127    Stick nach oben.
AND  00000001     1    siehe Zeile 30
     --------
     00000000 =   0    Bedingung erfüllt
```

Prüfen nach der Richtung „unten/rechts"

```
     01110101   117    Stick nach „unten/rechts"
AND  00001010    10    prüfen
     --------
     00000000 =   0    Bedingung erfüllt
```

Mit der AND-Verknüpfung lassen sich alle Kombination prüfen, um in entsprechende Programmteile zu springen.
Der Zeichnung können Sie entnehmen, mit welchem Wert Sie „AND-Verknüpfen" müssen, um die entsprechende Stellung zu prüfen.

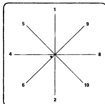


Abb. 28 Joystick

F3.0 Was ist ein Paddle

Neben dem Joystick ist der Paddle (Ruder) die älteste Steuereinheit für Spiele. Schon das legendäre PING-PONG Spiel der siebziger Jahre benutzte einen Paddle. Heute spielt der Paddle bei Spielen kaum noch eine Rolle und ist fast vollständig vom Joystick verdrängt worden.
Der Paddle besteht aus einem einfachen, regelbaren Widerstand (Poti), der zwischen einer Spannung von 0V und +5V liegt. Je nach Stellung des Poti's fällt eine unterschiedliche Spannung an ihm ab. Dieser Spannungsabfall (zwischen 0V und +5V) liegt dann an einem „Analog-Digital"-Wandler, der sich im Computer befindet. Ein Analog-Digital-Wandler erzeugt aus einem Spannungswert eine digitale Information zwischen 0 und 255 (8-Bit AD-Wandler). Ist die Spannung 0V, so wird ein Wert von „0" ausgegeben, bei +5V sind es „255". Außerdem befindet sich an dem Paddle noch eine Taste (Feuerknopf). Es besteht die Möglichkeit, zwei Paddlepaare anzuschließen, wobei jeweils nur ein Paar gleichzeitig betrieben werden kann. Die Ursache liegt darin, daß nur 2 AD-Wandler im C=64 zur Verfügung stehen. Die Paddlepaare werden ebenfalls, wie auch der Joystick, über die Control-Ports 1 oder 2 betrieben. Die beiden AD-Wandler liegen im SID-Chip in den Adressen 54297 und 54298

Paddle 1 (3) : Adresse 54297
Paddle 2 (4) : Adresse 54298

Je nach Stellung des Widerstandes (Drehregler) steht ein Wert zwischen 0 und 255 zur Verfügung. Einige Paddles nutzen den gesamten 8-Bit-Bereich nicht aus, so daß Sie bei Ihrem Paddle eventuell einen anderen Endwert erhalten. Die Ursache liegt im benutzten Widerstand (bester Bereich: 200 Ohm bis 200 Kiloohm).
Die Werte sind sehr leicht auszulesen:

```
100 print peek(54297), peek(54298)
110 goto 100
```

Je nach Stellung der Paddle wird der entsprechende Wert ausgegeben. Bei vielen Paddles ist der Arbeitsbereich nicht sehr günstig, so daß nur ein relativ kleiner Regelbereich eine Änderung des Wertes bewirkt.

F3.1 Abfrage der Tasten

Genau wie bei dem Joystick (siehe F2.3.) wird bei Betätigung der (Feuer-) Taste ein Bit im Register 56320 (Paddlepaar 2) bzw. 56321 (Paadlepaar 1) gelöscht ('0').

56321:	01111111	keine Funktion
56321:	01110111	Taste Paddle 1
56321:	01101111	Taste Paddle 2
56320:	01111111	keine Funktion
56320:	01110111	Taste Paddle 3
56320:	01101111	Taste Paddle 4

Ein kleines Spiel soll die Anwendung und Programmierung des Paddles verdeutlichen. Mit einem kleinen „Raumschiff" sollen Sterne abgeschossen werden. Das Raumschiff kann am oberen Bildschirmrand hin und her bewegt werden. Programmtechnisch taucht das Problem der Umrechnung zwischen Paddlewert (0-255) und Bildschirmposition auf. In der Zeile 130 findet diese Umrechnung statt, der Paddlewert liegt dann zwischen 20 und -20.

```
10 rem**************** spiel mit einem paddle ************
20 poke 53280,0: poke 53281,0
30 printchr$(147)
40 for i = 55336 to 55375
50 poke i, 1
60 next i
70 for i = 1 to 150
80 x=int(rnd(0)*920)+1104
90 poke x, 42: poke x+54272, 1
100 next i
110 gosub 500:ti$="000000"
120 s1=1083+(128-peek(54297))/6.3
130 s2=1083+(128-peek(54297))/6.3
140 if s2<1064 then s2=1064: goto 160
150 if s2>1104 then s2=1104
160 if (peek(56321) and 4) = 0 then 200
170 poke s1, 32: poke s2, 121: s1=s2
180 goto 130
200 rem******************** schuss **********************
```

```
210 for i=s2+40 to s2+920 step 40
220 if peek(i)=42 then poke1,32:zs=zs+1:gosub 500:goto 170
230 poke i, 93: poke i+54272,1: poke i, 32
240 next i:zf=zf+1:gosub 500
250 goto 170
500 rem***************** treffer anzeige *****************
510 print chr$(19);zs, zf
520 if zs=100 then 600
530 return
600 rem********************* spiel ende *****************
610 print chr$(147);
620 print " * spielende *"
630 print: print
640 print " * treffer :";zs
650 print " * vorbei  :";zf
660 print
670 print " * zeit    : ";ti$
680 print " *---------------------*"
690 print " * ergebnis:";
700 eg=20000-(val(ti$)*(zf+1)*2)
710 if eg<0 then eg=1
720 print eg
730 end
740 rem**************************************************
```

F3.2 Anwendung der AD-Wandler

Die AD-Wandler im C=64 sind sehr vielseitig nutzbar, um analoge Signale in digitale, dem Rechner verständliche Zustände, umzuwandeln. Eine Spannung (0V - +5V) kann in einem Bereich von ca. 20 mV (5/256) unterschieden werden. Bei einfachen Anwendungen, wie z.B einem Thermometer oder einer Lichtschranke, läßt sich mit einem minimalen Aufwand ein großer Nutzen erzielen. Jedem Hobbybastler bieten sich hier ungeahnte Möglichkeiten.

F4.0 Lightpen

Der Lightpen ist ein „Stift" in der Größe eines Kugelschreibers. Der Anschluß des Lightpen ('Lichtschreiber') befindet sich im CONTROL PORT 1. Mit dem

Lightpen läßt sich feststellen, an welcher Stelle des Bildschirms er aufgesetzt ist. Aus diesen Daten ist es dann per Programm möglich, Informationen vom Bildschirm „zu lesen".

F4.1 Funktion

Bei einem Fernseher oder einem Monitor wird das Bild dadurch erzeugt, daß ein oben-links startender „Punkt" Zeile für Zeile durchsaust und eine fluoreszierende Fläche aktiviert. Der Lightpen ist nun ein Widerstand, der sehr lichtempfindlich ist. Immer wenn dieser „Punkt" an die Stelle kommt, wo sich der Lightpen befindet, registriert der Lightpen dieses und gibt eine Information an den Computer weiter. Im Computer wird durch einen „Zeitvergleich" errechnet, an welcher Stelle der Lightpen steht. In den beiden Registern 53267 für die X-Koordinate und in 53268 für die Y-Koordinate stehen die entsprechenden Werte.

F4.2 Justieren

Der Lightpen arbeitet nur innerhalb des Bildschirms, nicht aber in dem „Border" (nicht genutzter Bildschirmrahmen). Die obere, linke Ecke bildet den Ausgangspunkt für die Berechnung der Lightpenstellung. Die Ausgangskoordinaten sind mit dem folgenden Programm leicht zu ermitteln:

```
10 PRINT PEEK(53267), PEEK(53268)
20 PRINT CHR$(147);: GOTO 10
```

Nun sind, wenn sich der Lightpen in der oberen, linken Ecke des Bildschirms (nicht im Border) befindet, die Anfangswerte für die X- und Y-Koordinate ablesbar. Diese beiden Werte sind die Ausgangswerte, die ständig benutzt werden, um die genaue Stellung des Lightpens zu ermitteln.

F4.3 Ein Zeichen auslesen

Um festzustellen, auf welches Zeichen der Lightpen zeigt, müssen die X- und Y-Koordinaten in eine 8*8 Matrix (Zeichengröße) umgerechnet werden. Dabei ist anzumerken, daß die X-Koordinate immer mit 2 multipliziert werden muß!

Die X- und Y-Koordinaten sind dann:

X = INT((PEEK(53267)-30)/4)
Y = INT((PEEK(53268)-50)/8)

Die Werte 30 und 50 sind die Ausgangswerte des Lightpens in der oberen, linken Ecke. Die errechneten Koordinaten entsprechen dann der Spalte und der Zeile. Als Sicherheit ist eine Bereichskontrolle sinnvoll.

IF X < 0 OR X > 39 THEN ...... Fehler
IF Y < 0 OR Y > 24 THEN ...... Fehler

Um einen Lightpen praktisch anzuwenden, müssen Sie immer wissen, welche Daten an welcher Stelle des Bildschirms stehen. Für Zahlen bietet sich ein zweidimensionales Array an, das zeilen- und spaltenorientiert ist. Mit den Zeilen- und Spaltenwerten lassen sich dann die entsprechenden Arrays „auslesen".

F4.4 Übergabe von Lightpendaten

Immer wenn am Lightpen eine ausreichende „Lichtinformation" ansteht, werden Werte an den Computer weitergegeben. Um Daten vom Lightpen nur dann zu akzeptieren, wenn sie auch sinnvoll sind (z.B. Suchen der richtigen „Stelle" auf dem Bildschirm), kann mit einer zusätzlichen Information bestimmt werden, daß die Stellung richtig ist. Dieses ist zum Beispiel mit der RETURN-Taste möglich.

```
1000 X = INT((PEEK(53267)-30)/4)
1010 Y = INT((PEEK(53268)-50)/8)
1020 GET A$: IF A$ = CHR$(13) THEN ..... weiter
1030 GOTO 1000
```

Viele Lightpentypen bieten eine komfortablere Möglickkeit. Am Lightpen selber befindet sich ein kleiner „Schalter", mit dem eine Information an den Computer weitergegeben wird. Oft ist dieser „Schalter" auch ein Sensortaster

(touch contact). Beim bekannten „STACK Lightpen" wird mit dem touch contact z.B. das Bit 3 der Adresse 653 auf „1" (high) gelegt. Mit einer einfachen Abfrage kann entschieden werden, ob die Lightpen Information korrekt ist.

```
IF PEEK(653) AND 4 = 0 THEN ..... weiter
```

F4.5 Lightpen in der hochauflösenden Grafik

Auch in der hochauflösenden Grafik ist eine Anwendung möglich. Beim Malen z.B. werden die Punkte gesetzt (X,Y),auf die der Lightpen gerade zeigt. Es bieten sich sicherlich viele Anwendungsmöglichkeiten in der hochauflösenden Grafik. Wie sinnvoll und nützlich sie sind, muß jeder Anwender für sich entscheiden.

Mit einem Lightpen ist ein bildschirmorientiertes Arbeiten nur begrenzt möglich. Der wesentliche Nachteil ist, daß der Lightpen nur dann arbeitet, wenn eine ausreichende Lichtinformation zu ihm gelangt. Bei vielen Bildschirmfarben arbeitet der Lightpen nur unzureichend oder gar nicht (z.B. bei Schwarz). Trotzdem sind einige Möglichkeiten der Lightpen-Anwendung denkbar, die gerade das Arbeiten mit großen Datenmengen (Zahlen,Tabellen ..) erleichtern. Auch die „MAUS" läßt sich mit einem Lightpen gut ersetzen (simulieren). Hier liegt wohl der beste Anwendungsbereich eines Lightpen.

F5.0 IEC - Bus

Zum Anschluß von Peripheriegeräten (Drucker, Floppy) befindet sich auf der Rückseite des C=64 ein 6-poliger Diodenanschluß. Über diesen Ausgang werden Daten in serieller Form zu einem anderen Gerät gesendet. Seriell bedeutet, daß ein Byte nicht gleichzeitig (8 Bit auf einmal), sondern jedes Bit einzeln - hintereinander - übertragen wird. Der Nachteil eines seriellen Ausgangs gebenüber einem parallelen Ausgang ist die Übertragungsgeschwindigkeit. Im Gegensatz zur parallelen Übertragung, die 8 Datenleitungen benötigt (pro Bit eine Leitung), wird für die serielle Datenübertragung nur eine Leitung benötigt.

An diesen Anschluß kann nicht nur ein Gerät angeschlossen werden. Werden mehrere Peripheriegeräte angeschlossen, so werden sie von einem Gerät zum anderen „geschleift". Es ist Ihnen sicherlich aufgefallen, daß die Diskettenstation VC-1541 zwei „Eingangbuchsen" besitzt (sie liegen beide parallel). Schließen Sie z.B. eine weitere Diskettenstation an, schleifen Sie die zweite Diskettenstation ein. Also von der zweiten Buchse der ersten Diskette in eine Buchse der zweiten Diskette. Ein weiteres Gerät wird dann ebenso hinter dem zweiten angeschlossen. So lassen sich insgesamt 12 verschiedene Geräte an den IEC-Bus anschließen.

F5.1 Gerätenummer

Beim Anschluß mehrerer Geräte taucht sofort die Frage auf, woher wissen die Peripheriegeräte, für welches von ihnen die Daten bestimmt sind? Jedes Gerät besitzt eine „Gerätenummer", die Datasette z.B. die Gerätenummer 1 und die Diskettenstation die Nummer 8. Jedes andere Gerät, das ebenfalls am IEC-Bus angeschlossen ist, muß eine Gerätenummer benutzen, die sich von den anderen unterscheidet. Es stehen die Nummer 4 bis 15 zur Verfügung. Wird der IEC-Bus angesprochen, z.B. mit

OPEN 1,8, ...

wird ein „Achtung"-Signal (ATN) an die angeschlossenen Geräte gesendet. Alle Geräte sind aufnahmefähig für die vom Computer gesendete Gerätenummer (in diesem Fall 8). Das Gerät, das diese Nummer trägt, wird „aktiviert", alle anderen „gesperrt". Ist kein Gerät mit dieser Gerätenummer angeschlossen, wird die Fehlermeldung „DIVICE NOT PRESENT" ausgegeben. Die nächste Information, die dem Peripheriegerät mitgeteilt wird, ist die Richtung, von wo nach wo die Daten geschickt werden sollen. Vom Gerät zum Computer oder vom Computer zum Gerät. Endlich können die Daten auf die Reise geschickt werden. Bit für Bit. Damit es zu keinen Störungen im Datenfluß kommt, werden nur dann Daten übermittelt, wenn das empfangende Gerät auch dazu bereit ist (handshake). Ist die Übertragung beendet, sendet der Computer ein Signal und gibt alle angeschlossenen Geräte wieder „frei".

F5.2 Der Drucker

An dem IEC-Bus können normalerweise nur zwei Arten von Geräten angeschlossen werden. Diskettenlaufwerk(e) und die COMMODORE-eigenen

Drucker. Andere Drucker oder Diskettenlaufwerke können schon auf Grund dieses IEC-Busses nicht an den C=64 angeschlossen werden.
Die COMMODORE-eigenen Drucker besitzen einen großen Vorteil: alle Zeichen (auch die Grafikzeichen) können gedruckt werden. Besitzen Sie aber schon einen Drucker (nicht von COMMODORE), können Sie diesen nicht an den IEC-Bus anschließen. Außer den COMMODORE-Druckern gibt es fast keine mit einem eingebauten IEC-Bus! Dagegen haben fast alle Drucker eine Centronics-Schnittstelle.

F5.3 CENTRONICS-Schnittstelle

Die Firma Centronics führte bei ihren Produkten diese Schnittstelle ein, die man heute als (inoffizielle) Norm bezeichnen kann. Es handelt sich hierbei um eine Parallel-Schnittstelle. Neben den Datenleitungen existieren zusätzlich die STROBE- (neue Daten vorhanden) und die BUSY-Leitung (bin beschäftigt). Um einen solchen Drucker mit dem C=64 zu betreiben, müssen Sie entweder ein Hardware-Interface kaufen (150 - 300 DM) oder aber ein entsprechendes Programm haben, das die Centronics-Schnittstelle über den USER-Port steuert. Benötigt wird neben einem solchen Programm noch eine Verbindung zwischen dem USER-Port und der Centronics-Schnittstelle. Dieses Kabel (Flachband o.ä.) muß auf der einen Seite einen USER-Port-Stecker und auf der anderen Seite einen Centronics-Stecker besitzen (Elektronikfachladen). Der Preis der Stecker und des Kabels liegt bei knapp 30.- DM. Das Beschalten ist sehr einfach und kann von jedem durchgeführt werden. Hier die Kabelverbindung:

USER-PORT		CENTRONICS
A	MASSE	16
B	AKN.	10
C	D0	2
D	D1	3
E	D2	4
F	D3	5
H	D4	6
J	D5	7
K	D6	8
L	D7	9
M	STROBE	1

Die Belegung des USER-Port-Steckers finden Sie unter F6.0.
Und nun noch ein kleines Programm zum Betreiben des USER-Ports. Es ist leider in Maschinensprache und hier als BASIC-Loader abgedruckt.

Nach dem Starten des Programms wird die Gerätenummer 4 (Standard für Drucker) auf den USER-Port umgeleitet. Aktiviert wird das Programm mit SYS 49197. Natürlich ist es nicht möglich, die COMMODORE Grafikzeichen auszudrucken. Nach einem RUN/STOP-RESTORE müssen Sie das Maschinenprogramm mit SYS 49197 wieder neu aktivieren.

```
10 for i = 49152 to 49152+86
20 read x: poke i,x
30 next
40 sys 49197
50 rem*****************************
60 rem******* ohne linefeed ********
70 rem***     poke 49170, 234     ***
80 rem***     poke 49171, 234     ***
90 rem*****************************
100 data 072,165,154,201,004,240,003
110 data 076,205,241,104,201,013,208
120 data 005,032,020,192,169,010,141
130 data 001,221,173,000,221,041,251
140 data 141,000,221,009,004,141,000
150 data 221,173,013,221,041,016,240
160 data 249,024,096,120,169,255,141
170 data 003,221,173,002,221,009,004
180 data 141,002,221,173,000,221,009
190 data 004,141,000,221,169,016,141
200 data 013,221,173,013,221,169,000
210 data 141,038,003,169,192,141,039
220 data 003,088,096
```

F6.0 USER-Port

User-Port, wie der Name schon sagt, ist ein „Benutzer-Port", der frei programmierbar ist. Immer wenn Daten zwischen einem Gerät, das keine IEC-Schnittstelle besitzt und dem C=64 ausgetauscht werden sollen, wird der

USER-Port benutzt. Sei es, um Daten von einem Computer zu einem andern zu übertragen, um Meßergebnisse von einer Meßeinheit aufzunehmen, um einen Roboter zu steuern, einen Drucker ohne IEC-Schnittstelle zu betreiben oder um ...., (fast) immer wird sich des USER-Ports bedient. Und das nicht ohne Grund. Die Programmierung des USER-Ports ist auch in BASIC relativ einfach möglich.

F6.1 USER-Port und CIA

Der USER-Port befindet sich an der (linken) Rückseite des C=64 und besitzt 24 Kontakte. Auf der oberen Seite des USER-Ports befinden sich vor allem Leitungen von CIA 1 und Betriebsspannungen (GND, +5V, 9V Wechselspannung). Die Oberseite ist von 1 bis 12 bezeichnet, die Unterseite dagegen von A bis N und sie ist für die Benutzung des USER-PORTs besonders wichtig.

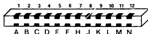


Abb. 29 USER-Port

01 : GND	A : GND
02 : +5V	B : FLAG2
03 : RESET	C : DATA0
04 : CNT1 (CIA 1)	D : DATA1
05 : SP1 (CIA 1)	E : DATA2
06 : CNT2 (CIA 2)	F : DATA3
07 : SP2 (CIA 2)	H : DATA4
08 : PC2 (CIA 2)	J : DATA5
09 : ATN OUT (seriell)	K : DATA6
10 : 9V AC	L : DATA7
11 : 9V AC	M : PA2
12 : GND	N : GND

Die Spannungen (+5V und 9V AC) können als Spannungsversorgung benutzt werden. Bitte achten Sie darauf, daß Sie diese Spannungen nicht höher als 100

mA belasten. Fließt ein größerer Strom (100 mA), werden sich die Sicherungen im C=64 und/oder im Netzteil verabschieden! Also: Achtung! Verwenden Sie eine separate Spannungsquelle, vergessen Sie dabei nicht die Masse (GND) mit GND des C=64 zu verbinden.
Das Entscheidende an dem USER-Port sind die Kontakte C bis L (DATA0 bis DATA7). Diese 8 Bits (1 Byte) bilden das Tor zur Außenwelt. Entweder können Daten gesendet (Computer-Außenwelt) oder empfangen (Außenwelt-Computer) werden.

F6.2 Datenrichtungsregister

Die Leitungen DATA0 bis DATA7 werden vom CIA2-Chip bedient. Der CIA2 befindet sich in den Adressen 56576 bis 56832. Damit der USER-Port betrieben werden kann, muß im Datenrichtungsregister (56579) bestimmt werden, ob die Datenleitungen als Ausgang oder als Eingang betrieben werden sollen. Jedes Bit des Datenrichtungsregisters repräsentiert eine Datenleitung.

Bit0	DATA0
Bit1	DATA1
......	....
Bit7	DATA7

Ist ein Bit gesetzt ('1'), so arbeitet die entsprechende Datenleitung als Ausgang, bei gelöschtem Bit ('0') als Eingang. Um alle Datenleitungen als Ausgänge zu schalten, wird das Datenrichtungsregister mit 255 geladen.

```
POKE 56579, 255     1111 1111
```

Im Einschaltzustand sind alle Leitungen als Eingang geschaltet (0000 0000). Weiterhin besteht die Möglichkeit, bestimmte Datenleitungen als Ein- und andere als Ausgang zu benutzen.

```
POKE 56579, 240     1111 0000
```

Als Beispiel wurden die Datenleitungen 0 - 3 als Eingang und die Datenleitungen 4 - 7 als Ausgang geschaltet. Bitte achten Sie UMBEDINGT darauf, daß Sie Datenleitungen, die als Ausgang geschaltet worden sind, nicht mit einer

Spannung belegen (Eingang). Es führt zur Zerstörung des CIA2 (ca. 50.-DM)!! Einen Ausgang NUR als Ausgang (Computer-Außenwelt) und einen Eingang NUR als Eingang (Außenwelt-Computer) benutzen.

F6.3 Datenregister

Das Datenregister (56577) ist direkt mit den Datenleitungen DATA0 - DATA7 verbunden. Nachdem die Datenleitungen als Ausgang geschaltet worden sind,

```
POKE 56579, 255        1111 1111
```

können die Datenleitungen gesetzt oder gelöscht werden. Jedes Bit des Datenregisters (56577) „versorgt" eine Datenleitung. Ist das Bit gesetzt, so wird die entsprechende Datenleitung „1" (+5V), bei gelöschtem Bit „0" (0V).

```
POKE 56577, 255        1111 1111
```

An allen Datenleitungen liegt eine Spannung von +5V ('1') an. (Haben Sie ein Meßgerät zur Hand, messen Sie ruhig einmal - gegen GND (Pin 1) -). Beachten Sie bitte, daß die auf Ausgang geschalteten Datenleitungen nur sehr gering belastet werden dürfen (TTL). Es ist nicht ratsam, eine kleine Lampe oder eine Leuchtdiode direkt an die Datenleitugen anzuschließen. Der Rechner wird es Ihnen danken und funktionsfähig bleiben. Verwenden Sie immer einen Treiber (siehe Schaltplan). Es eignen sich die Bausteine SN 7406, SN74LS06, CD4009, die im Elektronikfachhandel erhältlich sind.
Ist das Datenrichtungsregister auf Eingang geschaltet und liegen an den Datenleitungen Signale an ('0' oder '1'), so kann das Datenregister mit

```
PRINT PEEK(56577)
```

ausgelesen werden und die Informationen lassen sich entsprechend verarbeiten.

F6.4 Lauflicht

Um die Anwendungsmöglichkeiten des USER-Ports zu demonstrieren, stelle ich Ihnen ein Lauflichtprogramm vor. Als „Lampen" dienen 8 Leuchtdioden,

die über einen Treiber geschaltet sind. Wird eine Datenleitung „1", so leuchtet die Leuchtdiode auf. Mit einer entsprechenden Treiberstufe lassen sich statt der Leuchtdioden auch Strahler etc. betreiben. Hier nun die kleine Schaltung (ca. 10.- DM).

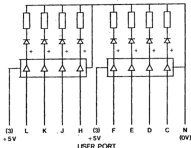


Abb. 30 Schaltung

Das Programm dient nur der Demonstration und kann auch ohne Schaltung betrieben werden. Die Leuchtdioden werden auf dem Bildschirm simuliert, so daß die Arbeitsweise deutlich wird. Das Menü bietet Ihnen auch die Möglichkeit, ein Lichtprogramm (bis 1000 Schritte) zu erstellen und „abzuspielen".

```
100 rem*********************************************
110 rem*********** simulation eines lauflichts ***********
120 rem*********************************************
130 :
140 rem************* user-port vorbereiten ***************
150 poke 56579, 255: rem datenrichtung-alle auf ausgang !!
160 rem*********************************************
```

```
170 :
180 dim de(1000):bl$=chr$(145)+"
190 poke 53280,0
200 print chr$(147)
210 print " --- steuern ueber user-port ---"
220 print: print
230 print " *  (1)  lauflicht ..........."
240 print " *  (2)  programm erstellen ..."
250 print " *  (3)  programm starten ....."
260 print " *  (4)  zufallsprogramm ......"
270 print " *  (5)  ende ................"
280 print
290 input " *        bitte waehlen sie ....";a$
300 a=val(a$):if a<1 or a>5 then print chr$(145);: goto 290
310 on a gosub 350, 430, 590, 730, 810
320 goto 200
330 rem*********************************************
340 :
350 rem***************** lauflicht *********************
360 z=0:gosub 860: print
370 for j = 0 to 7:
380 de(z)=2ßj:gosub980
390 get b$:if b$ <> "" then de (z) = 0: gosub 1070: return
400 nextj:goto 370
410 rem*********************************************
420 :
430 rem************** programm erstellen ****************
440 z=1:gosub 860
450 x=5:y=20:gosub 920
460 input " wert (0-255) :";de(z)
470 if de(z)<0orde(z)>255 then sp=z-1:z=0:gosub 1070:return
480 gosub 980
490 x=5: y=21:gosub920
500 print "wert o.k. ? (j/n)"
510 get b$: if b$= "" then 510
520 if b$ = "j" then print bl$:z=z+1:goto 450
530 if b$ <> "n" then 510
540 print bl$
550 de(z)=0:gosub 980
```

```
560 goto 450
570 rem*************************************************
580 :
590 rem************** programm abspielen ****************
600 gosub 860
610 if sp >0 then 660
620 x=5:y=20:gosub 920
630 print "kein  programm vorhanden !!!!!!!"
640 for t = 1 to 2500:next
650 return
660 for z = 1 to sp
670 gosub 980
680 get b$: if b$ <> "" then z=0:gosub 1070: return
690 next
700 goto660
710 rem*************************************************
720 :
730 rem*********** zufaellige kombinationen ************
740 z=0:gosub 860
750 de(z)=int(rnd(1)*255)+1
760 gosub 980
770 get b$:if b$ <> "" then de(z)=0: gosub 1070: return
780 goto 750
790 rem*************************************************
800 :
810 rem************** programm ende*********************
820 print chr$(147)
830 poke 56579, 0  : rem normalzustand alle auf eingang !!
840 end
850 rem*************************************************
860 rem************** bildschirm aufbau *****************
870 for i = 0 to 39: print chr$(99);:next i:print
880 print tab(6);" lampen";tab(20);"wert     step
890 return
900 rem*************************************************
910 :
920 rem**************** x - y - cursor ******************
930 poke 211,x: poke 214,y: sys 58640
940 return
```

```
950 rem**************************************************
960 :
970 rem************ umrechnung und anzeige ***************
980 di=de(z):bi$="":n$=chr$(46):r$=chr$(113)
990 di=di/2:d$=n$:if di <> int(di) then d$=r$
1000 di=int(di):bi$=d$+bi$: if di >0 then 990
1010 if len (bi$)<8 then bi$=n$+bi$:goto1010
1020 x=7:y=15:gosub 920:print bi$
1030 x=20:gosub 920: print"    "
1040 gosub 920: print tab(30);"    "
1050 gosub 920: print de(z);tab(30);z
1060 rem************ ausgabe auf user-port ****************
1070 poke 56577, de(z)
1080 return
1090 rem**************************************************
```

ANHANG

G1.0 Dezimalsystem

Der Mensch benutzt üblicherweise das ihm bekannte Dezimal- oder Zehnersystem. Alle denkbaren Zahlen lassen sich mit 10 Symbolen, den Zeichen „0" bis „9", darstellen. Ist eine Zahl größer als 9 darzustellen, findet ein Übertrag in die nächsthöhere Potenz statt. Die Dezimalzahl 2345 läßt sich auch anders ausdrücken:

2345 = 2*1000 + 3*100 + 4*10 + 5*1

Mathematisch läßt sich das auch so ausdrücken:

2345 = 2*10^3 + 3*10^2 + 4*10^1 + 5*10^0

Zahlen mit einer Null potenziert, ergeben immer 1 (z.B. 10^0=1,3424^=1 etc.). Eine Ausnahme bildet hierbei der Wert 0^0, denn er ist mathematisch nicht definiert. Somit ist jede endliche Zahl zu bestimmen.

Der Rechner ist nicht in der Lage, „bis Zehn zu zählen", denn er müßte sich dann 10 verschiedene Zustände merken können (von 0 bis 9). Er ist nur in der Lage, zwei Zustände zu unterscheiden: Strom fließt, Strom fließt nicht oder besser, ein Speicher ist geladen ('1') oder eben nicht ('0').

G1.1 Binärsystem

Der Computer besitzt eine bestimmte Anzahl von Speicherstellen. Jede dieser Speicherstellen kann entweder geladen oder nicht geladen sein. Eine geladene Speicherstelle wird allgemein mit „1" und eine nicht geladene mit „0" bezeichnet. Diese Speicherstelle heißt „Bit" (eng.: binary digit). 8 Bits sind zu einem „Byte" (Kunstwort) zusammengefaßt und bilden für die Speicherung eine „feste" Einheit.

8 Bit = 1 Byte

Eine große Anzahl von Bytes wird in „kByte" (nicht Kilo-Byte sondern „kByte") angegeben. 1 kByte entspricht 1024 Byte.

1 kByte = 1024 Byte = 8192 Bit

Der C=64 besitzt Speicherraum für 64 kByte, das sind:

64 kByte = 64*1024 = 65536 Bytes
oder 65536*8 = 524288 Bits

Das ist schon eine Menge Speicherraum, doch um Werte größer als 1 darzustellen, benötig er mehr als ein Bit, denn ein Bit unterscheidet nur den Zustand „0" und „1". Sehen Sie sich einmal diese beiden Bits an:

00 = 0
01 = 1
10 = 2
11 = 3

Die beiden Bits sind jeweils nur an- oder ausgeschaltet. Durch die Kombination zweier Bits ist es also möglich, 4 verschiedene Werte darzustellen. Durch das Kombinieren vieler Bits lassen sich wieder endliche Zahlen bilden.

10101 = 21

Der Übertrag in die nächsthöhere Potenz findet hier immer von 1 auf 2 statt und die Basis ist 2.

10101 = 1*16 + 0*8 + 1*4 + 0*2+ 1*1 = 21
10101 = 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0

Mit einem Byte (8 Bit) sind insgesamt 256 verschiedene Kombinationen möglich und Sie können so die Zahlen 0 - 255 darstellen.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^1		
128	064	032	016	008	004	002	001		
0	0	0	0	0	0	0	0	=	0
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	1	1	=	3
0	0	0	0	0	1	0	0	=	4
0	0	0	0	0	1	0	1	=	5
0	0	0	0	0	1	1	0	=	6
0	0	0	0	0	1	1	1	=	7
0	0	0	0	1	0	0	0	=	8
1	1	1	1	1	1	0	0	=	252
1	1	1	1	1	1	0	1	=	253
1	1	1	1	1	1	1	0	=	254
1	1	1	1	1	1	1	1	=	255

Die Umwandlung von Dezimal- in Binärwerte ist durch eine ständige Teilung durch 2 zu erzielen.

Umrechnung der Dezimalzahl 200 in Bi[illegible]:

200 / 2	=	100	Rest	0
100 / 2	=	50	Rest	0
50 / 2	=	25	Rest	0
25 / 2	=	12	Rest	1
12 / 2	=	6	Rest	0
6 / 2	=	3	Rest	0
3 / 2	=	1	Rest	1
1 / 2	=	0	Rest	1

Die Binärzahl entsteht durch Lesen des Restes von unten nach oben. 200 = 11001000.

Der gesamte Speicherbereich des C=64 ist generell in Bytes (8 Bit) organisiert. Wird ein Speicher ausgelesen (PEEKen), erscheint immer die Dezimalzahl des Inhalts eines Bytes. Beim Beschreiben (POKEn) eines Speicherplatzes

wird immer ein Byte gesetzt. Durch die 8-Bit-Struktur einer Speicherstelle kann nur ein ganzzahliger Wert zwischen 0 und 255 gePOKEt werden. Oftmals ist es allerdings notwendig, ein bestimmtes Bit eines Bytes zu verändern und alle anderen Bits unverändert zu lassen.

G1.2 Logische Arithmetik

Die beiden BASIC-Operatoren „AND" (UND) und „OR" (ODER) sind Ihnen sicherlich von BASIC-Programmen her bekannt (Bereichsabfragen z.B. IF A=0 OR A=1 THEN ...). Der AND- bzw. OR-Befehl ist ein logischer Verknüpfungsbefehl, mit dem auch zwei Zahlen binär miteinander verknüpft werden können.

1. AND: die AND-Funktion verknüpft Binärzahlen nach folgendem Prinzip:

```
    10
AND 11
------
    10
```

Die beiden Zahlen „10" (Dez. =2) und „11" (Dez. =3) wurden „AND-verknüpft". Die wertgleichen Stellen der beiden Zahlen werden verknüpft. Immer dann, wenn beide Stellen den Wert „1" tragen, ist das Ergebnis ebenfalls „1" („1" UND „1" = „1"). Ist eine der Stellen „0", ist das Ergebnis auch Null.

```
'0' AND '0' = '0'
'0' AND '1' = '0'
'1' AND '0' = '0'
'1' AND '1' = '1'
```

Auch größere Zahlen (8 Bit - 1 Byte), lassen sich so verknüpfen. Hierzu ein Beispiel:

```
    10010111     = Dez. 151
AND 01011110     = Dez.  94
------------
    00010110     = Dez.  22
```

In BASIC ist dies der Eingabe

PRINT 151 AND 94

entsprechend. Probieren Sie ein bißchen die AND-Verknüpfung aus, damit Sie etwas „Gefühl" für Verknüpfungen erhalten.

2. OR: die OR-Verknüpfung ist etwas einfacher zu verstehen. Immer dann, wenn eine der beiden Stellen den Wert „1" besitzt, ist das Ergebnis ebenfalls „1".

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

Größere Zahlen verknüpfen sich entsprechend.

```
    10010111    = Dez. 151
OR  01011110    = Dez.  94
- - - - - - - - - - - -
        1       = Dez. 223
```

Im BASIC sieht es dann so aus:

PRINT 151 OR 94

Mit dem AND- bzw. OR-Befehl ist es möglich, einzelne Bits eines Bytes zu manipulieren. Das Setzen eines bestimmten Bits ist recht einfach. Als Beispiel soll das Bit 3 eines Bytes gesetzt werden. Die Bits eines Bytes sind von rechts nach links von 0 bis 7 durchnummeriert.

```
Bit 76543210
-----------
    10010111
OR  00001000
-----------
    10011111
```

Das Bit 3 ist jetzt gesetzt und die restlichen Bits blieben unverändert, da sie mit „0" OR-Verknüpft werden. In BASIC wird mit der Wertigkeit des zu setzenden Bits „OR-Verknüpft".

PRINT 151 OR 8 oder PRINT 151 OR 2∧3

Allgemein ist das Setzen eines Bits mit der Eingabe

Wert OR 2∧Bit

möglich.

Das Setzen wird vor allem dazu benutzt, um in Speicherstellen irgendetwas einzuschalten. Das bedeutet, daß der geänderte Inhalt sofort wieder in die Speicherstelle zurückgeschrieben wird. Somit ist das Setzen eines Bits in einer Speicherstelle mit der Zeile

POKE Adresse, PEEK(Adresse) OR 2∧Bit

zu erreichen. Anzumerken ist noch, daß ein bereits gesetztes Bit gesetzt bleibt.

Das Löschen eines Bits ist etwas verwirrender. Mit einer OR-Verknüpfung ist dieses nicht möglich, denn sowie ein Bit „1" ist, wird das Ergebnis ebenfalls „1". Nur die AND-Verknüpfung hilft hier weiter. Das zu löschende Bit muß auf „0" gesetzt und alle anderen Bits auf „1" gelegt werden.

```
    10001011
AND 11110111
------------
    10000011
```

Das Bit 3 ist gelöscht. Überall dort, wo mit einer „1" AND-verknüpft wird, bleibt der ursprüngliche Wert erhalten. So ergibt 150 AND 255 wieder 150! Der Wert des zu löschenden Bits ist also von 255 abzuziehen.

255 - 2∧Bit

Ein Bit einer Speicherstelle ist mit

POKE Adresse, PEEK(Adresse) AND 255-2∧Bit

zu löschen. Das Setzen und Löschen von bestimmten Bits ist speziell in der Grafik ein alltägliches Problem. Sie werden immer wieder vor der Aufgabe stehen, ein bestimmtes Bit eines Bytes zu setzen oder zu löschen.

G1.3 Zerlegen in High- und Lowbyte

In einem Byte (8 Bit) lassen sich nur Werte bis 255 darstellen. Größere Werte müssen in zwei Bytes zerlegt werden. Damit der Computer alle seine 65536 Speicherstellen ansprechen und bestimmen kann (z.B. BASIC-Anfang, BASIC-Ende...), muß er zumindestens Zahlen bis 65535 darstellen können. Alle Stellen eines Bytes sind bei dem Wert 255 mit „1" gefüllt. Durch hinzuaddieren einer „1" müßte sich der Wert 256 ergeben. Es wäre allerdings ein Übertrag in die nächsthöhere Potenz nötig, doch die ist nicht vorhanden.

```
    11111111     255
 +  00000001       1
 -----------------
  1 00000000     256
```

Der Übertrag ist nicht mehr in dem Byte darstellbar. Das Byte selber hat dann den Inhalt Null! Nur ein zweites Byte kann die Aufgabe übernehmen, diesen Übertrag zu speichern. Dieses Byte wird als „Highbyte" (hochwertiges Byte) bezeichnet, der niederwertige Teil „Lowbyte". Das Bit 0 des Highbytes entspricht nun nicht mehr dem Wert Dez. 1, sondern Dez. 256.

00000001 00000000 = 256

Daraus ist zu folgern, daß der Inhalt des Highbytes immer mit dem Faktor 256 zu multiplizieren ist.

Highbyte = 1, Lowbyte =0 = 1*256 + 0 = 256

Insgesamt lassen sich mit High- und Lowbyte

255*256 + 255 = 65535

Werte unterscheiden und darstellen.
Der Mikroprozessor ist unter bestimmten Bedingungen in der Lage, 2-Byte-Zahlen zu verarbeiten und zwar immer dann, wenn er auf eine seiner 65536 Speicherstellen zugreifen will. Die beiden Bytes liegen immer direkt hintereinander in der Reihenfolge Lowbyte, Highbyte. Mathematisch ist die Umrechnung recht einfach:

```
Highbyte = INT(Wert/256)
Lowbyte  = Wert - Highbyte*256
```

Als Beispiel soll der BASIC-Speicherbereich der Adresse 32768 geschützt werden (siehe A4.5). Zunächst die Umrechnung in Low- und Highbyte:

```
HB=INT(32768/256)    = 128
LB=32768 - HB*256   =   0
```

Dann werden die entsprechenden Adressen mit High- und Lowbyte geladen.

```
POKE 55, LB: POKE 56, HB
POKE 51, LB: POKE 52, HB
```

Umgekehrt ist der Dezimalwert mit

```
Dezimal = Highbyte*256 + Lowbyte
  32768 = 128*156 + 0
```

zu bestimmen.

G1.4 Hexadezimalsystem

Noch vor Jahren war das Programmieren von Rechnern sehr umständlich: Speicherstellen mußten binär, also mit unendlich vielen Nullen und Einsen, eingegeben werden. Als erstes Hilfsmittel für die Programmierung wurde das „Hexadezimalsystem" verwendet. Der große Vorteil lag (liegt) darin, daß jede Zahl von 0 bis 255 mit nur zwei Zeichen dargestellt werden kann. Heute ist die hexadezimale Darstellung zwar nicht mehr notwendig, hat sich aber der-

art „eingebürgert", daß sie immer noch verbreitet ist. Für Speicherauszüge wird die hexadezimale Darstellung auch heute noch bevorzugt, da alle Werte leichter zu überblicken sind (HEX-DUMP).
Das hexadezimale System beruht auf der Zahlendarstellung zur Basis 16. Somit findet der Übertrag in eine höhere Potenz erst bei 16 statt. Es müssen also 16 Symbole für die Darstellung vorhanden sein. Für den Wert 0 bis 9 können die normalen Ziffern benutzt werden. Die 10 ist allerdings nicht mehr mit einer Ziffer darstellbar, so daß hierfür ein Buchstabe einspringen muß.

```
Dez.  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
Hex  $0 $1 $2 $3 $4 $5 $6 $7 $8 $9 $A $B $C $D $E $F
```

Die hexadezimalen Zahlen werden zur Unterscheidung üblicherweise mit dem vorgestellten „$"-Zeichen gekennzeichnet. Vielfach wird auch ein führendes Hochkomma (') oder ein vor- oder nachgestelltes „H" zur Kennzeichnung einer hexadezimalen Zahl benutzt.

```
$7 =  7 Dez
$A = 10 Dez
$F = 15 Dez
```

Der Übertrag findet bei 16 statt, somit ist

```
$10 = 1*16 + 0 = 16 Dez
```

Mit einer 2-stelligen Hex-Zahl lassen sich

```
$FF = 15*16 + 15 = 255
```

255 dezimale Zahlen darstellen. Die Umrechnung einer beliebig großen Hex-Zahl (z.B. $A3FC) ist mit folgender Gleichung möglich

```
   A        3          F          C
10*16^3 + 3*16^2 + 15*16^1 + 12*16^0
```

In diesem Buch ist das hexadezimale Zahlensystem nur im Abschnitt zur Diskettenstation benutzt worden. Der Grund dafür liegt ausschließlich in der besseren Darstellungsmöglichkeit von Speicherinhalten.

G2.0 Sprite-Definitionsblatt

Das Erstellen einer Sprite-Grafik ist nicht immer ganz einfach. Vor allem das Umsetzen von Grafik in Bytes ist sehr unübersichtlich. Das „Sprite-Definitionsblatt" bietet eine kleine Hilfe. Zeichnen Sie den Sprite ein und tragen Sie die dezimalen Werte rechts in die entsprechenden Felder, dann können Sie die Daten als DATA-Zeilen in Programm übernehmen.

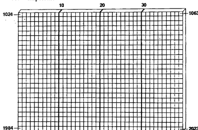
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

Abb. 31 Sprite-Definitionsblatt

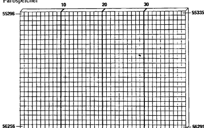
G2.1 Bildschirm- und Farbspeicher

Noch einmal ist der Bildschirm- und der Farbspeicher als Raster für das Planen und Erstellen von Bildschirmen dargestellt.

Bildschirmspeicher



Farbspeicher



G2.2 ASCII- und Bildschirmcode

Der Zusammenhang zwischen ASCII- und Bildschirmcode ist aus den beiden folgenden Tabellen zu entnehmen. „SATZ1" entspricht dem Groß/Grafik- und „SATZ2" dem Groß/Klein-Modus.

ASCII	CODE	SATZ 1	ASCII	CODE	SATZ 1	ASCII	CODE	SATZ 1
32	32		64	0	@	96	64	─
33	33	!	65	1	A	97	65	♠
34	34	"	66	2	B	98	66	│
35	35	#	67	3	C	99	67	─
36	36	$	68	4	D	100	68	─
37	37	%	69	5	E	101	69	─
38	38	&	70	6	F	102	70	─
39	39	'	71	7	G	103	71	│
[illegible]	40	(	72	8	H	104	72	│
	41	[illegible]	73	9	I	105	73	╮
			74	10	J	106	74	╰
			75	11	K	107	75	╯
			76	12	L	108	76	└
			77	13	M	109	77	╲
			78	14	N	110	78	╱
			79	15	O	111	79	┌
			80	16	P	112	80	┐
			81	17	Q	113	81	●
			82	18	R	114	82	─
			83	19	S	115	83	♥
			84	20	T	116	84	│
			85	21	U	117	85	╭
			86	22	V	118	86	╳
[illegible]			87	23	W	119	87	○
56			88	24	X	120	88	♣
57			89	25	Y	121	89	│
58			90	26	Z	122	90	♦
59			91	27	[	123	91	┼
60			92	~~60~~	£	124	92	▒
61	[illegible]		93	~~61~~	]	125	93	│
62	62		94	~~62~~	↑	126	94	π
63	63		95	~~63~~	←	127	95	◥
160	96		171	107	├	182	118	▐
161	97	▌	172	108	▗	183	119	▔
162	98	▄	173	109	└	184	120	▃
163	99	▔	174	110	┐	185	121	▁
164	100	▁	175	111	▂	186	122	┘
165	101	▏	176	112	┌	187	123	▖
166	102	▒	177	113	┴	188	124	▝
167	103	▕	178	114	┬	189	125	┘
168	104	▒	179	115	┤	190	126	▘
169	105	◤	180	116	▎	191	127	▚
170	106	▕	181	117	▍	192	64	─

ASCII	CODE	SATZ 2	ASCII	CODE	SATZ 2	ASCII	CODE	SATZ 2
32	32		64	0	@	96	64	—
33	33	!	65	1	a	97	65	A
34	34	"	66	2	b	98	66	B
35	35	#	67	3	c	99	67	C
36	36	$	68	4	d	100	68	D
37	37	%	69	5	e	101	69	E
38	38	&	70	6	f	102	70	F
39	39	'	71	7	g	103	71	G
40	40	(	72	8	h	104	72	H
41	41	)	73	9	i	105	73	I
42	42	*	74	10	j	106	74	J
43	43	+	75	11	k	107	75	K
44	44	,	76	12	l	108	76	L
45	45	-	77	13	m	109	77	M
46	46	.	78	14	n	110	78	N
47	47	/	79	15	o	111	79	O
48	48	0	80	16	p	112	80	P
49	49	1	81	17	q	113	81	Q
50	50	2	82	18	r	114	82	R
51	51	3	83	19	s	115	83	S
52	52	4	84	20	t	116	84	T
53	53	5	85	21	u	117	85	U
54	54	6	86	22	v	118	86	V
55	55	7	87	23	w	119	87	W
56	56	8	88	24	x	120	88	X
57	57	9	89	25	y	121	89	Y
58	58	:	90	26	z	122	90	Z
59	59	;	91	27	[	123	91	
60	60	<	92	60	£	124	92	⌇
61	61	=	93	61	]	125	93	│
62	62	◣	94	62	↑	126	94	▒
63	63	?	95	63	←	127	95	▧
160	96		171	107	├	182	118	▐
161	97	▌	172	108	▗	183	119	▔
162	98	▄	173	109	└	184	120	▔
163	99	▔	174	110	┐	185	121	▂
164	100	▁	175	111	▁	186	122	✓
165	101	▏	176	112	┌	187	123	▖
166	102	▒	177	113	┴	188	124	▝
167	103	▕	178	114	┬	189	125	┘
168	104	▒	179	115	┤	190	126	▘
169	105	◤	180	116	▎	191	127	▚
170	106	▕	181	117	▍	192	64	—

G2.3 Notenwerte

Mit dem SID-Chip ist es möglich, Töne im Bereich von 0 bis knapp 4000 Hertz zu erzeugen. Der Frequenzwert wird in einem dem Computer verständlichen Parameter nach der Formel

Parameter = Frequenz · 17.0284116

umgerechnet. Der Parameter ist dann als High- und Lowbyte in die Frequenzregister der Oszillatoren zu legen.

Aus der Tabelle können Sie die Frequenz, den Parameter und das High- und Lowbyte aller Töne entnehmen.

Note-Oktave	Frequenz (HZ)	Parameter	Highbyte	Lowbyte
C -0	16.4	278	1	22
C# -0	17.3	295	1	39
D -0	18.4	313	1	57
D# -0	19.4	331	1	75
E -0	20.6	351	1	95
F -0	21.8	372	1	116
F# -0	23.1	394	1	138
G -0	24.5	417	1	161
G# -0	26.0	442	1	186
A -0	27.5	468	1	212
A# -0	29.1	496	1	240
H -0	30.9	526	2	14
C -1	32.7	557	2	45
C# -1	34.6	590	2	78
D -1	36.7	625	2	113
D# -1	38.9	662	2	150
E -1	41.2	702	2	190
F -1	43.7	743	2	231
F# -1	46.2	788	3	20
G -1	49.0	834	3	66
G# -1	51.9	884	3	116
A -1	55.0	937	3	169

Note-Oktave	Frequenz (HZ)	Parameter	Highbyte	Lowbyte
A# -1	58.3	992	3	224
H -1	61.7	1051	4	27
C -2	65.4	1114	4	90
C# -2	69.3	1180	4	156
D -2	73.4	1250	4	226
D# -2	77.8	1325	5	45
E -2	82.4	1403	5	123
F -2	87.3	1487	5	207
F# -2	92.5	1575	6	39
G -2	98.0	1669	6	133
G# -2	103.8	1768	6	232
A -2	110.0	1873	7	81
A# -2	116.5	1985	7	193
H -2	123.5	2103	8	55
C -3	130.8	2228	8	180
C# -3	138.6	2360	9	56
D -3	146.8	2500	9	196
D# -3	155.6	2649	10	89
E -3	164.8	2807	10	247
F -3	174.6	2974	11	158
F# -3	185.0	3150	12	78
G -3	196.0	3338	13	10
G# -3	207.7	3536	13	208
A -3	220.0	3746	14	162
A# -3	233.1	3969	15	129
H -3	246.9	4205	16	109
C -4	261.6	4455	17	103
C# -4	277.2	4720	18	112
D -4	293.7	5001	19	137
D# -4	311.1	5298	20	178
E -4	329.6	5613	21	237
F -4	349.2	5947	23	59
F# -4	370.0	6301	24	157
G -4	392.0	6676	26	20
G# -4	415.3	7072	27	160
A -4	440.0	7493	29	69
A# -4	466.2	7939	31	3

H -4	493.9	8411	32	219
C -5	523.3	8911	34	207
C# -5	554.4	9441	36	225
D -5	587.3	10002	39	18
D# -5	622.3	10597	41	101
E -5	659.3	11227	43	219
F -5	698.5	11894	46	118
F# -5	740.0	12602	49	58
G -5	784.0	13351	52	39
G# -5	830.6	14145	55	65
A -5	880.0	14986	58	138
A# -5	932.3	15877	62	5
H -5	987.8	16821	65	181
C -6	1046.5	17821	69	157
C# -6	1108.7	18881	73	193
D -6	1174.7	20004	78	36
D# -6	1244.5	21193	82	201
E -6	1318.5	22454	87	182
F -6	1396.9	23789	92	237
F# -6	1480.0	25203	98	115
G -6	1568.0	26702	104	78
G# -6	1661.2	28290	110	130
A -6	1760.0	29972	117	20
A# -6	1864.7	31754	124	10
H -6	1975.5	33642	331	106
C -7	2093.0	35643	139	59
C# -7	2217.5	37762	147	130
D -7	2349.3	40008	156	72
D# -7	2489.0	42387	165	147
E -8	2637.0	44907	175	107
F -7	2793.8	47578	185	218
F# -7	2960.0	50407	196	231
G -7	3136.0	53404	208	156
G# -7	3322.4	56580	221	4
A -7	3520.0	59944	234	40
A# -7	3729.3	63508	248	20

G2.4 Diskettenaufbau

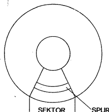


Abb. 32 Disk

Spuren	Sektoren	Anzahl/Spur
01-17	00-20	21
18-24	00-18	19
25-30	00-17	18
31-35	00-16	17

BAM (Block-Availability-Map)

Spur 18: Sektor 00

Bytes	Funktion
000-001	Spur/Sektor des ersten Directory Blocks
002	Formatzeichen ('A')
003	Null-Byte (unbenutzt)
004-007	Belegungskennzeichnung der Spur 01 (BAM)
008-011	Belegungskennzeichnung der Spur 02 (BAM)
....	....

140-143	Belegungskennzeichnung der Spur 35 (BAM)
144-161	Diskettenname
162-163	ID
164	Null-Bytes (unbenutzt)
165-166	Formatzeichen ('2A')
167-255	Null-Bytes (unbenutzt)

Belegungskennzeichnung am Beispiel: Byte 004-007 der BAM

Bytes	Funktion
004	Anzahl der freien Blöcke dieser Spur
005	Jedes Bit repräsentiert einen Sektor (Bit 0 für Sektor 00 bis Bit 7 für Sektor 07).
006	Jedes Bit repräsentiert einen Sektor (Bit 0 für Sektor 08 bis Bit 7 für Sektor 15).
007	Jedes Bit repräsentiert einen Sektor (Bit 0 für Sektor 16 bis Bit 7 für Sektor 23).

Directory-Aufbau

Erster Directory-Block Spur 18/ Sektor 01

Bytes	Funktion
000-001	Spur/Sektor nächster Directory-Eintrag.
002-031	File-Eintrag Nr. 1.
032-033	Null-Bytes (unbenutzt).
034-063	File-Eintrag Nr.2.
064-065	Null-Bytes (unbenutzt).
224-253	File-Eintrag Nr.8.
254-255	Null-Bytes (unbenutzt).

File-Eintrag am Beispiel Nr.1 (Bytes 002 - 031)

Bytes	Funktion
002	File-Typ
003-004	Spur/Sektor File-Beginn.
005-020	File-Name
021-022	Nur bei REL-Files benutzt: Spur/Sektor des Side-Sektor-Blocks. Sonst Null-Byte.
023	Nur bei REL-Files benutzt: Recordlänge. Sonst Null-Byte.
024-027	Null-Bytes (unbenutzt).
028-029	Spur/Sektor nach Überschreiben.
030-031	Anzahl der belegten Blöcke (Filelänge).

Side-Sektor-Block

Nur bei REL-Files vorhanden. Maximal 6 Side-Sektor-Blöcke.

Bytes	Funktion
000-001	Spur/Sektor nächster Side-Sektor-Block.
002	Side-Sektor-Block-Nummer.
003	Recordlänge.
004-005	Spur/Sektor Side-Sektor-Block Nr. 0
006-007	Spur/Sektor Side-Sektor-Block Nr. 1
014-015	Spur/Sektor Side-Sektor-Block Nr. 5
016-017	Spur/Sektor wo gesuchter Record zu finden ist.
018-019	Spur/Sektor wo gesuchter Record zu finden ist.
254-255	Spur/Sektor wo gesuchter Record zu finden ist.

G3.0 Tips und Tricks

Das COMMODORE V2 BASIC ist bekanntlich nicht das Stärkste seiner Art. Viele nützliche Funktionen lassen sich nur durch PEEKen und POKEn erzielen. Eine kleine Liste faßt einige PEEKs und POKEs zusammen.

Repeatfunktion der Tastatur:

`POKE 650, 0`	Keine Taste hat Dauerfunktion
`POKE 650,127`	Nur Cursor, Space, Delete
`POKE 650,255`	Alle Tasten haben Dauerfunktion

Geschwindigkeit der Cursor-Bewegung:

`POKE 56325, 5`	sehr schnell (unkontrolliert)
`POKE 56325,255`	sehr langsam
`POKE 56325, 50`	normal

Blinkgeschwindigkeit des Cursors:

`POKE 207, x`	x=1 bis 255

Zeile, in der sich der Cursor befindet:

`PRINT PEEK(214)`	0-23

Spalte, in der sich der Cursor befindet:

`PRINT PEEK(211)`	0-39

Cursor auf Zeile/Spalte setzen:

```
POKE 211, Spalte: POKE 214, Zeile: SYS 58640
```

Farbe unter Cursor ändern:

`POKE 647, x`	x=0-15

Bei GET ein Cursor erzeugen:

`POKE 204, 0`	Anschalten
`POKE 204, 1`	Ausschalten

Kein Cursor vor INPUT, bis eine Taste gedrückt wurde:

```
POKE 207, 1
```

Verzögern der Repeatfunktion:

```
POKE 652, x
```
x=1 bis 255

Letzte gedrückte Taste:

```
PRINT PEEK(215)
```
in Bildschirmcode

Welche Taste ist gedrückt:

```
PRINT PEEK(203)
```
64 = keine Taste

INPUT ohne Fragezeichen:

```
10 OPEN 1,0
20 INPUT#1, X
..
90 CLOSE 1
```

Oder:

```
10 POKE 19, 64
20 INPUT X
30 POKE 19, 0
```
(sehr wichtig!)

Farben setzen:

POKE 646, x	Zeichenfarbe: x= 1-15
POKE 53280, x	Borderfarbe : x= 1-15
POKE 53281, x	Bildschirmfarbe : x= 1-15

Bildschirmausgabe:

POKE 53265, PEEK(53265) AND 239	Bildschirm AUS
POKE 53265, PEEK(53265) OR 16	Bildschirm AN

Langsame Bildschirmausgabe:

POKE 56325, 0	alle PRINTs werden langsam ausgeführt.

RUN/STOP+RESTORE

POKE 808,255	nicht möglich
POKE 808,237	möglich

RUN/STOP

POKE 788, 52	nicht möglich
POKE 788, 49	möglich

Umschaltung Groß/Grafik- Groß/Klein-Modus:

POKE 53272, 21	Groß/Grafik-Modus
PRINT CHR$(142)	Groß/Grafik-Modus
POKE 53272, 23	Groß/Klein-Modus
PRINT CHR$(14)	Groß/Klein-Modus

Sperren der Grafik-Umschaltung:

POKE 657,128	die Umschaltung mit C=/SHIFT ist nicht mehr möglich.

Reverse:

```
10 POKE 199, 1: PRINT "TEXT"alles Reverse
20 POKE 199, 0: PRINT "TEXT"normal
```

Löschen einer Zeile auf dem Bildschirm:

POKE 781, Zeile: SYS 59903	löscht eine Zeile auf dem Bildschirm.

Zeile wo Programm gestoppt:
`PRINT PEEK(57)`

Länge der momentanen Zeile:
`PRINT PEEK(213)`

Anzahl der offenen Dateien:
`PRINT PEEK(152)`

Schließen aller offenen Dateien:
`SYS 65511`

Status-Variable (ST)
`PRINT PEEK(144)`

Anzahl der Lesefehler:

```
PRINT PEEK(182)
```

Uhr TI$ auf 0 stellen:

`SYS 65499`	**TI$='000000'**

Systemreset:

```
SYS 65499
```

Richtiges FREE:

```
PRINT FRE(0)+65538
```

Warten auf eine Taste:

`WAIT 203, x`	Wartet auf Taste x

Warten auf C= oder SPACE:

```
SYS 58592
```

Umrechnung ASCII nach Bildschirmcode:

```
C=A-161-33*(A<255)-64*(A<192)-32*(A<160)+32*(A<92)
      64*(A<64)
```

MERGE - zweites Programm zuladen:

```
Z=PEE(45)+PEE(46)*256
Z=Z-2
POKE 43, Z AND 255: POKE 44, Z/256
LOAD "PROG.2",8(1)
POKE 43,1: POKE 44,8
```

LIST-Schutz:

`POKE 774, 0`	nur Zeilennummern
`POKE 774, 27`	listet alles als Token
`POKE 774, 26`	normales LISTen.
`POKE 775, 1`	löscht Bildschirm und führt Reset durch
`POKE 775,167`	normales LISTen.

SAVE-Schutz:

POKE 0, 0	AN
POKE 0,47	AUS

POKE 819,253: POKE 818,253: POKE 808,225	AN
POKE 818, 34: POKE 819,253: POKE 808,225	AN
POKE 818,237: POKE 819,245: POKE 808,237	AUS

Zerstörungs-Befehle:

POKE 776,1	füllt Programm mit Zeichen
POKE 777,1	keine Eingabe mehr möglich
POKE 770,0	unendliche READY-Ausgabe
POKE 649,0	keine Eingabe mehr möglich
POKE 120,2	keine Eingabe mehr möglich

Registertabelle VIC : Adresse 53248 - 53294

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Name	Bemerkung
53248	X-Position von Sprite 0								X-S0	VIC+00
53249	Y-Position von Sprite 0								Y-S0	VIC+01
53250	X-Position von Sprite 1								X-S1	VIC+02
53251	Y-Position von Sprite 1								Y-S1	VIC+03
53252	X-Position von Sprite 2								X-S2	VIC+04
53253	Y-Position von Sprite 2								Y-S2	VIC+05
53254	X-Position von Sprite 3								X-S3	VIC+06
53255	Y-Position von Sprite 3								Y-S3	VIC+07
53256	X-Position von Sprite 4								X-S4	VIC+08
53257	Y-Position von Sprite 4								Y-S4	VIC+09
53258	X-Position von Sprite 5								X-S5	VIC+10
53259	Y-Position von Sprite 5								Y-S5	VIC+11
53260	X-Position von Sprite 6								X-S6	VIC+12
53261	Y-Position von Sprite 6								Y-S6	VIC+13
53262	X-Position von Sprite 7								X-S7	VIC+14
53263	Y-Position von Sprite 7								Y-S7	VIC+15
53264	Sprite 7 HB X-Pos	Sprite 6 HB X-Pos	Sprite 5 HB X-Pos	Sprite 4 HB X-Pos	Sprite 3 HB X-Pos	Sprite 2 HB X-Pos	Sprite 1 HB X-Pos	Sprite 0 HB X-Pos	X-MSB	VIC+16
53265	Rasterreg. HB von 18	Erw.Hinter. 1=an	Hires 1=an	Bildschirm 1=aus	Zeilenzahl 0=40 1=24	Y-Scrolling Wert für Zeilenverschiebung			MOD	VIC+17
53266	Rasterregister (LB im VIC + 17)								RAS	VIC+18
53267	Lightpen X-Koordinate								LPX	VIC+19
53268	Lightpen Y-Koordinate								LPY	VIC+20
53269	Sprite 7 1=ein	Sprite 6 1=ein	Sprite 5 1=ein	Sprite 4 1=ein	Sprite 3 1=ein	Sprite 2 1=ein	Sprite 1 1=ein	Sprite 0 1=ein	ON	VIC+21
53270	Unbenutzt		Reset Bit	Mehrfarb. 1=an	Spaltenzahl 0=38 1=40	X-Scrolling Wert für Spaltenverschiebung			MOD	VIC+22
53271	Sprite 7 1=Y-Dehn	Sprite 6 1=Y-Dehn	Sprite 5 1=Y-Dehn	Sprite 4 1=Y-Dehn	Sprite 3 1=Y-Dehn	Sprite 2 1=Y-Dehn	Sprite 1 1=Y-Dehn	Sprite 0 1=Y-Dehn	Y-EXP	VIC+23
53272	Startadresse des Bildschirmspeichers				Startadresse der Bitmap				BIT	VIC+24
53273	Interrupt-Flaggen-Register				Lightpen Interrupt	Sprite/Sprite Interrupt	Sprite/Hint. Interrupt	Raster Interrupt	IFR	VIC+25
53274	Interrupt-Masken-Register				Lightpen Interrupt	Sprite/Sprite Interrupt	Sprite/Hint. Interrupt	Raster Interrupt	IMR	VIC+26
53275	Sprite 7 1=Hinter	Sprite 6 1=Hinter	Sprite 5 1=Hinter	Sprite 4 1=Hinter	Sprite 3 1=Hinter	Sprite 2 1=Hinter	Sprite 1 1=Hinter	Sprite 0 1=Hinter	H/PRIO	VIC+27
53276	Sprite 7 1=Multic.	Sprite 6 1=Multic.	Sprite 5 1=Multic.	Sprite 4 1=Multic.	Sprite 3 1=Multic.	Sprite 2 1=Multic.	Sprite 1 1=Multic.	Sprite 0 1=Multic.	MULTI	VIC+28
53277	Sprite 7 1=X-Dehn	Sprite 6 1=X-Dehn	Sprite 5 1=X-Dehn	Sprite 4 1=X-Dehn	Sprite 3 1=X-Dehn	Sprite 2 1=X-Dehn	Sprite 1 1=X-Dehn	Sprite 0 1=X-Dehn	X-EXP	VIC+29
53278	Sprite 7 1=SS Kol	Sprite 6 1=SS Kol	Sprite 5 1=SS Kol	Sprite 4 1=SS Kol	Sprite 3 1=SS Kol	Sprite 2 1=SS Kol	Sprite 1 1=SS Kol	Sprite 0 1=SS Kol	S/S KOL	VIC+30
53279	Sprite 7 1=SH Kol	Sprite 6 1=SH Kol	Sprite 5 1=SH Kol	Sprite 4 1=SH Kol	Sprite 3 1=SH Kol	Sprite 2 1=SH Kol	Sprite 1 1=SH Kol	Sprite 0 1=SH Kol	S/H Kol	VIC+31
53280	Unbenutzt				Farbe des Bildschirmrahmens				BORDER	VIC+32
53281	Unbenutzt				Hintergrundfarbe				PAPER	VIC+33
53282	Unbenutzt				Erweiterte Farbe Register 1				COL 1	VIC+34
56283	Unbenutzt				Erweiterte Farbe Register 2				COL 2	VIC+35
56284	Unbenutzt				Erweiterte Farbe Register 3				COL 3	VIC+36
56285	Unbenutzt				Sprite-Mehrfarben-Register 0				MSCOL 0	VIC+37
56286	Unbenutzt				Sprite-Mehrfarben-Register 1				MSCOL 1	VIC+38
56287	Unbenutzt				Farbregister Sprite 0				SCOL 0	VIC+39
56288	Unbenutzt				Farbregister Sprite 1				SCOL 1	VIC+40
56289	Unbenutzt				Farbregister Sprite 2				SCOL 2	VIC+41
56290	Unbenutzt				Farbregister Sprite 3				SCOL 3	VIC+42
56291	Unbenutzt				Farbregister Sprite 4				SCOL 4	VIC+43
56292	Unbenutzt				Farbregister Sprite 5				SCOL 5	VIC+44
56293	Unbenutzt				Farbregister Sprite 6				SCOL 6	VIC+45
56294	Unbenutzt				Farbregister Sprite 7				SCOL 7	VIC+46