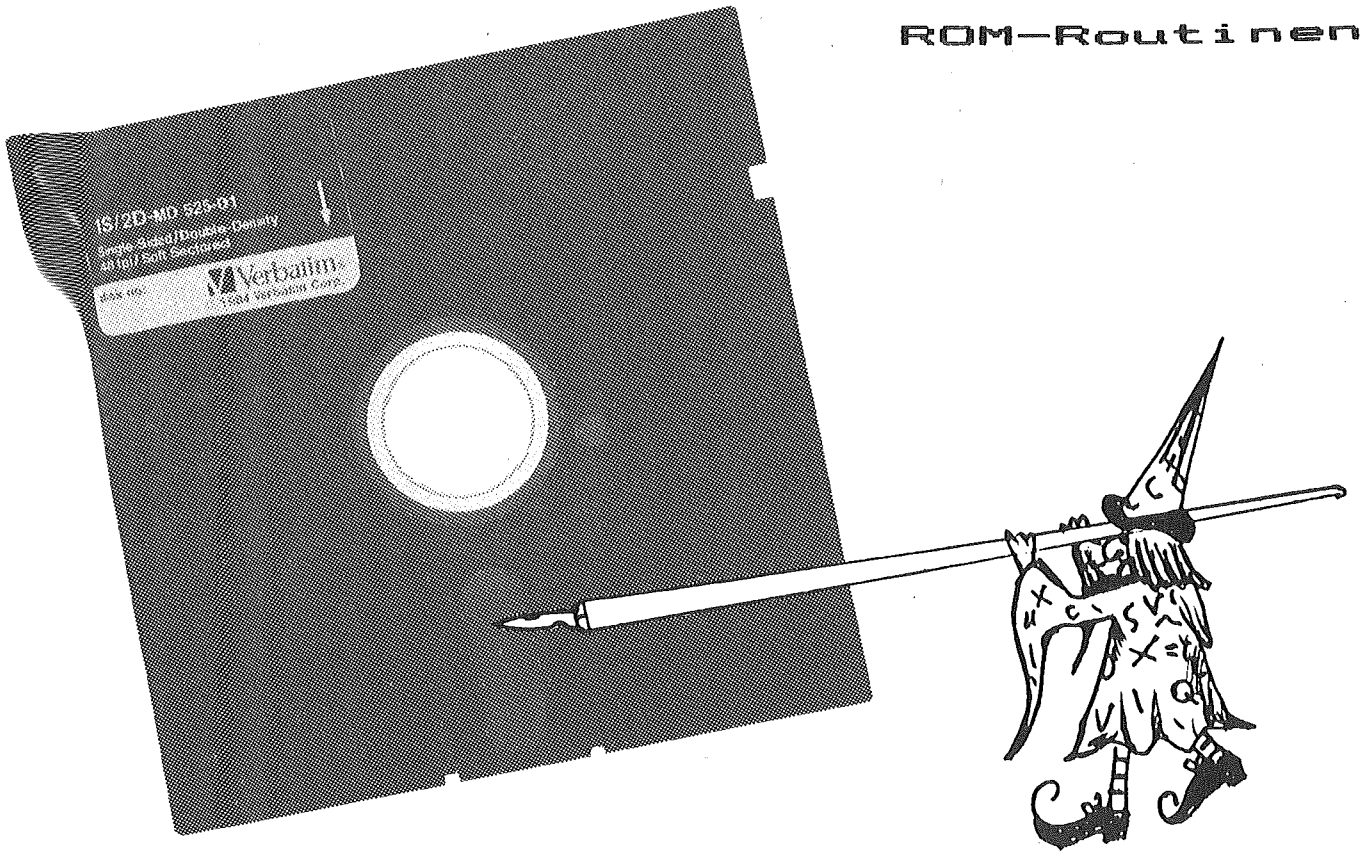
SV JOURNAL

Die Zeitschrift des Spectra-Video-Club Austria

ROM—Routinen



Diskeditor in BASIC

Heft 7/85

MSX

S 15.-

Lieber SVI-Journal-Leser!

Wir fragen uns manchmal wirklich, was mit unseren Clubmitgliedern los ist. Als wir noch "magere" 30 bis 50 Leute im Club waren, bevölkerten Mitglieder die Clubabende derart, daß die uns zur Verfügung gestellten Räume fast zu klein waren. Doch seit einiger Zeit müssen wir feststellen, daß weniger Personen als früher an den diversen Abenden ins Clublokal kommen. An ein bis zwei Abenden waren überhaupt nur drei bis vier anwesend! Ist vielleicht das Interesse gesunken?

Wir wollen nicht den Fehler begehen, den schon viele Computerclubs vor uns begangen haben. Daß sich nämlich kleine Gruppen bilden, die sich gar nicht mehr im Clublokal treffen, beziehungsweise wenn sie sich treffen, daß nur in der Gruppe kommuniziert wird. "Clubneulinge" stehen dann meistens in einer Ecke, warten vergeblich auf einen Computer und wissen auch nicht, an wen sie sich wenden sollen. Um die Kommunikation zwischen Clubmitgliedern zu fördern, wollen wir folgendes vorschlagen:

Damit es wieder verstärkten Anreiz gibt, zu den Clubabenden zu kommen, sollten wir vielleicht größere Softwareprojekte starten. Es gibt Spezialisten auf dem Graphiksektor, Soundprogrammierer und ähnliches. Einer allein kann zum Beispiel nur schwer ein ganzes Maschincodespiel schreiben. Zu dritt oder zu viert wird es schon eher möglich. Ideen dazu gibt es genug:

Flugsimulator, 3D-Autorennen, Simulationen von Vorgängen (eventuell das Rinnen von Wasser in Behältersystemen), Steuerungssysteme (vielleicht ein Punkt, der am Bildschirm selbsttätig seinen Weg von links nach rechts findet und beliebigen Hindernissen ausweicht), Schach, Mühle, ein Adventurespiel mit Graphik und viele andere Algorithmen (auch 'professionelle').

Wenn Sie an so einem Projekt Interesse haben, dann kommen Sie zu uns und sagen Sie uns Ihre Stärken, und sei es nur das Wissen, sehr gut schachzuspielen. Jedes Programmierteam für Schachcomputer braucht gute Schachspieler, die Tricks und die gebräuchlichsten Eröffnungen kennen. Wir können nicht garantieren, daß auch nur ein Projekt gestartet werden kann. Aber es ist zumindest einen Versuch wert. Neben den - bestimmt sehr

INHALT

Die nächsten Clubtermine:

Sonderregelung für August!

Sa, 3. August 1985, ab 17 Uhr
Mi, 21. August 1985, ab 19 Uhr
Di, 27. August 1985, ab 19 Uhr

Clubabende wie immer im Clublokal im Computer-Studio, 1040 Wien, Paniglgasse 18-20
Nichtmitglieder sind willkommen
Ende jeweils ca. 22 Uhr!

Aktivitäten an den Clubabenden:
Arbeiten an Spectravideo-Systemen, Informationsaustausch zwischen Clubmitgliedern, Gelegenheit zum Ausdrucken von Programmlistings.

Telefonische Auskünfte über den Club und seine Aktivitäten erhalten Sie unter der Telefonnummer 65 88 93.

guten - Ergebnissen haben wir wieder einen aktiven Club!

Daß wir kritische Clubmitglieder haben, die auch bereit sind, aktiv mitzuwirken, wissen wir. Die Fragebogenaktion hat uns inzwischen schon zwölf Meinungen von Lesern vermittelt. Wir werden uns das Ergebnis zu Herzen nehmen. Da fast alle der Meinung waren, daß zu wenig Systeminformationen im Journal zu finden sind (wir sind übrigens der gleichen Meinung), haben wir schon jetzt dem Trend Rechnung getragen und veröffentlichen einmal die Sprungtabelle des ROMS zwischen 38H und 180H. Weitere Informationen werden folgen.

Um noch kurz zum Clubabend zurückzukommen! Es ist natürlich ein Service unseres Clubs, daß man bei einem Programmlisting, das man unbedingt ausdrucken möchte, nicht immer auf den nächsten Clubabend warten muß. Da unser Clublokal in den Geschäftsräumen der Wehsner Ges.m.b.H. ist, sind die Räume, in denen unser Club zu finden ist, den ganzen Tag geöffnet. Das heißt aber nicht, daß dort den ganzen Tag Club stattfindet. Wir haben ein Arrangement mit der Firma, daß Clubmitglieder Listings ausdrucken können, aber Stunden vor dem Computer sitzen, muß denn das sein?

Etwas Interesse haben wir auch für eine Art Leserbriefkasten erkannt. Die Fragen, die uns erreichen, werden wir natürlich beherzigen und veröffentlichen. Doch auch hier gilt wieder: Je mehr mitmachen, desto mehr können wir veröffentlichen. Im übrigen möchten wir wieder daran erinnern, daß Programmlistings bei uns willkommen sind. Einen großen Teil unserer Zeitschrift füllt ja diesmal das Programm 'Funktionsplotter' aus. Daß wir manchmal Veränderungen vornehmen müssen, darum bitten wir um Verzeihung, aber oft verlangt es die Länge des Programms oder das Programm selber, korrigierend einzugreifen. Das sollte aber keinen abhalten, uns sein Programm zu schicken (oder vorbeizubringen). Es brauchen nicht unbedingt Assemblerlistings oder BASIC-Listings sein. Von Lisp bis Pascal nehmen wir alles!

Ihr SVI-Journal-Chefredakteur Gerhard Fally!

Sprites sind besonders für Spiele sehr nützliche Einrichtungen. Bewegte Graphiken können mit ihnen viel einfacher dargestellt werden. Leider mangelt es oft an vollständiger Erklärung seitens der Computerhersteller. Sprites sind nämlich etwas komplizierter, als etwa ein LINE-Befehl. Wir wollen daher das Hauptgewicht dieser Folge auf die Sprites des SVI legen.

Schon in der vorigen Folge haben wir einiges über Sprites - vor allem Grundsätzliches - gehört. Nun wollen wir uns näher ansehen, wie man Sprites generiert, beziehungsweise wie man mit ihnen umgeht. Wir haben ja schon davon gesprochen, daß Sprites beim Spectravideo entweder aus 8*8 oder aus 16*16 Punkten bestehen können. Wir sehen uns nun mit Hilfe einiger Graphiken den Werdegang eines Sprites an:

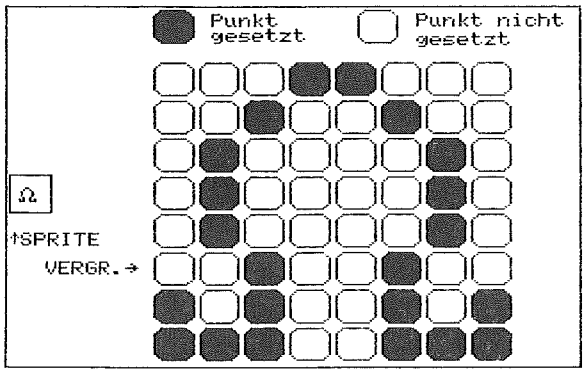


*Bild 1: Links im Bild ist im kleinen Rahmen ein 8*8-Sprite ersichtlich. Rechts davon sieht man es vergrößert wieder. Nun kann man die einzelnen Punkte erkennen und sieht, wie das Sprite definiert wurde. Die ausgefüllten Punkte sind dabei am Bildschirm gesetzt, die leeren bleiben transparent.*

In Bild 1 ist ein Sprite ersichtlich, vergrößert sieht es wie im Bild rechts abgebildet aus. Jeder Punkt ist entweder gesetzt oder unsichtbar. Wenn wir nun die Form unserem Computer mitteilen wollen, müssen wir die Gestalt des Sprites in einen RAM-Speicher schreiben. Anders versteht es der Computer nicht. Da bei uns die Speicher byteweise organisiert sind, das heißt, eine Speicherzelle faßt 8 Bits, liegt es nahe, die Form des Sprites ebenfalls in eine Achterstruktur zu bringen. Da unser Sprite sowieso die Seitenlänge 8 hat, wird es für uns relativ einfach. Wir legen fest, daß jede Zeile des Sprites durch ein Byte dargestellt wird. Jedes Bit der Bytes legt dann fest, ob der angesprochene Punkt im Sprite sichtbar oder unsichtbar sein soll. Da wir acht Zeilen haben, brauchen wir acht Bytes. Nun müssen wir lediglich acht Bytes im RAM reservieren und dem Computer sagen, daß dort die Form des Sprites zu finden ist. Reserviert wurden diese Plätze übrigens schon, die Programmierer von Microsoft haben sie im Video-RAM untergebracht. Ab der Adresse 3800H liegen die Formen für die 32 Sprites. Bei der Adresse 3800H liegt Sprite 0, bei 3820H Sprite 1 und so weiter. Warum 32 Bytes für ein Sprite reserviert wurden, klären wir später.

Wollen wir nun ein Sprite definieren, brauchen wir nur mittels VPOKE einzelne Werte ins VRAM poken. Wir tun dies mit einem kleinen Programm. Durch diese Routine wird Sprite 0 aktiviert. Wichtig ist für uns vorerst nur die Lese- und Speicherroutine. Der Rest dient nur dem Positionieren am Bildschirm.

```
10 SCREEN1
20 FOR I=&H3800 TO &H3807:READA:POKEI,A:NEXT
30 PUTSPRITE 0,(128,96),,0
40 GOTO 40
50 DATA 24,36,66,66,66,36,165,231
```

Mit POKE oder VPOKE wird bei manchen Computern auch hantiert, wenn man Sprites definieren will. Wir haben das aber jetzt nur testweise probiert. Obwohl es manche Clubmitglieder etwas stört, daß wir soviel Werbung für Spectravideo machen, wollen wir doch wieder erwähnen, daß das SVI-BASIC komfortablere Befehle zur Verfügung stellt (Spectravideo ist nun mal besser als so mancher andere Computer). Wir werden nämlich mit SPRITE$ arbeiten. Mit SPRITE$ werden keine Bytes sondern ein String übergeben. Ein Sprite wird demnach so definiert:

```
SPRITE$('Spritenummer')='String'
```

Wie kommt nun dieser String zustande?

Die Antwort ist ganz einfach: Jedes Zeichen eines Strings wird im Computer intern als Zahl abgelegt. Der Interpreter bedient sich hier des ASCII-Codes. Da der ASCII-Code nur 8 Bit Datenbreite hat, verwendet der Computer pro Zeichen ein Byte. Wenn wir nun 8 Zeichen zu einem String zusammenfassen, sind im Computer acht Bytes verbraucht. Jedes Byte beinhaltet nun eine bestimmte Zahl, die ihrerseits wieder eine bestimmte Form einer Spritezeile repräsentieren kann. So kann man zur Definition eines Sprites auch Zeichen verwenden. In Bild 2 ist das Ganze noch einmal veranschaulicht.

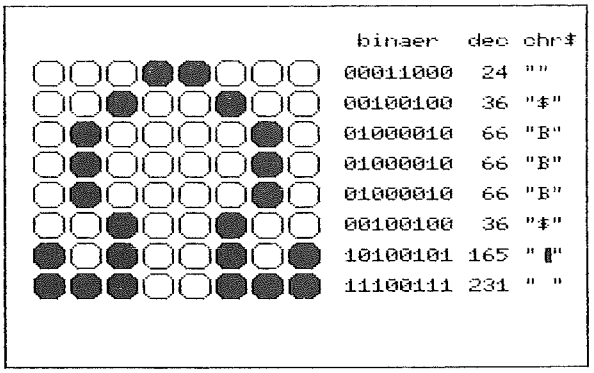


Bild 2: Das Sprite von Bild 1 wird in eine Binärmatrix zerteilt. Jede Zeile entspricht einem Byte. Die Binärzahlen werden weiter in dezimale Zahlen umgewandelt. Zum Schluß kann man noch den Zeichencode mittels CHR$ bestimmen. Die beiden Codes 24 und 231 erzeugen keine druckbaren Zeichen.

Wir ändern daher unser Programm und sehen gleichzeitig, daß es etwas kürzer wird. Wir müssen nämlich bedenken, daß die Routine, die wir ganz oben verwendet haben, nur für ein Sprite gilt. Für alle 32 Sprites sieht sie komplizierter aus:

vollständige Routine mit VPOKE:

```
10 SCREEN1
20 FOR I=T*32+&H3800 TO T*32+&H3807:READA
25 VPOKEI,A:NEXT
30 PUTSPRITE 0,(128,96),,0
40 GOTO 40
50 DATA 24,36,66,66,66,36,165,231
```

Routine mit SPRITE$:

```
10 SCREEN1
20 FOR I=1 TO 8:READA:S$=S$+CHR$(A):NEXT
30 SPRITE$(T)=S$:PUTSPRITE0,(128,96),,0
40 GOTO 40
50 DATA 24,36,66,66,66,36,165,231
```

T steht in beiden Programmen für eine Spritenummer zwischen 0 und 31.

Nun kann man den Unterschied deutlicher sehen. Neben dem angenehmen Effekt, daß die Routine kürzer wird, kann man sich den Algorithmus auch leichter merken. Wieder die obligate Frage: Was ist leichter zu merken, die Zahlen 3800H - 3BFFH oder SPRITE$('Zahl')?

In Spezialfällen kann man auch einen direkten String als Sprite zulassen. SPRITE$(0)="ABCDEFG" ist natürlich auch erlaubt. Bei SPRITE$ muß man übrigens nicht die volle Länge von acht Zeichen definieren, wie oben kann man ruhig weniger Buchstaben angeben, der Computer füllt den Rest mit dem ASCII-Code 0 auf.

Bevor wir uns nun (noch) bequemere Möglichkeiten zum Spritedefinieren suchen, wollen wir auf die bis jetzt sträflich vernachlässigten 16*16 Sprites eingehen. Bei 8*8 Sprites war ja alles recht einfach, eine Zeile war ein Byte. Bei 16*16er-Formen wird das Ganze etwas schwieriger. Wir können nicht mehr eine Zeile pro Byte angeben, daher müssen wir zwei Bytes pro Zeile spendieren. Jetzt braucht man aber nicht zu glauben, daß die Bytes Eins und Zwei in der ersten Zeile zu finden sind, sondern die Bytes Eins und Sechzehn. Dies erklärt sich aus der Speicheraufteilung eines 16*16-Sprites. Zuerst werden 16 Bytes für die linke Seite verwendet, danach 16 für die rechte Seite. Bild 3 veranschaulicht dies.

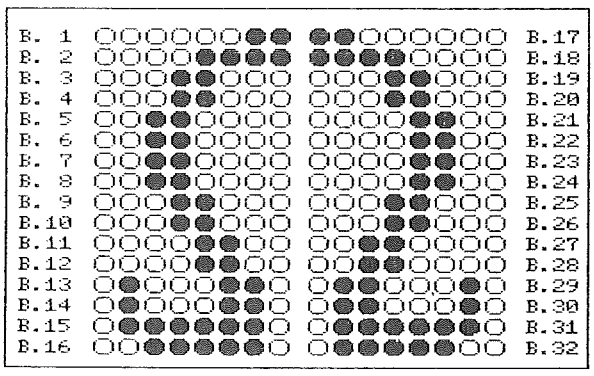


*Bild 3: Die Organisation eines 16*16-Sprites. Die linke Hälfte wird durch die ersten 16 Bytes, die rechte durch die zweiten 16 Bytes gebildet. Daraus resultiert, daß die Spritegenerierung von größeren Sprites etwas komplizierter wird.*

Ab nun werden wir immer zwei Routinen abdrucken, eine für die kleinen und eine für die großen Sprites.

Wie wir schon oben gesehen haben, speichert man in der Regel die Formen der einzelnen Spritezeilen in DATA-Zeilen ab. Dabei werden aber keine Zeichen sondern Zahlen abgelegt. Es gibt ja für einige ASCII-Codes keine eindeutigen Zeichen am Bildschirm. So wird man zum Beispiel die ersten 32 Codes nur äußerst schlecht ohne CHR$('Zahl") zu Strings wandeln können. Lediglich in den seltensten Fällen läßt sich ein Sprite nur aus eindeutigen Bildschirm-Codes darstellen. Ist dies jedoch der Fall, kann man wie oben erwähnt gleich direkt das Sprite definieren.

Da Zeichen oder Zahlen nur dem Computer zeigen, wie das Sprite aussieht, nicht aber einem Bediener, der das Programm gerade auflistet, wollen wir einen weiteren Algorithmus vorstellen, der bedienerfreundlicher ist. Wir speichern nicht mehr die Zahlen als Zahlen ab, sondern in ihrer binären Form. 33 wird zum Beispiel als 00100001B abgelegt. Man sieht schon, das Ganze wird viel anschaulicher. Nun erkennt man auch ohne Rechnerei sofort die Form des verwendeten Sprites. Die Leseroutine wird jedoch etwas komplizierter. Wir müssen die Binärzahl als String lesen. Dann "stellen" wir vor diesen String die beiden Zeichen "&B". "&B" bewirkt, daß die dahinterstehenden Ziffern als Teile einer Binärzahl erkannt werden. 00100001 ist für den Computer demnach nicht mehr 100001D, sondern 100001B. Um nun auch wirklich eine Zahl zu erhalten, verwenden wir VAL. VAL wandelt, wie wir schon wissen, einen String in eine Zahl um. Ab nun funktioniert alles analog zu den vorigen Leseroutinen.

```
10 SCREEN 1
20 S$=""
30 FOR I=1 TO 8
40 READ A$
50 S$=S$+CHR$(VAL("&B"+A$))
60 NEXT
70 SPRITE$(0)=S$
80 GOTO 70
90 DATA 00011000
91 DATA 00100100
92 DATA 01000010
93 DATA 01000010
94 DATA 01000010
95 DATA 00100100
96 DATA 10100101
97 DATA 11100111
```

Zur Veranschaulichung zeigen wir noch, wie die Routine, die eigentlich sehr komprimiert geschrieben ist, mit Variablen arbeiten würde:

```
                A$ enthält      B enthält

READ A$:          00100001      unwichtig
A$="&B"+A$       &B00100001     unwichtig
B=VAL(A$)        &B00100001        33
C$=CHR$(B)       &B00100001        33
            C$ enthält "!"
```

Im Prinzip könnten wir nun genauso mit 16*16-Sprites verfahren. Nur ergibt sich hier die Schwierigkeit, daß zuerst die linke Seite und dann die rechte eingelesen wird. Außerdem muß man eine Zeile in zwei Teile teilen. Hier gibt es drei gebräuliche Methoden:

Jede Zeile wird in der Mitte durch einen Beistrich getrennt. Zwei Zeichenketten lesen separat die linke und die rechte Hälfte ein. Erst am Schluß werden beide zusammengefügt:

```
10 S$="":T$=""
20 FOR I=1 TO 16:READ A$,B$
30 S$=S$+CHR$(VAL("&B"+A$))
40 T$=T$+CHR$(VAL("&B"+B$))
50 NEXT
60 SPRITE$(0)=S$+T$
50 DATA 00000011,11000000
51 DATA 00001111,11110000
52 DATA 00011000,00011000
53 DATA 00011000,00011000
54 DATA 00110000,00001100
55 DATA 00110000,00001100
56 DATA 00110000,00001100
57 DATA 00110000,00001100
58 DATA 00011000,00011000
59 DATA 00011000,00011000
60 DATA 00001100,00110000
61 DATA 00001100,00110000
62 DATA 01000110,01100010
63 DATA 01000110,01100010
64 DATA 01111110,01111110
65 DATA 00111110,01111100
```

Da der Beistrich zwischen den Punkten störend ist, kann man noch eine zweite Variante aufstellen, die zwar etwas komplizierter ist, die Übersichtlichkeit jedoch steigert:

```
10 S$=SPACE$(32):FOR I=1 TO 16:READ A$
20 MID$(S$,I)=CHR$(VAL("&B"+LEFT$(A$,16)))
30 MID$(S$,I+16)=CHR$(VAL("&B"+RIGHT$(A$,16)
))
40 NEXT
45 SPRITE$(0)=S$+T$
50 DATA 0000001111000000
60 DATA 0000111111110000
65 DATA 0001100000011000
70 DATA 0001100000011000
75 DATA 0011000000001100
80 DATA 0011000000001100
90 DATA 0011000000001100
91 DATA 0011000000001100
92 DATA 0001100000011000
93 DATA 0001100000011000
94 DATA 0000110000110000
95 DATA 0000110000110000
96 DATA 0100011001100010
97 DATA 0100011001100010
98 DATA 0111111001111110
99 DATA 0011111001111100
```

Durch MID$ wird jede Zeile in zwei Teile geteilt. Vorher wurde die Länge von S$ auf 32 fixiert. Nun wird die linke Hälfte des Sprites in die ersten 16 Bytes gespeichert, die rechte Hälfte kommt gleichzeitig in die Bytes 17-32.

Eine dritte Möglichkeit gibt es noch. Sie ist zwar die kürzeste und übersichtlichste, dauert aber etwas länger. Die DATA-Zeilen bleiben wie im Beispiel Zwei:

```
10 S$="":FOR I=0 TO 31:READ A$
20 S$=S$+CHR$(VAL("&B"+MID$(A$,I/16+1,16)))
30 NEXT
40 SPRITE$(0)=S$
```

Jetzt ist übrigens auch klar, wieso 32 Bytes für ein Sprite reserviert werden. Die großen 16*16-Matrizen brauchen 32 Byte im VRAM. Um unnötige Rechenarbeiten zu vermeiden, definiert man daher auch die Speicherbereiche der kleinen Sprites auf 32 Zeichen, obwohl das kleine Sprite nur 8 Zeichen in Anspruch nimmt.

Es kann manchmal vorkommen, daß man die Form eines Sprites braucht. In diesem Fall kann man mit der Funktion SPRITE$ in jeden beliebigen String jedes Sprite abspeichern. Dadurch kann auch die unsinnig anmutende Zeile 'SPRITE$(0)=SPRITE$(1)' erscheinen, die bei näherer Betrachtung sehr wohl sinnvoll ist.

Wie schon in der vorigen Folge erwähnt, kann man durch den SCREEN-Befehl einstellen, welche Spritegröße man am Bildschirm sehen möchte. Wir werden jetzt ein Programm schreiben, das ein Sprite definiert und am Bildschirm bewegt.

Dabei schneiden wir ein zweites Kapitel der Spriteprogrammierung an. Zuerst wird ein Gebilde definiert, danach muß es jedoch auch richtig am Schirm positioniert werden. Hier ist PUTSPRITE zuständig. Die Syntax von PUTSPRITE:

PUTSPRITE 'Ebene', ('X-Koordinate', 'Y-Koordinate'), 'Farbe', 'Spritenummer'

Für 'Ebene' und 'Spritenummer' kann man einen Wert zwischen 0 und 31 einsetzen. Wenn mehrere Sprites verwendet werden, muß man aufpassen, daß immer nur ein Sprite pro Ebene existiert. Zwei Sprites sind nicht erlaubt. Man kann jedoch ein Sprite in mehreren Ebenen unterbringen.

'Farbe' dürfte klar sein. Alle gesetzten Punkte nehmen die angegebene Farbe an, die nicht gesetzten bleiben transparent. Mit 'X-Koordinate', 'Y-Koordinate' wird immer der Punkt angegeben, auf dem die linke obere Ecke des Sprites liegen soll.

Mit diesem Wissen können wir nun ein kleines Programm schreiben:

```
5 SCREEN1,2
10 S$="":FOR I=0 TO 31:READ A$
20 S$=S$+CHR$(VAL("&B"+MID$(A$,I/16+1,16)))
30 NEXT
40 SPRITE$(0)=S$
41 REM *** POSITIONIERUNG
45 FOR I=191 TO 0 STEP -1
46 PUTSPRITE 0,(110,I),,0
47 NEXT:GOTO45
50 DATA 0000001111000000
60 DATA 0000111111110000
65 DATA 0001100000011000
70 DATA 0001100000011000
75 DATA 0011000000001100
80 DATA 0011000000001100
90 DATA 0011000000001100
91 DATA 0011000000001100
92 DATA 0001100000011000
93 DATA 0001100000011000
94 DATA 0000110000110000
95 DATA 0000110000110000
96 DATA 0100011001100010
97 DATA 0100011001100010
98 DATA 0111111001111110
99 DATA 0011111001111100
```

Am Bildschirm wird ein Ohmzeichen sichtbar, das sich senkrecht am Bildschirm auf und ab bewegt. Wenn wir die Zeilen 40-50 modifizieren, bewegt sich das Zeichen auch der Seite nach (oder nur der Seite nach, wenn in der Schleife statt 'STEP -1' 'STEP 0' eingesetzt wird).

```
41 REM *** POSITIONIERUNG
45 FOR I=191 TO 0 STEP-1:T=T+3
46 PUTSPRITE 0,(T,I),,0
47 NEXT:GOTO45
```

Da die X-Koordinate automatisch durch ein AND 255 verknüpft wird, brauchen wir für T keinen Regulator. Bis zur Zahl 32767 in der Variablen T erscheint das Raumschiff immer wieder am Bildschirm, wenn es einmal am Rand verschwindet.

In der nächsten Folge schließen wir die Sprites ab und wenden uns den letzten Anweisungen zu.

Das Directory unter CP/M

Jede unter CP/M benutzte Diskette ist (wie auch unter Disk-BASIC) in 40 Tracks (von 0 bis 39 nummeriert) geteilt.

Der Unterschied zu Disk-BASIC besteht in der Byteanzahl pro Sektor und in der Anordnung des Directorys.

Die Tracks einer CP/M-Diskette können wie folgt eingeteilt werden:

Track	Bedeutung
00 .. 02	CP/M Betriebssystem
03	Directory
04 .. 39	frei für Files

Das CP/M-System ist auf denselben Tracks abgespeichert, auf denen sich sonst das Disk-BASIC-System befindet. Das Directory befindet sich im Gegensatz zu Disk-BASIC auf Track 3 und nicht auf Track 20. Hierbei muß noch beachtet werden, daß nicht der gesamte Track für das Directory zur Verfügung steht. Es werden nur die Sektoren von 1 bis 8 benutzt, der Rest (Sektor 9 bis 34) ist ebenfalls frei für die Speicherung von Files. Wie bereits oben erwähnt, besitzt ein CP/M-Sektor eine andere Anzahl von Bytes als ein Disk-BASIC-Sektor. Genauer gesagt, er hat nur 128 Bytes statt der physikalischen 256 Bytes. Daraus folgt auch, daß es 34 statt 17 Sektoren pro Track gibt.

Im CP/M-Directory ist für jede Datei ein 32 Byte langer Eintrag vorgesehen. In diesem Eintrag finden Sie dann ähnlich wie unter Disk-BASIC die Angaben über den Filenamen, den Filetyp, die gesetzten Attribute und selbstverständlich auch die genaue Position des Files. Diese Einteilung ist wesentlich simpler als unter Disk-BASIC. Diese Einfachheit wird uns später noch von Nutzen sein.

Der 32 Byte lange Eintrag hat folgende Einteilung:

Byte	Bedeutung
00	Usernummer (00 - 0F (hex))
01 -- 08	Filename (im ASCII-Format)
09 -- 11	Filetyp (im ASCII-Format)
12	Fortsetzungstrack
13 -- 14	sind im Directory 00
15	Anzahl der Records
16 -- 31	Position der Datenblöcke

Filename und Filetyp sind bereits von der letzten Folge her bekannt.

In Byte 12 steht der Fortsetzungstrack des Files. Dieses Byte wird nur benutzt, wenn die Datei eine Größe von 16 Kbyte Überschreitet.

Byte 13 und 14 werden nur während des Zugriffs auf die Datei verwendet. Im Directory sind sie sonst immer auf 00 gesetzt.

Byte 15 enhält die Anzahl der Records beziehungsweise Sektoren. Record bedeutet dasselbe wie Sektor und enthält somit ebenfalls 128 Bytes.

In den Bytes 16 bis 31 steht die genaue Lage der Datenblöcke.

Um die Lage dieser Datenblöcke zu berechnen, benötigen Sie die folgende Formel:

```
Track = INT ((Byteinhalt * 4) / 17) + 3

Sektor = (Byteinhalt * 4) MOD 17 + 1
```

Wenn Sie den Inhalt des Bytes berechnen wollen, hilft Ihnen diese Formel:

```
Byteinhalt = (Track * 17 + Sektor) / 4 - 13
```

Der kleinste Speicherplatz, den CP/M für ein File zur Verfügung stellen kann, beträgt 1024 Byte oder 1 Kbyte. Das entspricht 8 CP/M-Sektoren oder 4 Disk-BASIC-Sektoren. Dieser Wert liegt deutlich unter dem von Disk-BASIC (dort sind es 17 Sektoren mit einer Länge von 256 Byte (= 4.25 Kbyte)). Kurz gesagt, die Fileverwaltung unter CP/M ist bedeutend ökonomischer.

Wenn Sie an die Länge der Anleitung zum Restaurieren von geKILLten Files im letzten Heft zurückdenken, so werden Sie sicher von der Kürze der Beschreibung für CP/M Dateien überrascht sein. Da der Befehl ERA unter CP/M, wie üblich, nur geringe Veränderungen auf der Diskette (im Directory oder im Allocation Table) vornimmt, ist es wieder mehr oder weniger leicht möglich, gelöschte Files zu reparieren. Bei der Wiederherstellung von gelöschten Files unter CP/M spielt das Byte 0 eine sehr große, um nicht zu sagen, die entscheidende Rolle. In diesem Byte steht normalerweise eine Usernummer zwischen 0 und 17 (siehe obige Tabelle). Ist das File gelöscht, so steht keine Zahl im Userebenenbereich in diesem Byte, sondern die Zahl E5 (hex). Die Zahl E5 bezeichnet allgemein einen freien Speicherplatz auf der Diskette. Wenn Sie eine leere Diskette (selbstverständlich formatiert) mit einem Diskeditor (z.B.: DSKED.BAS in diesem Heft) betrachten, werden Sie sehen, daß alle Sektoren mit E5 gefüllt, also leer sind.

Nehmen wir wieder an, daß ein File namens TEST mit der Extension COM (MC-File unter CP/M) versehentlich gelöscht wurde. In Disk-BASIC müßten wir nun wieder eine beachtliche Zahl Allocation Pointer und Tables "zurechtbiegen", in CP/M hingegen genügt es, in das Byte 0 eine Userebene im Bereich von 0 bis 15 zu schreiben. Bei unserem Beispiel würde das dann so aussehen:

```
E5 54 45 53 54 20 20 20 20 43 4F 4D

entspricht

__ T E S T         C O M ........
```

(Für die obige Darstellung sind die Regeln der letzten Folge gültig!)

Statt der Zahl E5 setzen Sie nun einfach die Userebene ein, und schon steht das File wieder auf Ab- beziehungsweise Aufruf bereit.

Zuletzt noch eine kleine Korrektur der letzten Folge, Punkt a) sollte selbstverständlich so heißen:

a) Sie haben Disk-BASIC geladen und versuchen mit "Files" das Directory einer CP/M-Diskette auszugeben.

Die Betonung liegt auf CP/M-Diskette! Entschuldigen Sie bitte dieses kleine Mißgeschick. Danke!

Ein wichtiger Baustein unserer SVI-Peripherie blieb bis jetzt unerwähnt: der Floppy Disk Controller FD 1793, mit dessen Hilfe wir erst in die Lage versetzt werden, Diskettenoperationen durchzuführen.

Der Floppy Controller FD 1793 von Western Digital ist für das reibungslose Funktionieren unserer SVI-Diskettenlaufwerke zuständig. Ob es sich um einen SVI-601-Expander mit den dafür vorgesehenen Laufwerken SVI-902 oder um die neueren Modelle der Serie SVI-605 mit einseitigen oder doppelseitigen Laufwerken handelt, überall kommt der Floppy Disk Controller FD 1793 zum Einsatz.

Bei den Expandern SVI-601 befindet er sich auf einer eigenen Floppy Controller Karte SVI-801, bei den Super-Expandern der Serie SVI-605 ist er bereits auf der Hauptplatine im Expander eingebaut.

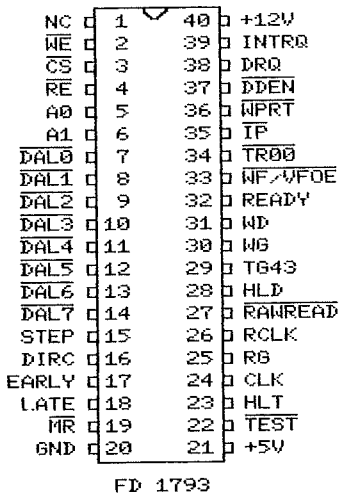
Diskettenlaufwerke können lediglich um einen Track vor- bzw. zurückfahren und eine Spur lesen bzw. beschreiben. Um dem Ganzen jedoch System zu verleihen, bedarf es eines Floppy Controllers, der jede einzelne Aktion des Laufwerks überwachen und steuern muß.

Der Aufgabenbereich des Floppy Disk Controllers bei den SVI-Computern umfaßt demnach:

- Spur 0 anfahren
  vor dem Booten notwendig
  nach Positionierungsfehlern
- eine bestimmte Spur suchen
- einen vorgegebenen Sektor lesen oder beschreiben
- beschreiben eines ganzen Tracks
  beim Formatieren erforderlich

Die Kommunikation zwischen der CPU und dem Floppy Controller findet mittels Ein-Byte-Befehlen statt. Über die erwähnten Funktionen hinaus gibt es noch zahlreiche andere Befehle, wie zum Beispiel READ TRACK, die beim SVI nicht verwendet werden.

Fehler-Bits im Statusregister des FD 1793 geben über die Korrektheit der gerade durchgeführten Operation Auskunft. Es ist Sache des Programmierers diese auszuwerten.



FD 1793

Ehe wir uns mit den Registern näher befassen, sehen wir uns die Anschlußbelegung des Bausteins an:

Pin

1 kein Anschluß

19 MASTER RESET Logisch 0 an diesem Eingang löst Initialisierung des FDC und Anfahren der Spur 0 aus

20 GND Masse

21 +5V

40 +12V

Computer-Interface:

2 WRITE ENABLE Logisch 0 erlaubt Schreiben in die durch die Adreßleitungen A0 und A1 selektierten Register

3 CHIP SELECT nur bei LOW ist Zugriff auf den FD 1793 möglich, wird durch OUT-Befehle erzeugt

4 READ ENABLE Logisch 0 ermöglicht Lesen aus den selektierten Registern

5,6 REGISTER SELECT LINES (A0 und A1)
Durch A0 und A1 erfolgt die Auswahl der Register gemäß RE- und WE-Signal:

A1	A0	RE	WE
0	0	Status-R.	Command-R.
0	1	Track-R.	Track-R.
1	0	Sektor-R.	Sektor-R.
1	1	Data-R.	Data-R.

7-14 DATA ACCESS LINES
8 Bit-bidirektionaler Bus zum Datentransfer zwischen CPU und Registern

24 CLOCK 1 MHz-Takt

38 DATA REQUEST Dieses Signal teilt der CPU mit, daß ein von einem Sektor gelesenes Datenbyte im Datenregister bereit steht. Beim Beschreiben eines Sektors fordert dieses Signal ein Datenbyte an.

39 INTERRUPT REQUEST
Dieser Ausgang teilt der CPU durch logisch 1 mit, daß eine Operation abgeschlossen ist. Wird auf 0 gesetzt, wenn das Status-Register gelesen bzw. in das Command-Register geschrieben wird.

Mit den Anschlüssen, die das Floppy Disk Interface betreffen, werden wir uns in der kommenden Folge befassen, ebenso über die Register des FD 1793.

4.2.2 Die Transient-Program-Area (TPA)

Wie der Name schon sagt, beinhaltet der auch als Benutzerspeicher bezeichnete CP/M-Abschnitt im Gegensatz zu den anderen Teilen des CP/M-Betriebssystems, "zeitlich Vorübergehendes" oder auch "Flüchtiges".

Das bezieht sich natürlich nicht auf den Netzschalter und dessen, allerdings auch für den Rest des Speichers, löschende Wirkung.

Gemeint ist hier vielmehr die Hauptaufgabe der TPA, jedem für den Mikroprozessor Z80 oder 8080 geschriebenen CP/M-Programm, dessen Daten und Stack "Arbeitsraum" zu geben, solange es vom Benutzer gewünscht wird.

Die oben angesprochenen "anderen Speicherbereiche" und die darin liegenden Teile des CP/M-Betriebssystems sind ja, wie wir wissen, für die Zeit der CP/M-Sitzung, unbeeinflußt vom Wechsel eines Programms im Benutzerspeicher immer zur gleichen Aufgabe vergattert. Also ist deren Inhalt im Gegensatz zum Benutzerspeicher nicht "transient" oder "flüchtig".

Das gilt uneingeschränkt für die Zero-Page, für das BDOS und BIOS, aber nicht für den Consol-Command-Prozessor (CCP), dessen Speicherplatz zur Vergrößerung der TPA herangezogen werden kann. Doch dazu später.

Unserer Aufforderung folgend, lädt der CCP das auf der Diskette abgelegte Programm in die TPA ab 100H und übergibt diesem mit einem "CALL 100H" die Kontrolle. Warum das mit einem "CALL" geschieht, wird bei der Beschreibung des CCP erklärt.

Um diesem nun in die TPA geladenen Programm ein uneingeschränktes Arbeiten und freies Verfügen über den ihm zugeteilten Speicherplatz zu ermöglichen, muß sichergestellt sein, daß das CP/M-Betriebssystem unmittelbar nach dem Warmstart bzw. dem Laden eines Programms in diesem Bereich weder Systemvariable noch irgendwelche Buffer betreibt. So verhält es sich auch.

Es sei denn, es wird vom Programmierer ausdrücklich gewünscht, daß unter Umgehung von außerhalb der TPA liegenden Buffern ein direkter Kontakt zwischen Programm und Betriebssystem stattfindet. Da es sich ja dann bereits um das in der TPA arbeitende Programm handelt, ist es dessen Verantwortung, ein ordnungsgemässes Lesen oder Schreiben des Betriebssystems aus bzw. in die vom Programm bereitzustellenden Bereiche im Benutzerspeicher sicherzustellen.

Das gilt vor allem für in Maschinencode zu erstellende Programme. Hochsprachen, Interpreter oder Compiler, nehmen bereits von sich aus auf die Umgangsform mit dem CP/M-Betriebssystem Rücksicht. Hier braucht sich der Programmierer ja eigentlich gar nicht mit der Richtigkeit von Betriebssystemaufrufen beschäftigen, weil das in diesem Falle das "Hochsprache-Programm" selbst erledigt.

Funktionen, wie Ausgabe eines Zeichen auf den Bildschirm oder Einlesen eines Zeichens von der Tastatur, sind Operationen mit einzelnen Zeichen und werden direkt über die vorhandenen Register der CPU durchgeführt.

Operationen, die etwas komplexere Auswirkungen nach sich ziehen, wie beispielsweise die Ausgabe einer ganzen Zeichenkette oder das Öffnen eines Files auf der Diskette, benötigen einen Direktkontakt über Speicherzellen. Schließlich ist es hier ja notwendig, mehrere Bytes an Information an das Betriebssystem zu übergeben oder von diesem zu übernehmen.

Im einen Fall ist das ein Zeiger auf einen vom Programm freigehaltenen Buffer zur Datenentnahme für das BDOS, im anderen Fall ein Zeiger auf einen Speicherbereich, z.B. ein File-Control-Block, der genau definierte Informationen in einer ganz bestimmten Reihenfolge beinhaltet.

Diese Speicherbereiche werden vom Betriebssystem nach einem Warmstart ohne Ausnahme außerhalb der TPA bereitgestellt und können eben auf Wunsch des Programmierers in die TPA verlegt werden. Diese Tatsache ermöglicht für eine große Anzahl von in Maschinencode zu schreibenden Programmen erhebliche Vereinfachungen des Programmieraufwandes und zuletzt auch beachtliche Reduzierungen der Laufzeit.

Im Folgenden werden kurz die Funktionen erläutert, die einen "Direktkontakt" zwischen dem CP/M-Betriebssystem und dem in der TPA arbeitenden Programm auslösen können.

A. Schreib- und Lesezugriffe

So ist es möglich, über einen BDOS-, aber auch einen BIOS-Aufruf den Disk-Datenbuffer (DMA) in den Speicherbereich der TPA zu verlegen, was zur Folge hat, daß ein von Diskette gelesener 128-Byte Sektor nicht wie üblich in den voreingestellten Disk-Datenbuffer bei 80H, sondern in den als neue DMA bezeichneten Speicherbereich gelesen wird. Das gleiche gilt natürlich auch für das Schreiben auf Diskette, was in beiden Fällen eine gewisse Vorsicht im Sinne eines an dieser Stelle freizuhaltenden Bufferspeichers voraussetzt.

Die oben beschriebene Anweisung verschafft den Vorteil, daß man sich angenommen bei einem sequentiellen Einlesen eines auf der Diskette befindlichen Files, durch stetiges Erhöhen und Neusetzen der in die TPA verlegten DMA um 80H, das Auslesen aller Records (128 Bytes) aus der ursprünglich voreingestellten DMA bei 80H ersparen kann.

Weiters kann man auch dem CP/M-Betriebssystem, im Zuge von Fileoperationen, mit dem entsprechenden BDOS-Aufruf mitteilen, daß sich der für Datei-Operationen notwendige File-Control-Block nicht wie vorgesehen bei 5CH, sondern irgendwo im Bereich der TPA befindet. Das hat, genauso wie beim Verlegen des Disk-Datenbuffers, ein Schreib-Lese-Zugreifen des BDOS in die TPA zur Folge.

Ein Schreib-Zugriff des Betriebssystems erfolgt auch noch durch Verlegen jenes Buffers in die TPA, der bei der BDOS-Funktion "Read Console Buffer" zur Verfügung gestellt werden muß.

B. Lesezugriffe

Die einzige Funktion, die nur einen Lesezugriff des Betriebssystem in die TPA er-

zeugt, ist der BDOS-Aufruf "Print String".

Alle anderen, vom BDOS und BIOS zur Verfügung gestellten Aufrufe, erfordern einen Datenfluß über die Register der CPU und nicht über irgendwelche Buffer. Da sie somit nie einen direkten Kontakt zur TPA und dem darin arbeitenden Programm haben können, bleiben diese Funktionen in diesem Abschnitt unbeachtet.

Für alle gerade beschriebenen Funktionen gilt, ausgenommen beim Verlegen der DMA in die TPA, daß man bei jedem Aufruf einer solchen BDOS-Funktion in Register DE der Z80-CPU die jeweilige Adresse des Zeichenbuffers oder des File-Control-Blocks angeben muß. Diese Zeiger können auf die von CP/M bereitgestellten Speicherbereiche gerichtet sein oder eben auf die TPA.

Also entscheidet der Programmierer hier, ob er einen direkten Zugriff des Betriebssystems in die TPA wünscht oder lieber alles über die vorgesehenen Buffer laufen läßt.

Eine Ausnahme bildet hier natürlich die BDOS-Funktion "Print String". Hier muß man, will man nicht übertreiben, einen Zeiger direkt auf die Zeichenkette in der TPA übermitteln, welche darauffolgend auf den Bildschirm ausgegeben werden soll.

Bevor wir zu den zwei verschiedenen Größen der TPA kommen, muß noch etwas über das Abspeichern des TPA-Inhalts auf Diskette gesagt werden.

Es war nicht immer der Fall, daß der Benutzerspeicher nach dem Ausstieg aus einem Programm und einem anschließenden Warm- oder Kaltstart hundertprozentig unverändert blieb. Das war vor allem bei den allerersten CP/M-Versionen ein großer Nachteil.

Für die nun erhältlichen, auch bei Spectravideo eingesetzten CP/M-Versionen gilt, daß der gesamte Speicherinhalt der TPA nach einem Warm- bzw. Kaltstart nicht verändert wird. Dadurch ist es möglich, durch den CCP-Befehl "SAVE" seitenweise (256 Bytes) bei 100H beginnend den Inhalt der TPA auf die Diskette zu schreiben. Der Befehl "SAVE" wird im nächsten Heft untersucht.

Das 1:1 - Speichern der TPA funktioniert aber nur mit der "normalen" Größe der TPA, also von 100H bis CFFFH. Eine durch Überschreiben oder Ignorieren des CCP, vergrößerte TPA muß aber spätestens bei einem Warmstart dem neuerlich einzulesenden CCP Platz machen. Dadurch ist es nicht mehr möglich, anstelle des ursprünglichen CCP-Bereichs gelegene Programmteile oder Daten auf die Diskette zu sichern.

Die folgenden Betrachtungen mögen dem einen oder anderen etwas merkwürdig vorkommen, da wir ja bereits wissen wie groß der uns zur Verfügung stehende Benutzerspeicher ist. Das hat zunächst auch seine Richtigkeit.

Da es aber auch andere CP/M-Rechner gibt, und wir möglicherweise beabsichtigen, das von uns erstellte Programm auch auf einem solchen laufen zu lassen, müssen wir auf die oft erheblich kleiner gehaltene TPA anderer CP/M-Computer Rücksicht nehmen.

Praktisch alle unter CP/M zu erhaltende Programme, vor allem jene, die sich unter das BDOS legen, fragen gleich bei Programmbeginn jene Speicherzellen ab, die auf die Größe der TPA schließen lassen.

Es kommt natürlich auf den Programmtyp an, ob es notwendig ist, die Höhe des "Speicher-Plafonds" des Rechners zu kennen.

Handelt es sich um kleine Programme ist die Größe des Speichers belanglos, da wir niemals in die höheren Gefielde der TPA vordringen werden müssen.

Aber für einen CP/M-BASIC-Interpreter ist es unbedingt notwendig zu wissen, wo er, bei höchstmöglicher RAM-Adresse beginnend, seine Stringvariablen usw. ablegen kann.

Beim DDT bzw. ZSID besteht der erste Schritt aus dem Berechnen des BDOS-Beginns, um sich danach darunter zu legen. Das mit dieser Aktion, die der Simulation einer freien, allerdings etwas kleineren TPA dient, der CCP überschrieben wird, geht auch aus der CP/M-Speicheraufteilung hervor (Siehe Heft 6/85 Seite 7).

In diesem Zusammenhang sei angeregt, daß bei der Absicht, die uns zur Verfügung stehende Speichergröße zu ermitteln und auch tatsächlich auszunützen, der BDOS-Beginn als Endpunkt der TPA angenommen wird.

Schließlich wird der vor dem BDOS liegende CCP nach der Übergabe der Kontrolle an das geladene Programm sicher nicht mehr gebraucht, zumal er ja gar keine Sprungleiste für irgendwelche Funktionsaufrufe bietet. Bei einem Warm- oder Kaltstart wird der CCP sowieso wieder neu in den Speicher geladen.

Will man allerdings nach dem Ablauf eines bestimmten Programms, auch die an höchster Stelle abgelegten Daten oder ähnliches mit dem CCP-Befehl "SAVE" auf die Diskette schreiben, muß man den Speicherbereich des CCP unberührt lassen, da er ja nach dem Ausstieg aus dem Programm nachgeladen wird und dort liegende Daten sicher zerstört.

Die Absicht, den ganzen Benutzerspeicher auf Diskette zu schreiben, ist sicher ein ungewöhnlicher Vorgang. Er stellt aber eine gute Methode dar, nach Ablauf eines Programms, z.B. die Datenverteilung im Speicher zu untersuchen.

Es wäre zum Beispiel denkbar, im Nachhinein mit dem Disk-Editor die dem interessanten Speicherbereich entsprechenden Sektoren sichtbar zu machen.

Der Disk-Editor empfiehlt sich deshalb, weil z.b. der ZSID Teile der TPA überschreiben würde, und somit nur eine beschränkte Untersuchung zuließe.

Hiemit kann es im Zuge der vielfältigen Möglichkeiten notwendig sein, zum Einen die Größe der TPA mit Rücksicht auf den CCP zu ermitteln, zum Zweiten die Größe der TPA um den Speicherplatz des CCP vergrößert zu errechnen.

Ersteres kann durch Einlesen der BDOS-Adresse bei Speicherstelle 0006H und dem Abzug der Länge des CCP (= 800H Bytes) durchgeführt werden. Für die Berechnung der um den CCP-Speicherbereich vergrößerten TPA genügt das Einlesen der BDOS-Adresse alleine.

Jetzt bleibt es noch dem Programmierer überlassen, ob er die sechs, vor dem BDOS-Beginn stehenden Bytes unverändert läßt, oder in seinen Speicherbedarf einbezieht. Diese verkörpern nämlich die CP/M-Versionsnummer. Diese wird aber nicht, wie man glauben könnte, beim BDOS-Aufruf "Print Version Number" benötigt.

Der nächste Beitrag führt uns zum Consol-Command-Prozessor, der zum Dank gleich wieder überschrieben wird. Aber so leicht läßt der sich nicht abwimmeln. Spätestens bei einem Warmstart ist er wieder da. Und dann werden wir ihn genau untersuchen.

```
*****************************************
*                                       *
*       ROM-Adressen für den SVI-728    *
*                                       *
*****************************************
```

In der letzten Zeit tauchen immer mehr MSX-Computer am europäischen Homecomputerhimmel auf. Auch das Angebot an Software und Literatur steigt ständig. Um diesem Trend und dem Wunsch unserer MSX-Computer besitzenden Lesergemeinde gerecht zu werden, bringen wir diesmal Informationen über wichtige ROM-Routinen und Systemvariablen, nebenbei wird auch der Befehl VDP und seine vielfältigen Anwendungsmöglichkeiten beschrieben.

Die Zahl vor jedem Informationsblock gibt an, wo Sie diese Routine im MSX-ROM finden können. In den Informationsblöcken stehen - falls notwendig - die wichtigen Übergaberegister. Es werden nur die wichtigsten Teile jeder Routine beschrieben.

Eine ausführlichere Beschreibung würde den Umfang dieses Berichts sprengen. Für die Leser, die mehr Informationen haben wollen, wird (vermutlich) bald ein ROM-Listing, wie bereits auch für den SVI-328, erhältlich sein.

Alle Adressen sind hexadezimal angegeben!

0008
Diese Routine verwendet der Interpreter, um ein Token oder Zeichen eines Befehls zu holen und es gleich auf Richtigkeit zu testen. Hinter dem Restart-Aufruf steht das Byte, das als nächstes geholt werden soll. Die Routine springt zu 10H, wenn das geholte Byte und das Byte hinter dem Aufruf identisch sind. Andernfalls wird 'Syntax error' angesprungen. Es wird zum Beispiel ein Beistrich als nächstes Zeichen gesucht, dann sieht der Aufruf folgendermaßen aus:

```
RST 8
DEFB ","
```

Der Interpreter springt auf das dem DEFB "," folgenden Byte folgende Kommando, nachdem er den Restart abgearbeitet hat.

0010
Diese Routine holt das nächste Zeichen oder Token und legt es in A ab. HL, der Programmzeiger, wird um Eins erhöht. Wenn das Zeilenende oder ein Doppelpunkt erreicht ist (Befehlsende!), dann ist das Z-Flag gesetzt.

0018
Beim Aufruf dieser Adresse wird das im Akkumulator gespeicherte Zeichen auf dem aktuellen Ausgabegerät (meistens Monitor) ausgegeben.

0020
Hier wird das Z80 Registerpaar HL mit dem Registerpaar DE verglichen. Das Carryflag wird gesetzt, wenn HL kleiner ist als DE, sonst wird es gelöscht. Bei Gleichheit der Register wird das Zeroflag gesetzt.

0D6A
Diese Routine testet, ob die Shifttaste gedrückt ist und ob der Keyboardbuffer leer ist. Wenn die Shifttaste gedrückt und die Keystatusline eingeschaltet ist, werden die Belegungen der Keys von 6 bis 10 angezeigt. Ist der Keyboardbuffer leer, wird das Zeroflag gesetzt.

10CB
Mit dieser Routine wird ein Zeichen aus dem Keyboardbuffer eingelesen. Ist dieser leer, wird auf eine Zeicheneingabe von der Tastatur her gewartet. Dieses Zeichen befindet sich dann in beiden Fällen im Akkumulator.

08BC
Diese Routine druckt ebenfalls ein im Akku gespeichertes Zeichen aus. Die waagrechte Position ist in F3DDH und die senkrechte Position in F3DCH abgelegt. Nach dem Ausdruck wird ein Linefeed ausgeführt und falls nötig auch gescrollt.

085D
Hier wird ebenfalls ein Zeichen ausgedruckt, nur ist das Ausgabegerät diesmal der Drucker. Es wird auf den Drucker gewartet! Das auszudruckende Zeichen befindet sich wieder im Akku.

0884
Hier wird der Drucker auf seine Bereitschaft getestet. Ist der Drucker bereit, so wird der Akku mit 255 geladen und das Zeroflag gelöscht, sonst wird der Akku gelöscht und das Zeroflag gesetzt.

046F
Mit dieser Routine können Sie testen, ob CTRL-STOP gedrückt wurde. Wurden diese Tasten gedrückt, wird das Carryflag gesetzt, andernfalls wird es gelöscht.

03FB
Wenn bei dieser Routine die Stoptaste gedrückt wird, legt der Interpreter eine "Ruhepause" ein, solange bis die Stoptaste nochmals gedrückt wird.

Bei CTRL-STOP wird das Programm abgebrochen und ein Return zum Interpreter ausgeführt.

0C00
Diese Routine ist mit dem BASIC-Befehl BEEP identisch.

0848
Hier wird ein CLS ausgeführt, wenn das Zeroflag gesetzt ist.

088E
Diese Routine entspricht der LOCATE-Anweisung, wobei H die waagrechte und L die senkrechte Position angibt.

0B26
Wenn (F3DEH) ungleich Null ist, wird die Funktionstastenleiste eingeschaltet

0B15
entspricht KEY OFF

0B2B
entspricht KEY ON

0F3D
Diese Routine schaltet die CAPS-LOCK-taste ein, wenn das Zeroflag nicht gesetzt ist.

0468
Mit diesem Aufruf wird der Keyboard-buffer gelöscht.

11EE
Diese Routine ist sehr gut für Video-games in Maschinencode geeignet. Die Joysticknummer muß hierbei im Akku stehen (0=Cursortasten, 1=Joystick 1, 2=Joystick 2). Der momentane Status des Joysticks wird nach der Ausführung im Akku abgelegt. Die folgende Tabelle erklärt alle möglichen (und sinnvollen) Akkumulatorinhalte:

0 = Ruhestellung
1 = nach oben
2 = nach rechts und nach oben
3 = nach rechts
4 = nach rechts und nach unten
5 = nach unten
6 = nach links und nach unten
7 = nach links
8 = nach links und nach oben

1253
entspricht STRIG, wobei der Akku die Nummer des Feuerknopfs enthält. Wurde der Feuerknopf gedrückt, so wird der Akku mit 255 geladen, sonst wird er gelöscht. Da der MSX-Standard zwei verschiedene Feuerknöpfe pro Joystickport vorsieht, gilt folgende Unterteilung:

Tastatur:
0 = SPACE-Taste

Joystickport Nummer 1:
1 = Feuerknopf 1A
3 = Feuerknopf 1B

Joystickport Nummer 2:
2 = Feuerknopf 2A
4 = Feuerknopf 2B

1A63
Diese Routine schaltet den Kassetten-recorder ein und liest den Header des nächsten Programms.

1ABC
Mit dieser Routine können Sie ein Byte vom Kassettenrecorder einlesen. Das gelesene Byte befindet sich dann im Akku.

19E9
Mit dieser Routine wird der Lesevorgang beendet und das Band gestoppt.

19F1
Diese Routine schaltet den Kassetten-recorder ein und schreibt einen Header auf die Kassette. Ist der Akku gleich 0, so wird nur ein kurzer, sonst wird ein langer Header sein. Dabei gilt, daß der kurze Header nur den Filenamen enthält. Der lange Header beinhaltet zusätzlich noch Informationen über das Programm selbst.

1A19
Diese Routine schreibt das im Akku enthaltene Byte auf eine Kassette.

19DD
Mit dieser Routine wird der Schreibvorgang beendet und das Band gestoppt.

1384
Mit dieser Routine wird der Kassetten-recordermotor ein bzw. aus geschaltet.

04BD
Diese Routine initialisiert das Sound-register (PSG). Vor dem Aufruf muß unbedingt ein "Disable Interrupt (DI, Opcode = F3H) ausgeführt werden.

1102
Mit dieser Routine ist es möglich, direkt in den Soundchip (PSG) zu schreiben (ähnlich dem Befehl SOUND). Der Akku enthält die Registernummer (des PSG!) und das Register E das zu schreibende Datenbyte. Die Registernummer ist eine Zahl von 0 bis 13.

110E
Diese Routine erlaubt es, Daten aus dem PSG zu lesen. Das PSG-Register steht wieder im Akkumulator. Nach der Ausführung steht der gelesene Wert im Akku.

11C4
Beim Aufruf dieser Adresse wird die Hintergrundmusik eingeschaltet.

0577
Mit diesem Aufruf schalten Sie den Bildschirm ab.

0570
Dieser Aufruf schaltet den Bildschirm wieder ein.

057F
Diese Routine dient zum Schreiben der Daten aus Register B in das in C enthaltene Register des VDP.

1449
entspricht VDP(8), wobei der Akku eine Kopie des VDP Statusregister enthält.

07D7
Diese Routine liest ein Byte aus dem VRAM. Das Registerpaar HL gibt die Adresse im VRAM an. Der Akku wird den Wert von HL enthalten.

07CD
Diese Routine dient zum Schreiben in das VRAM. Alle Angaben sind wie oben.

07EC
Mit dieser Routine wird das VRAM zum Lesen vorbereitet. In HL steht wieder die Adresse im VRAM.

07DF
Diese Routine bereitet das VRAM zum Schreiben vor. HL hat die Funktion wie bei der letzten Routine.

0815
Mit dieser Routine wird das VRAM mit einem einzigen Wert gefüllt. Die Start-adresse steht in HL und die Blockgröße in BC. Der zu schreibende Wert steht im Akku.

070F
Diese Routine liest einen Block aus dem VRAM ein und schreibt ihn ins RAM. HL gibt die Startadresse im VRAM und DE die Startadresse im RAM an. BC enthält wieder die Blocklänge.

0744
Ist die Umkehrung der obigen Routine, nur die Funktionen von HL und DE sind vertauscht.

084F
entspricht SCREEN, der Akku enthält den gewünschten Modus

0 = Textmodus 0
1 = Textmodus 1
2 = Grafikmodus 2
3 = Multicolormodus

07F7
entspricht COLOR, wobei folgendes gilt:

Vordergrundfarbe steht in F3E9H
Hintergrundfarbe steht in F3EAH
Randfarbe steht in F3EBH

In diese Adressen müssen Sie die gewünschte Farbe schreiben.

06A8
löscht alle Sprites.

050E
initialisiert Textmodus 0.

0538
initialisiert Textmodus 1.

05D2
initialisiert Grafikmodus 2.

061F
initialisiert Multicolormodus.

0594
schaltet Textmodus 0 ein.

05B4
schaltet Textmodus 1 ein.

0602
schaltet Grafikmodus 2 ein.

0659
schaltet Multicolormodus ein.

06E4
Diese Routine dient zum Errechnen der Adresse, die ein bestimmtes Sprite im VRAM hat. Der Akku enthält die Spritenummer. Nach der Ausführung enthält HL die Adresse im VRAM.

06F9
Diese Routine errechnet die Position eines Sprites im 'Sprite Attribute Table'. Der Akku enthält wieder die Spritenummer und HL die Adresse im VRAM.

0704
Nach Aufruf dieser Routine steht im Akku die momentane Spritegröße (Anzahl der Bytes pro Sprite).

1510
Mit dieser Routine wird ein Zeichen an der aktuellen Position des Grafikcursors ausgegeben. Das Zeichen befindet sich im Akku.

1599
Diese Routine überprüft, ob die Koordinaten innerhalb des darstellbaren Rahmens liegen. Die X-Koordinate ist in BC, die Y-Koordinate in DE gespeichert. Wenn die Koordinaten im zugelassenen Bereich liegen, wird das Carryflag gesetzt.

15DF
Die Koordinaten aus BC und DE werden in die Zeichenkoordinaten umgerechnet, das Ergebnis ist eine Maske (F92AH) und eine Adresse (F92CH).

1639
Hiermit wird das obige Ergebnis nach HL und in den Akku geladen (F92CH nach HL und F92AH in den Akku).

1640
Diese Routine speichert HL und den Akku als Zeichenkoordinaten.

1676
Mit dieser Routine setzen Sie die Farbe für die Routine von (167EH). Diese Farbe wird in F3F2H gespeichert, wenn sie kleiner als 15 ist.

1647
Diese Routine liest die Farbe des Punktes, auf dem der Grafikcursor steht. Dieser Wert wird im Akku gespeichert.

167E
Diese Routine dient zum Setzen eines Punktes. Die Farbe steht in F3F2H, die Zeichenkoordinaten in F92CH und F92AH. Benutzen Sie die Routine 15DFH, um die Zeichenkoordinaten zu berechnen.

16C5
Diese Routine bewegt den Grafikcursor um ein Pixel nach rechts.

16EE
wie zuvor, allerdings nach links.

175D
wie zuvor, allerdings nach oben.

173C
wie zuvor nach oben, aber mit Bereichstest.

172A
wie zuvor, allerdings nach unten.

170A
wie zuvor nach unten, aber mit Bereichstest.

1809
Durch Aufruf dieser Routine wird eine Anzahl von Pixel, die durch HL festgelegt ist, vom aktuellen Grafikcursor aus nach rechts gesetzt.

18CF
Mit dieser Routine wird PAINT initialisiert. Der Akku enthält die Randfarbe des Abschnittes, der ausgemalt werden soll.

Die MSX-Computer und der SVI-328 sind mit dem TMS 9929A von Texas Instruments ausgerüstet. Dieser Videochip besitzt 8 Register, von denen man in die ersten 7 nur schreiben kann. Bei den MSX-Computern wurde das geändert. Es wurde ein eigener Befehl entwickelt, um diese Register auch lesen zu können. Eigentlich liest der Befehl VDP aber nicht die Register, sondern deren Kopien aus dem System-RAM. Diese Kopien befinden sich im RAM von F3E0H bis F3E6H. Da das achte Register auch ohne Kopie gelesen werden kann, sind nur Kopien für die ersten 7 Register vorhanden.

Die Syntax der VDP-Anweisung:

'Variable'=VDP('Register')

Sehen wir uns nun die Bedeutungen der einzelnen Register an:

Register 0:

Bei diesem Register darf kein Bit außer dem Bit 0 und dem Bit 1 gesetzt sein. Wenn das Bit 0 gesetzt ist, so gibt es noch einen zweiten, externen Videochip. Aus Bit 1 wird zusammen mit den Bits 3 und 4 des Registers 1 der momentane Screenmodus berechnet. Nennen wir dieses Bit 1 S/A.

Register 1:

Hier darf Bit 2 nicht gesetzt sein. Bit 0 gibt die Auflösung der Spritedarstellung (0 = normale, 1 = vergrößerte Darstellung) und Bit 1 die Größe der Sprites (0 = 8x8, 1= 16x16) an. Bit 3 nennen wir S/B und Bit 4 S/C. Mit der folgenden Tabelle können Sie beziehungsweise der Computer dann den aktuellen Screenmodus erechnen.

S/C	S/B	S/A	
0	0	0	Textmodus 1
0	0	1	Grafikmodus 2
0	1	0	Multicolormodus
1	0	0	Textmodus 0

Wenn Bit 5 gesetzt ist, sind Videochipinterrupts möglich. Bei gesetztem Bit 6 ist die Bildschirmanzeige eingeschaltet (ähnlich der Routine bei 41H). Bit 7 gibt die Größe des VRAM an (0= 4Kbyte, 1 = 16Kbyte).

Bei den folgenden Registern geben die Zahlen in den Klammern den möglichen Bereich des Inhalts an.

Register 2: (0-15)

Der Inhalt von Register 2 gibt bei der Multiplikation mit 400H die 'Name Table'-Adresse an.

Register 3: (0-255)

Wenn Sie den Inhalt dieses Registers mit 40H multiplizieren, kommt die 'Color Table'-Adresse heraus.

Register 4: (0-7)

Der Inhalt von Register 4 - mit 800H multipliziert - ergibt die Patternnummer-, Text-, Multicolor-Generator Adresse.

Register 5: (0-127)

Der Inhalt dieses Registers - mit 80H multipliziert - ergibt die 'Sprite Attribute Table'-Adresse.

Register 6: (0-127)

Wenn Sie den Inhalt dieses Registers mit 800H multiplizieren, ist das Ergebnis gleich der 'Sprite Pattern'-Adresse.

Register 7: (0-15)

Die ersten 4 Bits - mit 10H multipliziert - ergeben die Vordergrundfarbe. Die zweiten vier Bits geben die Hintergrundfarbe an.

Das achte Register kann nur gelesen werden. Es wird übrigens auch das Statusregister genannt.

Register 8:

Bit 7 zeigt einen VDP-Reset an, Bit 6 gibt an, ob ein fünftes Sprite in einer Zeile vorhanden ist, und die Bits 0 bis 4 geben die Nummer des fünften Sprites an (selbstverständlich nur, wenn es ein fünftes Sprite in einer Zeile gibt!). Ein gesetztes Bit 5 zeigt eine Kollision von zwei oder mehreren Sprites an.

Dieses Register wird alle 20 Millisekunden erneuert.

Mit dieser Beschreibung sollten auch die letzten Unklarheiten über die Registerbedeutungen des Videochips beseitigt worden sein.

Neben dem VDP-Kommando gibt es noch die BASE-Funktion. Sie gibt dem Bediener die einzelnen Startadressen der Tabellen im Video-RAM an. Welche Tabelle man gerade wissen möchte, gibt eine Zahl hinter dem Befehlswort an. Die Syntax von BASE:

'Variable' = BASE('Zahl')

Die 'Zahl' muß zwischen 0 und 19 sein. Die einzelnen Werte geben folgendes an:

0: Startadresse des 'Name Table' im Textmodus 1

1: keine Belegung

2: Startadresse des 'Pattern Table' im Textmodus 1

3: keine Belegung

4: keine Belegung

5: Startadresse des 'Name Table' im Textmodus 2

6: Startadresse des 'Color Table' im Textmodus 2

7: Startadresse des 'Pattern Table' im Textmodus 2

8: Startadresse des 'Sprite Data Table' im Textmodus 2

9: Startadresse des 'Sprite Pattern Table' im Textmodus 2

10: Startadresse des 'Name Table' im Graphikmodus 1

11: Startadresse des 'Color Table' im Graphikmodus 1

12: Startadresse des 'Pattern Table' im Graphikmodus 1

13: Startadresse des 'Sprite Data Table' im Graphikmodus 1

14: Startadresse des 'Sprite Pattern Table' im Graphikmodus 1

15: Startadresse des 'Name Table' im Graphikmodus 2

16: Startadresse des 'Color Table' im Graphikmodus 2

17: Startadresse des 'Pattern Table' im Graphikmodus 2

18: Startadresse des 'Sprite Data Table' im Graphikmodus 2

19: Startadresse des 'Sprite Pattern Table' im Graphikmodus 2

'Name Table' ist die Tabelle, die die Kennungen der einzelnen Byteblöcke für den Bildschirm beinhaltet.

'Color Table' enthält für jedes Byte die Farbcodes.

'Pattern Table' enthält das Muster des Bildschirms in einzelnen Bytes.

'Sprite Data Table' enthält die Koordinaten, die Farben und die Nummern der angezeigten Sprites für alle Spriteebenen.

'Sprite Pattern Table' enthält die Muster der einzelnen Sprites.

Wenn Ihnen noch weitere nützliche Romroutinen bekannt sind, dann teilen Sie uns diese bitte mit, damit wir noch mehr Informationen über das MSX-ROM an unsere Leser weitergeben können. Auch die Redaktion wird sich weiter bemühen, neue ROM-Routinen zu finden.

Diskeditor in BASIC

In den Berichten über das Directory bei CP/M und bei Disk-BASIC wurde öfters auf die Verwendung eines Diskeditors verwiesen. Da aber noch nicht alle Clubmitglieder oder SVI-Journal-Leser einen Diskeditor besitzen, schrieb Herr Hollinetz einen Diskeditor in Disk-BASIC.

Dieser Diskeditor soll kein Ersatz für das Programm DSKED.COM sein, das unter CP/M läuft, trotzdem besitzt er nahezu alle Funktionen des oben erwähnten Programms.

Diese Funktionen werden in sechs Teile geteilt.

DIRECTORY:

In diesem Modus wird ein beliebiger Sektor des Directorytracks (Track 20) in folgender Form gelistet:

Filename Attribute Pointer

Unter dem Namen Attribute wird die Fileart (MC-, BASIC-, ASCII-, oder Bildschirmdatei) und die gesetzten Attribute in binärer Schreibweise angezeigt. Pointer (für Allocation Table Pointer) gibt die Nummer des ersten Tracks an, in dem die Datei abgespeichert ist.

Die genaue Beschreibung des Allocation Tables finden Sie im vorigen Heft.

Bei der Eingabe von Sektor und Track müssen Sie in diesem Modus immer die Zahl 20 auf die Frage nach dem Track eingeben. Ist in dem von Ihnen bestimmten Directorysektor kein File verzeichnet, so werden statt eines Filenamens nur acht invers angezeigte Blanks ausgegeben.

HEX-LIST:

In diesem Modus wird ein Hexdump des vorher angegebenen Tracks und Sektors angezeigt. In der obersten Zeile befinden sich dabei die Zahlen von 00 bis 0F (hexadezimal). Senkrecht werden die Teile des Sektors mit :00 bis :F0 nummeriert. Diese Angaben sind wichtig für den eingebauten "Bildschirmeditor". Die Regeln für die Verwendung des Editors können Sie nach der Modi-Beschreibung lesen.

ASCII:

Dieser Modus funktioniert ähnlich dem HEX-LIST-Modus, nur fehlt die waagrechte Nummerierung der Datenbytes. Senkrecht werden die Teile des Sektors mit ;00 bis ;F0 nummeriert, sonst gelten diesselben Regeln wie oben, mit einer Ausnahme: Natürlich ist die Ausgabe dieses Modes nicht HEX- sondern ASCII-Code.

COPY:

In diesem Modus können Sie einzelne Sektoren beliebig auf der Diskette 'herumkopieren'. Dieser Modus hat eine ähnliche Funktion und Syntax wie der Befehl COPY unter Disk-BASIC.

FILES:

Dieser Modus zeigt schlicht und einfach die auf der Diskette vorhandenen Files an.

CP/M:

In diesem Modus können Sie das CP/M-Directory mit allen wichtigen Angaben auflisten. "User" gibt selbstverständlich den Userbereich an. Wenn hier "--" steht, ist das File gelöscht, und der Filename wird invers angezeigt. Ist das File schreibgeschützt, so wird die Extension des Filenamen invers angezeigt. "FS" gibt die Nummer des Fortsetzungtracks an. Bei "REC" steht die Anzahl der benutzten Records (Sektoren). Die danach folgende Zahlenschlange gibt, wie im Bericht über das CP/M-Directory bereits erwähnt, die genaue Lage der Datenblöcke an.

"BILDSCHIRMEDITOR":

Das Wort Bildschirmeditor ist, zugegeben, etwas übertrieben, dennoch sind die Funktionen im Verhältnis zur Länge des Programms sehr benutzerfreundlich.

Bei der Verwendung des Editors müssen Sie beachten, daß Sie nicht aus dem vorgegebenen Bereich hinausschreiben. Sie dürfen auch nicht die senkrechte Nummerierung der Sektorteile und das nachfolgende Blank überschreiben, da dies die Kennzeichen für die Einordnung auf der Diskette sind. Um den Editiervorgang zu beenden, fahren Sie den Cursor in die erste Zeile und drücken "ENTER".

Die verschieden Attribute bedeuten:

Attribut (binär)	Filestyp
00000000	ASCII
00000001	MC
10000000	BASIC
10100000	Bildschirm

Variablenübersicht:

```
CT, T ..... Track bzw. Copytrack
CS, S ..... Sektor bzw. Copysektor
M ......... enhält akt. Modusnummer
M$ ........ akt. Modusbezeichnung
L$ ........ löscht Bilds. bis Ende
E$ ........ entspricht ESC=chr$(27)
I$,O$ ..... Invers on / Invers off
MM$, A$, S$ ..... Standardabfragevariablen
A$(0), A$(1) ..... akt. Sektoreninhalt
A, B, C ..... Laufvariablen
X$, B$(0), B$(1),  Sektoreninh. für versch.
EX$ ....... Funktionen
```

Programmteile:

```
110 - 135 ..... Voreinstellung
140 - 155 ..... Track & Sektorabfrage
160 ........... Jump zu akt. Modus
165 - 220 ..... Hex-listroutine
225 - 280 ..... Directoryroutine
285 - 315 ..... Copyroutine
320 - 330 ..... Filesroutine
335 - 390 ..... ASCII-Dump
400 - 425 ..... Änderungen ? / Modus ?
430 - 470 ..... Modusnr. & -bezeichnung
475 - 550 ..... "Bildschirmeditor"
555 - 570 ..... Errormeldung
575 - 675 ..... CP/M-Directory
```

Mit der obigen Übersicht sollte es Ihnen leicht fallen, Veränderungen im Programm vorzunehmen. Eine dieser Veränderungen könnte zum Beispiel die Anpassung an CP/M-Disketten sein. Wenn Sie DSKED.BAS für CP/M verwenden wollen, müssen Sie nur die FOR-NEXT-Schleife (mit Variable B) streichen. Sollten Sie noch Wert auf die Funktionen FILES und DIRECTORY legen, erwarten Sie einige weitere Veränderungsarbeiten. Vergessen Sie nicht, daß sich das CP/M-Directory auf einem anderen Track befindet und eine vollkommen andere Struktur besitzt als das Disk-BASIC-Directory.

Beachten Sie vor der Anpassung auch noch die Hinweise aus dem Bericht "Das Directory unter CP/M" in diesem Heft!

```
100 '****************************************
101 '*                                      *
102 '*          D S K E D . B A S           *
103 '*     (C) Copyright by W.Hollinetz     *
104 '*      edited,cor. & ext. by C.S.       *
105 '*                                      *
106 '****************************************
107 '
110 SCREEN0,1:CLEAR 2000
115 KEY1,"1:DIREC":KEY2,"2:HDUMP":KEY3,"3:AS
CII":KEY4,"4:COPY":KEY5,"5:FILES"
120 L$=CHR$(27)+"J":E$=CHR$(27)
125 I$=E$+"p":O$=E$+"q"
130 FIELD#0,128 AS A$(0),128 AS A$(1)
135 S = 2:GOSUB 435
140 PRINTCHR$(12)I$M$O$TAB(12);:INPUT"Sektor
 ";S
145 IF S > 17 OR S < 1 THEN PRINT CHR$(30);:
GOTO 140
150 PRINT TAB(12);:INPUT"Track  ";T
155 IF M <>4 THEN X$ = DSKI$(1,T,S):B$(0) =
A$(0):B$(1) = A$(1)
160 ON M GOTO 225,165,335,285,320,575
165 LOCATE,,0
170 PRINTL$CHR$(10)"    0 1 2 3 4 5 6 7 8 9
A B C D E F"
175 FOR A=0 TO 1
180 FOR B=1 TO 128
185 IF ((B-1)AND15) = 0 THEN PRINT:PRINT":"R
IGHT$("0"+HEX$(A*128+B-1),2)" ";
190 PRINTUSING"ôô";RIGHT$("0"+HEX$(ASC(MID$(
A$(A),B,1))),2);
195 IF INKEY$<>"" THENLOCATE,,1 ELSE 205
200 GOSUB 400:GOTO 220
205 NEXT B,A
210 LOCATE,,1
215 GOSUB 400
220 LOCATE0,5:GOSUB 475:GOTO 140
225 IF T <> 20 OR S > 13 THEN 140
230 IF S=14 THEN PRINT L$A$(0)A$(1):GOTO 140
235 A$ = "0000000":PRINT L$:PRINT
240 FOR A=0 TO 1
245 FOR B=1 TO 128 STEP 16
250 C = ASC(MID$(A$(A),B,1))
255 IF C=0 OR C=255 THEN PRINTI$" "MID$(A$(A
),B+1,8)O$;:IF C=0THEN 270 ELSE 275
260 PRINT MID$(A$(A),B,9)TAB(12);
265 PRINT"Attr :";RIGHT$(A$+BIN$(ASC(MID$(A$
(A),B+9,1))),8);
270 PRINT TAB(27)"Pointer :"RIGHT$("0"+HEX$(
ASC(MID$(A$(A),B+10,1))),2);
275 PRINT
280 NEXT B,A:GOSUB 400:GOTO 140
285 CT = T:CS = S
290 PRINT L$"nach"TAB(12);
295 INPUT"Sektor ";CS:PRINT TAB(12);
300 INPUT"Track  ";CT
305 DSKO$ 1,CT,CS
310 GOSUB 400
315 GOTO 140
320 PRINT L$:FILES
325 GOSUB 400
330 GOTO 140
335 PRINT L$:PRINT
340 LOCATE,,0
345 FOR A=0 TO 1
350 FOR B=1 TO 128
355 IF ((B-1)AND15)=0 THEN PRINT:PRINT";"RIG
HT$("0"+HEX$(B-1+A*128),2)" ";
360 C = ASC(MID$(A$(A),B,1))
365 IF C<32 OR C>126 THEN A$ ="_" ELSE A$ =
CHR$(C)
370 IF INKEY$<>""THENLOCATE,,1 ELSE 380
375 GOSUB 400:GOTO 390
380 PRINT A$;:NEXT B,A:LOCATE,,1
385 GOSUB 400
390 LOCATE0,5:GOSUB 475:GOTO 140
395 END
400 PRINT:PRINT:PRINT"Aenderung (J/N/M) ? ";
:A$=INPUT$(1)
405 IF A$ = "n" OR A$ = "N" THEN 140
410 IF A$="j" OR A$="J" THEN RETURN
415 IF A$=CHR$(27) THEN CLS:FILES:DEFUSR=&H5
9:A=USR(0):END
420 IF A$="M" OR A$="m" THEN PRINT CHR$(30):
PRINT L$"Mode (1-6) ? ";
425 MM$=INPUT$(1)
430 S=VAL(MM$):IF S<0 OR S>6 THEN 420
435 ON S GOSUB 445,450,455,460,465,470
440 GOTO 140
445 M$="Directory":M=1:RETURN
450 M$="Hex-list ":M=2:RETURN
455 M$="ASCII    ":M=3:RETURN
460 M$="Copy     ":M=4:RETURN
465 M$="Files    ":M=5:RETURN
470 M$="CP/M     ":M=6:RETURN
475 LOCATE,,1:LINEINPUT A$:LOCATE,,0
480 IF LEFT$(A$,1)=":" THEN 525
485 IF LEFT$(A$,1)<>";" OR LEN(A$)<>20 OR MI
D$(A$,4,1)<>" "THEN 555
490 A = VAL("&h"+MID$(A$,2,2))
495 B = -1*(A>127):A = A AND 127
500 FOR C=A TO A + 15
505 C$ = MID$(A$,5+C-A,1)
510 IF C$ <> "_" THEN MID$(A$(B),C+1,1) = C$
515 NEXT C:LOCATE,,1
520 IF (A + B) = 0 THEN DSKO$ 1,T,S:RETURN
ELSE 475
525 IF LEN(A$)<>36 OR MID$(A$,4,1)<>" " OR I
NSTR(5,A$," ")<>0 THEN 555
530 A = VAL("&h"+MID$(A$,2,2))
535 B = -1*(A>127):A=AAND127:FORC=ATOA+15
540 MID$(A$(B),C+1,1) = CHR$(VAL("&h"+MID$(A
$,5+(C-A)*2,2)))
545 NEXT C:LOCATE,,1
550 IF (A + B) = 0 THEN DSKO$ 1,T,S:RETURN
ELSE 475
555 LOCATE0,22,1:PRINTCHR$(5)" ERROR";:BEEP:
RETURN
560 PRINT CHR$(5)I$" E R R O R "
565 PRINT O$
570 BEEP:RETURN
575 IF T<>3 OR S>8 THEN 140
580 FOR A=0 TO 1
585 FOR B=1 TO 128 STEP 32
590 PRINT:C = ASC(MID$(A$(A),B,1))
595 PRINT"User:";
600 IF C=&HE5 THEN PRINT"-- "; ELSE PRINTUSI
NG"##";C;
605 IF C=&HE5 THEN PRINTI$MID$(A$(A),B+1,8)O
$; ELSE PRINT" ";MID$(A$(A),B+1,8);
610 PRINT".";
615 C=ASC(MID$(A$(A),B+9)):EX$=MID$(A$(A),B+
10,2)
620 IF (C AND 128)<>128 THEN PRINTCHR$(C)EX$
; ELSE PRINTI$CHR$(C-128)EX$O$;
625 PRINT" FS:";:PRINTUSING"###";ASC(MID$(A$
(A),B+12));
630 PRINT" REC:";:PRINTUSING"###";ASC(MID$(A
$(A),B+15))
635 EX$=MID$(A$(A),B+16,16)
640 PRINTTAB(5);
645 FOR I=1 TO 16
650 PRINTUSING"ôô";"0"+HEX$(ASC(MID$(EX$,I))
);
655 NEXT
660 PRINT:NEXT:PRINT
665 IF A=0 THEN S$=INPUT$(1):LOCATE0,2:PRINT
L$CHR$(30):NEXT
670 NEXT
675 GOSUB 400:GOTO 140
```

ROM-Routinen für den SVI-328

In dieser und in den nächsten Ausgaben des SVI-Journals werde ich Sie über einige der wichtigsten ROM-Routinen informieren, die es in unserem Interpreter gibt.

Ein Großteil der wichtigsten Routinen kann über die Sprungtabelle zwischen &H38 und &H180 aufgerufen werden. Deshalb werde ich ich Ihnen lediglich die Sprünge in der Tabelle erklären.

0038: Interruptroutine. Sie liest bis zu 11 gleichzeitig gedrückte Tasten ein, überprüft auf ON KEY, ON SPRITE etc. und erhöht TIME immer um 1.
003B: Diese Routine zeigt bei SHIFT die Funktionstasten 6-10 an.
003E: Diese Routine wartet auf ein Zeichen von der Tastatur. Das Zeichen steht beim Rücksprung im Register A.
0041: Drucker bereit ? Wenn nicht, ist das Zeroflag gesetzt. AF wird verändert.
0044: Gibt A an den Drucker aus, wartet eventuell auf Drucker bereit.
0047: Textschirm SCREEN 0 einstellen.
004A: SCREEN 1 einstellen.
004D: SCREEN 2 einstellen.
0050: Funktionstastenzeile ausgeben, wenn SCREEN 0,x für x<>0 gilt. 0056 ist die Ausgaberoutine dazu.
0053: schaltet die Funktionstastenanzeige aus.
0056: schaltet obiges wieder ein.
0059: Originalfunktionstastenbelegung herstellen.
005C: CTRL-STOP Test, wenn ja, dann Carry gesetzt und Tastaturbuffer gelöscht.
005F: Gibt B an PSG-Register 15 aus und springt nach HL.
0062: wie 005F, doch wird HL aufgerufen und kehrt dann wieder zurück.
0065: Z80-Befehl NOP, keine Funktion.
0066: NMI-Einsprung, nicht verwendet.
0069: druckt 'Press Play on tape' und wartet darauf.
006C: liest A vom Band. Bei CTRL-STOP oder Cassette kommt es zu 'Device I/O error'.
006F: schaltet Motor aus.
0072: druckt 'Press Play and Record ..' und wartet darauf.
0075: schreibt A auf das Band, Errorbehandlung wie 006C.
0078: schreibt 0 aufs Band und geht dann zu 006F.
007B: Ausgabe von CR (CHR$(13)) und LF (CHR$(10)) an Bildschirm oder Drucker.
007E: Wenn der Cursor des Ausgabegerätes nicht bei 0 steht, wird 007B aufgerufen (CR LF ausgeben).
0081: wandelt CHR$(9) = TAB in entsprechende Anzahl von Blanks um und gibt sie aus.
0084: reserviert Platz für einen String mit der Länge A. DE enthält Anfangsadresse der Zeichenkette, HL enthält die Adresse der aktuellen Stringlänge.
0087: Schreibt nach einer Stringoperation ohne Längenänderung den String wieder an die gleiche Position in den Stringspeicher.
008A: Wenn der String, dessen Adresse im BASIC-Akku steht, der letzte abgelegte war, wird dervon ihm belegte Platz freigegeben.
008D: Ausdrucken einer Zeichenkette ab HL, das Ende muß mit 0 gekennzeichnet sein.
0090: Stackrestaurierung, Aufruf 009C, Aufruf der Warmstartroutine.
0093: erzeugt 'Syntax error'.
0096: wie ERROR-Befehl, Fehlernummer in E.
0099: Fehlertext suchen, HL zeigt auf Textanfang und E gibt Fehlernummer an.
009C: Direktmodus, Textmodus wird eventuell eingeschaltet, und 'Ok' ausgegeben.
00A2: Ausführen einer BASIC-Anweisung, HL muß auf 0 oder ':' vor Token stehen, da es um 1 erhöht wird.
00A5: erzeugt 'Illegal function call'.
00A8: löscht Druckerlaubnis und führt eventuell CR-LF aus.
00AB: Umwandlung der Funktionstokens in Adressen, Funtionsargument in BASIC-Akku übergeben und auf '(' und ')' prüfen.
00AE: Rechenroutine, hier werden alle Berechnungen durchgeführt. HL zeigt auf das erste Byte des Ausdruckes.
00C0: Codieren einer Eingabezeile in die BASIC-Tokens. Anfang der Eingabezeile wird durch HL adressiert, die codierte Zeile ist ab der Adresse &HF54F abgelegt.
00C3: neue Zeilenadressen für alle Zeilen nach der Zeile, die mit DE adressiert wird (Wichtig für das Einfügen von BASIC-Zeilen).
00C6: aus Akku wird Integerzahl, im BASIC-Akku abgelegt, HL enthält Zahl.
00C9: HL in Single-Zahl, wird im BASIC-Akku abgelegt.
00CC: Trägt HL als Integerzahl in den BASIC-Akku ein.
00CF: 'Device I/O error'.
00D2: POP AF, BC, DE, HL und RET.
00D5: Berechnet Fileende (EOF) für die Dateinummer im BASIC-Akku.
00D8: Holt Programmzeiger HL vom Stapel und RET.
00DB: Drucke HL dezimal aus.
00DE: 'Out of memory'.
00E1: Rückstellen aller Zeiger, führt DEFDBL A-Z durch, schließt alle Files, löscht aber nicht das Programm.
00E4: Dateinamen aus Tokenzeile - adressiert durch HL - erzeugen, Name wird ab der Adresse &hF99E abgelegt, die Gerätenummer wird im Akkumulator übergeben.
00E7: LD A,(HL), INC HL, DEC E, RET
00EA: BASIC-Akku zu Integerzahl wandeln, Lowbyte nach A, Aufruf der 00ED-Routine.
00ED: in A wird Dateinummer übergeben, in HL wird Pufferadresse abgelegt, A enthält nachher Gerätenummer.
00F0: Überprüft, ob Kanalnummer in E geöffnet worden ist.
00F3: Datei mit Nummer A wird geöffnet, Gerätenummer in D, (F99EH) enthält Filenamen.
00F6: Schließt Kanal.
00F9: Puffer wird gelöscht, Modus rückgesetzt (auf Null), Flag = 0, Dateinummer = 0, (F997H) als Parameter gebraucht. Aufruf durch CALL adr.
adr: PUSH HL, JP 00F9H.
00FC: schließt alle Dateien, solange nicht (F9B2H)<>0 gilt, F9B2H bestimmt, ob ein Programm geladen wird (<>0) oder nicht (=0).
00FF: Gibt A auf Kanal aus.
0102: Liest A von Kanal ein.
0105: Dateibuffer, der durch F997H adressiert wird, löschen.
0108: ab HL wird Pufferadresse abgelegt, A enthält nachher Gerätenummer.

010B: 'File already open'.
010E: 'Bad file name'.
0111: 'File already open'.
0114: 'File not found'.
0117: 'File not OPEN'.
011A: 'Internal error'.
011D: 'Input past end'.
0120: Wandelt eventuellen Kleinbuchstaben in A in Großbuchstaben um. Ergebnis in A.
0123: Bedingungen: HL zeigt auf Tokentabelle (295H), A enthält Anweisung (1. Zeichen), Stack enthält Zeiger auf Anweisung (1.Zeichen), Tokenberechnung, wenn gefunden, wird es eingesetzt.
0126: Akku in (HL)-Speicherzelle laden, (HL) befindet sich in einer anderen Bank. B enthält gewünschte Bank, C die eigene, Port 88H muß 15 enthalten, Interrupts müssen gesperrt sein.
0129: analog zu 0126, nur wird A aus (HL) geladen ('LD A,(HL)').
012C: Gerätenummer zu (F997H) nach A holen.
012F: Programm löschen und 'Out of memory'-Error.
0132: Wenn BASIC-Stapel (durch CLEAR verstellbar) < HL, dann 012F, sonst Return.
0135: Daten des aktuellen Dateipuffers herstellen.
0138: 'Field overflow'.
013B: wie 00E4, Dateiname wird nicht aus Tokenzeile gebildet sondern ist der zuletzt benutzte String.
013E: 'Sequential after PUT'.
0141: wie 00F0, jedoch ohne Einlesen der Dateinummer aus dem Programm.
0144: HL wird mit Adresse des Anfangs des Dateipuffers geladen und auf die Daten eingestellt.
0147: Erzeugt BEEPser.
014A: Wandelt den Code des ASCII-Zeichens in C in den Code des entsprechenden Videozeichens um. Ergebnis in C.
014D: Zeilenanzahl nach A laden (23 oder 24).
0150: Gibt den Status der Zeile L in A zurück. A=0 bedeuted, daß die Zeile in die nächste Zeile weitergeht.
0153: Wie 014A, jedoch enthält A den Code des Videozeichens und danach den ASCII-Code.
0156: Schreibt A in das Attributbyte der aktuellen Zeile.
0159: Wie 0156, A wird vorher mit &HAF geladen.
015C: Löscht Filezeiger (&HF997) und Druckerlaubnis (&HF542).
015F: RST 10H, dann wie 00B1
0162: Integerzahl aus Tokenzeile - adressiert durch HL - berechnen, wenn nicht positiv oder keine Integerzahl möglich, 'Illegal function call'
0165: Ausführen der eingegebenen Zeile, HL muß auf &HF68D zeigen, Zeile selber beginnt bei &HF68E.
0168: Test auf String im BASIC-Akku, bei 'falsch' 'Type mismatch' ausgeben.
016B: A Bytes im Stringspeicher reservieren, bei zu wenig Platz Speicherbereinigung (wie FRE("")) durchführen, wenn noch immer zu wenig Platz, 'Out of string space' ausgeben.
016E: CTRL-STOP-Test, wenn CTRL-STOP gedrückt, Abbruch und Rückkehr in den Direktmodus. ON STOP GOSUB wird berücksichtigt.
0171: NEW.
0174: CINT.
0177: POP DE,HL,BC und RET.
017A: VARPTR.
017D: POP HL,XOR A und dann &H00F6.

Mit dieser Tabelle besitzen Sie die meisten wichtigen Routinen des BASIC-Interpreters. Wir haben der Vollständigkeit halber die ganze Sprungleiste veröffentlicht, obwohl sich einige Sprünge aus anderen zusammensetzen oder sich sehr gleichen. Andere ROM-Routinen werden noch veröffentlicht.

```
*****************************************
*                                       *
*          3D-Graphik-Beispiele         *
*                                       *
*****************************************
```

Von einem Clubmitglied wurden uns einige schöne Beispiele von Ebenen zugesandt. Bevor wir mit der Theorie der Parameterdarstellung und Zylinderkoordinaten weitermachen, wollen wir eine Folge 'Praxis' einschieben. Hinter jedem Bild sind wie immer die Formel und die Eingabeparameter angegeben. Die drei Ebenen kann man natürlich auch durch geschicktes Verändern der Koordinaten in anderen Winkeln oder Positionen betrachten. Sehr Experimentierfreudige können auch versuchen, andere Formelelemente zusätzlich einzufügen. Vielleicht gelingt es jemandem, die Wellen in Bild 1 in der Mitte hochzuziehen und gegen den Bildschirmrand abzuflachen.

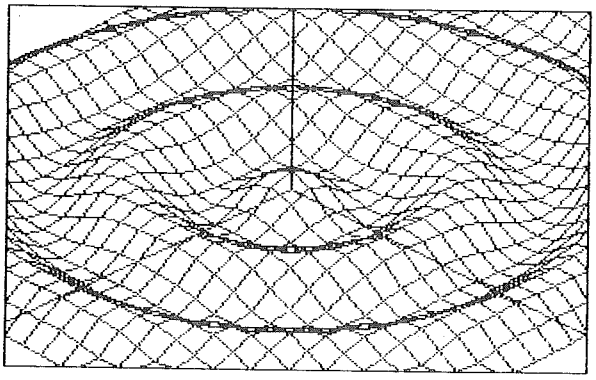


Bild 1: Z=COS(SQR(X*X+Y*Y)))
125,96,30,30,10,1,1,1

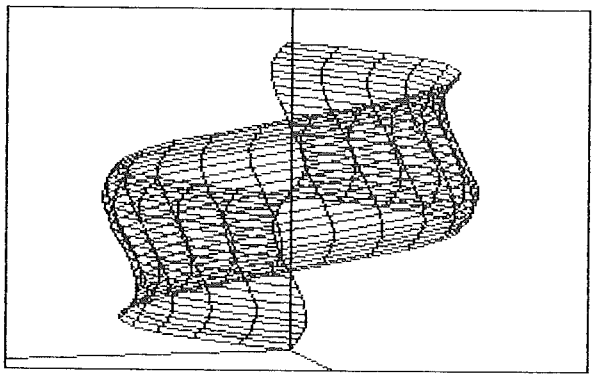


Bild 2: X=3*SIN(U):Y=SIN(V):Z=(U+V)/2
125,180,3,30,7,1,3,0,6.3,.3,0,6.3,.3

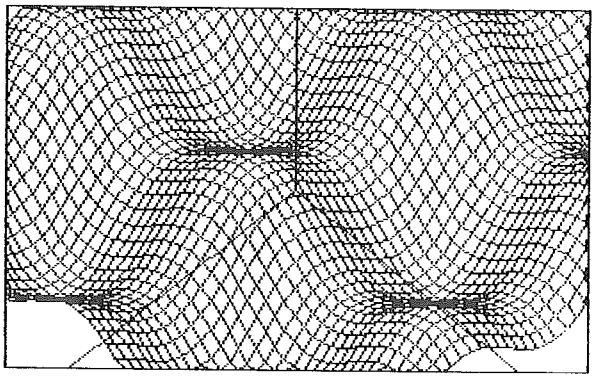


Bild 3: Z=3*SIN(X)+3*COS(Y)
127,96,44,44,20,1,1,1,.314159256

Kürzere Nachlaufzeiten für die Diskettenlaufwerke

Ich selbst habe mich schon oft über die lange Nachlaufzeit der Diskettenlaufwerke geärgert. Aus diesem Grund habe ich nach einer Möglichkeit gesucht, die Nachlaufzeiten zu verkürzen - und gefunden.

Eine Nachfrage bei unserem CP/M-Spezalisten Herrn Traxler ergab nämlich, daß eine Änderung auf eine kürzere Nachlaufzeit leicht selbst durchführbar ist. Dafür ist nämlich nur ein 2 Byte langes Wort verantwortlich. Dies sowohl unter CP/M als auch unter Disk-BASIC.

Eine Änderung ist allerdings direkt auf der Diskette auf den Systemspuren durchzuführen und kann erst danach mittels SYSGEN auch auf andere Disketten übertragen werden. Als Adressen auf der Diskette konnte ich sodann folgende Werte ermitteln (128 Bytes pro Sektor):

	Spur	Sektor	Byte
BASIC	0	23	13H
CP/M 2.20	0	33	5DH
CP/M 2.22	1	1	39H
CP/M 2.23	1	1	48H

Diese Adressen enthalten jeweils den Wert 0708H in der Reihenfolge LOW-Byte HIGH-Byte. Durch Veränderung dieses Wertes kann nun eine beliebige Nachlaufzeit eingestellt werden. Diese kann leicht errechnet werden:

Nachlaufzeit (sec)=Wert*0.02

Dabei ist allerdings zu beachten, daß die Nachlaufzeit nicht zu kurz wird, da sich ansonsten der Computer bei Diskettenzugriffen "aufhängen" kann. Als ideal ermittelte ich einen Wert von 00BFH. Dies entspricht einer Nachlaufzeit von ca. 4 Sekunden. Dabei entstanden auch beim Compilieren unter Turbo-Pascal auf der Diskette keine Fehler.

Weiters wird oft die standardmäßige Initialisierung der 80-Zeichenkarte als störend empfunden. Dafür kann ebenfalls leicht Abhilfe geschaffen werden, allerdings muß auch diese Änderung direkt auf den Systemspuren durchgeführt werden.

Im Betriebssystem befindet sich eine Tabelle, welche die Initialisierungswerte für die 80-Zeichenkarte enthält. Diese werden beim Laden des Systems in die Kontrollregister des VDP ausgegeben. Ab dem Betriebssystem CP/M 2.22 wird von der Ausgaberoutine auch das Vorhandensein einer 80-Zeichenkarte überprüft.

Diese Tabelle umfaßt 16 Byte und hat standardmäßig folgende Werte:

```
00   6D 50 59 0C 1F 02 18 1A
08   00 07 60 07 00 00 00 00
```

Diese Tabelle kann auf der Diskette in folgenden Positionen gefunden werden (128 Bytes pro Sektor):

	Spur	Sektor	Byte
BASIC	0	5	75H
CP/M 2.20	0	23	54H
CP/M 2.22	0	27	11H
CP/M 2.23	0	27	20H

Durch Änderung dieser Werte kann praktisch jede Monitorinitialisierung erreicht werden. Im folgenden nun eine Aufstellung mit den bis jetzt ermittelten optimalsten Werten und ihrer Bedeutung:

00	6C	Anzahl der Zeichen in einer Bildschirmzeile inklusive der nicht dargestellten (Rand)
01	50	Anzahl der anzuzeigenden Zeichen in einer Bildschirmzeile
02	58	HSYNC-Position (in dieser Spalte wird eine Horizontal-Synchronisierung durchgeführt)
03	08	HSYNC-Breite (In dieser Spalte beginnt die Zeichendarstellung)
04	1E	Anzahl der Zeichen in einer Bildschirmspalte inklusive der nicht dargestellten (Rand)
05	05	Vertikal-Abgleich (in dieser Zeile beginnt die Zeichendarstellung)
06	18	Anzahl der in einer Bildschirmspalte dargestellten Zeichen
07	1B	VSYNC-Position (in dieser Zeile endet die Zeichendarstellung)
08	02	Interlace (Zeilensprung)
09	09	Anzahl der Bildzeilen pro Zeichen
0A	68	Darstellungsart und Zeilenstartadresse des Cursors (in diesem Fall langsam blinken und Startzeile 8)
0B	08	Zeilenendadresse des Cursors (in diesem Fall Endzeile 8 und ergibt gemeinsam mit der Startzeile einen UNDERLINE-Cursor)
0C	00	Startadresse des Speichers H
0D	00	Startadresse des Speichers L
0E	00	Cursorposition H
0F	00	Cursorposition L

Der Video-Controller umfaßt eigentlich 18 Controllregister. Die beiden in der Tabelle fehlenden sind allerdings Read-Only Register wodurch eine Ausgabe an diese Register entfallen kann. Diese enthalten normalerweise die Lichtgriffelposition.

Bei einer Änderung im DISK-BASIC-System sollte das Byte 0AH allerdings den Wert 60H und das Byte 0BH den Wert 07H enthalten, da dies den BASIC-Standardwerten entspricht.

Für mehr Information über die Funktionen der verwendeten Werte sei auf das Buch

Hard- und Softwarepraxis

von Rolf Dieter Klein verwiesen. Im Heft 3/85 wurde im Hardware-Artikel übrigens auch über die Register der 80-Zeichenkarte geschrieben.

Rotschek Wolfgang

!!! WICHTIG !!!

Mitglieder aufgepaßt,

Zahlschein liegt bei!

Für alle Clubmitglieder, die noch keinen Clubbeitrag für das Jahr 1985 bezahlt haben, liegt in diesem Heft ein Zahlschein bei. Bitte füllen Sie ihn sorgsam aus, und zahlen Sie bitte so bald wie möglich.

Höhe der Clubbeiträge:

Erwachsene: S 500,-

Schüler, Studenten und Lehrlinge: S 250,-

```
*****************************************
*                                       *
*        Deutscher Zeichensatz für      *
*          SVI-318 und SVI-328          *
*                                       *
*****************************************
```

Die gleiche Headline finden Sie auch im SVI-Journal 4/84, Seite 19. Dort ist nämlich ein kleines BASIC-Programm untergebracht, das die Bitmuster der Umlaute in das VRAM schreibt. Dieses Programm hat nur einen Nachteil: Nach einem SCREEN 0 ist wieder der alte Zeichensatz im VRAM und man muß das Programm nocheinmal starten.

Um diesem Übel vorzubeugen, habe ich ein Maschinenprogramm entwickelt, das bei jedem SCREEN 0 die Umlaute wieder in das VRAM schreibt.

Die 'SCREEN 0'-Routine ruft mehrere Unterprogramme auf, unter anderem auch den Zeichengenerator. Jener schreibt ab &H800 96 normale und invertierte ASCII-Zeichen in das VRAM und danach 64 Graphikzeichen. Gleich die erste Anweisung des Zeichengenerators ist ein CALL &HFFAB, und dort fange ich den Computer ab.

Allen "Graphikfanatikern" sei gesagt, daß die Umlaute in der Graphik nicht verfügbar sind. Dies resultiert daraus, daß man die Routine zur Ausgabe eines Zeichens nicht abfangen kann. Es wäre theoretisch schon möglich, doch müßte man die komplette Routine zur Erkennung des aktuellen Ausgabegerätes mitverändern (40 oder 80 Zeichen Text, SCREEN 1 oder 2, Drucker oder Datei). Und das wäre zuviel. Um aber die Umlaute auch in der Graphik verwenden zu können, muß man das ROM in das RAM umkopieren und dort den Zeichensatz direkt verändern.

Sehen wir uns nun kurz das Programm an. Die einzelnen Routinen erklären sich weitgehend selbst. Lediglich zur Tabelle am Ende des Programms sei noch etwas gesagt:

Die Tabelle von TAB_CHR hat folgendes Aussehen:

Ein Zeichen besteht aus 9 Bytes:

Byte 0 enhält den ASCII-Code des Zeichens, die Bytes 1 bis 8 bilden dann das Bitmuster des Zeichens. Das Ende der Tabelle wird mit 0 gekennzeichnet.

Natürlich sind die ASCII-Zeichen auch invertiert verfügbar. Mit diesem Programm ist nun auch der Weg zu einem völlig eigenen Zeichensatz offen. Das Einzige, was man tun muß, um einem Zeichen ein "neues Kleid" zu geben, ist, die neue Matrix in der Tabelle "tab_chr" unterzubringen. Das Programm orientiert sich nämlich nicht an einer Zählvariablen, die die Anzahl der zu ändernden Zeichen beinhaltet, sondern nur nach der Null in der Tabelle.

Zum Abschluß ist wieder das Assemblerlisting mit einem Hexdump ausgedruckt, falls manche Bediener über keinen Assembler verfügen.

Die Tastenbelegung der deutschen Umlaute, sie liegen rechts von der P-Taste in folgender Form:

ohne SHIFT:	p	Ä	Ü	Ö
mit SHIFT:	P	ä	ü	ß
mit LEFT-GRAPH:	unwichtig	ö		unwichtig

Die Zeichen werden auf jedem Drucker mit deutschem Zeichensatz richtig ausgegeben.

```
100 REM           org  d000h
105 REM           ;
110 REM wr_hl     equ  373ch
115 ' 255 in INVERT invertiert Bitmuster
120 REM invert    equ  fdb4h
125 ' CHR_BUF Anfang Bitmusterbuffers
130 REM chr_buf   equ  fddch
135 ' GEN_ROM generiert Zeichen aus ROM
140 REM gen_rom   equ  35b0h
145 ' GEN_RAM schreibt Zeichen aus CHR_BUF
150 REM gen_ram   equ  35b4h
155 '
160 ' Initialisieren von RESTORE_CHR
165 '
170 REM           ld   a,c3h
175 REM           ld   (ffabh),a
180 REM           ld   hl,restore
185 REM           ld   (ffabh+1),hl
190 REM           jp   3541h
195 '
200 ' Einsprung von 3584H, der Routine,die
205 ' den Zeichensatz ins VRAM schreibt
210 '
215 REM restore   inc  sp
220 REM           inc  sp
225 '
230 ' Alle ASCII Zeichen von " " bis "DEL"
235 ' normal und danach invers schreiben
240 '
245 REM           ld   hl,800h
250 REM           call wr_hl
255 REM           call all_char
260 REM           xor  a
265 REM           ld   (fdb4h),a
270 '
275 ' Graphikzeichen schr.
280 '
285 REM           ld   a,a0h
290 REM rest_1    call char
295 REM           inc  a
300 REM           jr   nz,rest_1
305 REM           ret
310 '
315 ' ASCII-Zeichen normal
320 '
325 REM all_char xor  a
330 REM           call all_1
335 '
340 ' ASCII-Zeichen invers
345 '
350 REM           ld   a,255
355 REM all_1     ld   (invert),a
360 '
365 ' ASCII-Zeichen von " " bis "DEL"
370 '
375 REM           ld   a,20h
380 REM all_2     call char
385 REM           inc  a
390 REM           jp   p,all_2
395 REM           ret
400 '
405 ' Ein Zeichen ins VRAM schreiben
410 ' eventuell invertieren
415 '
420 REM char      call own_char
425 REM           jp   nc,gen_rom
430 REM           ;
435 REM           and  a
440 REM           push af
445 REM           jp   gen_ram
450 '
455 ' Herausfiltern der eigenen Zeichen
460 '
465 REM own_char  ld   c,a
470 REM           ld   hl,tab_chr
475 '
480 ' Durchsuchen der eigenen Tabelle
485 '
490 REM own_1     ld   a,(hl)
495 REM           and  a
500 REM           jr   z,no_own
505 REM           cp   c
510 REM           jr   z,own_2
515 REM           ld   de,9
520 REM           add  hl,de
525 REM           jr   own_1
530 '
535 ' Eigenes Zeichen gefunden, Bitmuster
540 ' in Bitmustertabelle schreiben und
```

```
545 ' vom ROM ausgeben lassen
550 '
555 REM own_2     inc   hl
560 REM           ld    de,chr_buf
565 REM           ld    a,c
570 REM           ld    bc,8
575 REM           ldir
580 REM           scf
585 REM           ret
590 '
595 ' Kein eigenes Zeichen gefunden, ROM-
600 ' Zeichengenerator arbeiten lassen
605 '
610 REM no_own    ld    a,c
615 REM           and   a
620 REM           ret
625 '
630 ' Tabelle fuer eigene Zeichen
635 ' Byte 0  : Charactercode
640 ' Byte 1-8: Bitmuster
645 ' Byte 0  : =0 Endemarkierung
650 '
655 REM tab_chr   db    123  ;ä
660 REM           db    80,0,112,8,120,136
665 REM           db    120,0
670 REM           ;
675 REM           db    125  ;ü
680 REM           db    144,0,144,144,144
685 REM           db    144,104,0
690 REM           ;
695 REM           db    126  ;ß
700 REM           db    96,144,144,160,144
705 REM           db    144,160,0
710 REM           ;
715 REM           db    91   ;Ä
720 REM           db    136,32,80,136,248
725 REM           db    136,136,0
730 REM           ;
735 REM           db    92   ;ö
740 REM           db    136,112,136,136,136
745 REM           db    136,112,0
750 REM           ;
755 REM           db    93   ;Ü
760 REM           db    136,0,136,136,136
765 REM           db    136,112,0
770 REM           ;
775 REM           db    124  ;ä
780 REM           db    80,0,112,136,136
785 REM           db    136,112,0
790 REM           ;
795 REM           db    0
800 REM           ;
805 REM           end
```

```
D000 3E C3 32 AB FF 21 0E D0 >.2..!..
D008 22 AC FF C3 41 35 33 33 "...A533
D010 21 00 08 CD 3C 37 CD 26 !...<7.&
D018 D0 AF 32 B4 FD 3E A0 CD ..2..>..
D020 39 D0 3C 20 FA C9 AF CD 9.< ....
D028 2C D0 3E FF 32 B4 FD 3E ,.>.2..>
D030 20 CD 39 D0 3C F2 31 D0  .9.<.1.
D038 C9 CD 44 D0 D2 B0 35 A7 ..D...5.
D040 F5 C3 B4 35 4F 21 64 D0 ...5O!d.
D048 7E A7 28 15 B9 28 06 11 ß.(..(..
D050 09 00 19 18 F3 23 11 DC .....#..
D058 FD 79 01 08 00 ED B0 37 .y.....7
D060 C9 79 A7 C9 7B 50 00 70 .y..äP.p
D068 08 78 88 78 00 7D 90 00 .x.x.ü..
D070 90 90 90 90 68 00 7E 60 ....h.ß`
D078 90 90 A0 90 90 A0 00 5B .......Ä
D080 88 20 50 88 F8 88 88 00 . P.....
D088 5C 88 70 88 88 88 88 70 ö.p....p
D090 00 5D 88 00 88 88 88 88 .ü......
D098 70 00 7C 50 00 70 88 88 p.öP.p..
D0A0 88 70 00 00 00 00 70 00 .p....p.
```

```
*****************************************
*                                       *
*                 HERO                  *
*                                       *
*****************************************
```

Man kann es gewissermaßen zu den gehobenen Adventurespielen reihen. Man braucht hier jedoch nicht vor einem mit Texten überhäuften Schirm zu sitzen, der einem weismachen will, daß man sich gerade in einem Schloß befindet und einen Schatz suchen muß, sondern man sieht in vorzüglicher Graphik, wo man sich befindet, beziehungsweise wohin man als nächstes weitergehen soll. Dieses Spiel nennt sich HERO und wurde von 'Activision' ins Leben gerufen.

Ziel beim HERO ist, durch ein Labyrinth eines Bergwerks zu eilen und eingeschlossene "Kumpel" zu befreien. Natürlich lauern auch wieder Gefahren auf den Retter, sonst wäre das Spiel ja fad. Bevor wir jedoch auf die Schikanen eingehen, wollen wir zuerst klären, welche Bewegungen und Waffen der Held dieses Spieles hat.

Die beiden Cursor-Tasten "Links" und "Rechts" sorgen für die analogen Bewegungen. Durch die Cursor-Taste "Oben" kann man einen Propeller in Gang setzen, der den 'Hero' zum Hubschrauber werden läßt. Er spricht erst nach einer kleinen Verzögerungszeit an und hört ebenso erst kurz nach Auslassen der Taste wieder auf. Durch Cursor-Taste "Unten" darf man Bomben legen. Hier erhebt sich natürlich die Frage, wozu braucht man Bomben?

Die Antwort ist ganz einfach. Eines der Hindernisse sind nämlich Mauern, die im Weg stehen. Diese kann man nur durch Bomben zerstören. Während des Fluges kann der Retter keine Bomben legen. Ebenso muß er danach trachten, so schnell wie möglich nach Ablage der Mine eine gewisse Sicherheitsdistanz zu bekommen oder in den 'Schwebemodus' zu kommen. Wenn man bestimmte Modi erreicht hat, werden Glühmauern eingesetzt, deren Berührung tödlich ist. Setzt man die Bomben übrigens zu weit von der Mauer entfernt ab, zerbricht sie nur teilweise.

Neben den Mauern gibt es noch allerlei Getier, das sich in den Gängen bewegt. Man muß es in den meisten Fällen vernichten, indem man mit der Space-Taste Schüsse abfeuert. Wir wollen auf die Vielfalt der Hindernisse nicht eingehen, da die Erforschung der Gänge und der auftretenden Gefahren einen Großteil des Reizes ausmacht, den dieses Game bietet. Verraten sei nur noch, daß es im Laufe der Zeit sehr schwer wird weiterzukommen. Neben glühenden Felsen, Wassergräben und Schluchten, die ins Nichts führen, werden auch die Tiere immer agressiver. Ab und zu wird es auch so dunkel, daß man (fast) nichts mehr sieht. Dann muß man sich im Dunkeln 'weitertasten'. Ob es ein Ende gibt, oder ob es immer schwerer wird, weiterzukommen, verraten wir nicht.

Gezählt wird in erreichten Sektionen und Punkten. Die Punkte setzen sich aus der Zeit, die gebraucht wurde, den Bomben, die verbraucht wurden und der Anzahl der Tiere und Mauern, die zerstört wurden, zusammen. Man bekommt übrigens pro Sektion immer nur eine bestimmte Zeit und sechs Bomben zur Verfügung. Aus diesem Grund sollte man sich nicht all zu lange Zeit lassen und Bomben nicht grundlos verschwenden.

Das Spiel wurde für MSX geschrieben, ist auf Kassette erhältlich und kostet 390 Schilling. Kaufen kann man es in jedem besseren MSX-Fachgeschäft.

```
*****************************************
*                                       *
*            UpString in PASCAL         *
*                                       *
*****************************************
```

Bei meinen Bemühungen, ein Karteiprogramm für die Mitgliederverwaltung des SVI-Clubs unter der Programmiersprache PASCAL zu schreiben, stand ich vor dem Problem, die Namen unter Berücksichtigung der deutschen Sonderzeichen in alphabetisch richtiger Reihenfolge zu sortieren.

Vergleicht man zum Beispiel mittels einer Vergleichsoperation die Wörter Öse (Oese) und Osten, so wird der Computer als Ergebnis liefern, daß das Wort Osten in der alphabetischen Wertigkeit kleiner ist. Weiters erscheint ein kleingeschriebenes Wort höherwertiger als ein großgeschriebenes (vergleiche ASCII Tabelle).

Dieses Problem umgehe ich dadurch, daß ich sämtliche Indexwörter (diese stehen gemeinsam mit einem Zeiger in einem Array und dienen einem raschen Diskettenzugriff) in Großschreibung umwandle, wobei außerdem noch alle deutschen Sonderzeichen durch Standardzeichen (ä wird AE) ersetzt werden. Nach einer solchen Umwandlung erscheinen die dadurch gewonnen Zeichenketten bei einem Sortier- oder Vergleichverfahren in alphabetisch richtiger Reihenfolge.

Zu der von mir entwickelten Procedur ist an und für sich nicht viel zu sagen, da sie weitestgehend selbsterklärend ist; beachtet werden müssen lediglich die Bedingungen für die Variablenübergabe. Das Programm ist in Bild 2 zu sehen. 'UpString' ist, wie man unschwer erkennen kann, ein Unterprogram und wird daher mit dem Routinennamen 'UpString' aufgerufen. Hinter dem Namen folgt eine Klammer mit der Variablen, die in Großbuchstaben umgewandelt werden soll. Ebenfalls in der Klammer findet sich auch die Variable, die die neue Zeichenkette wieder im Hauptprogramm beinhaltet. Ein Beispiel für eine Übergabe ist ebenfalls in Bild 2 zu sehen.

Will man auf die Umwandlung der deutschen Sonderzeichen verzichten, so kann die CASE-Anweisung und die Variable Hold entfallen.

Die einzige Funktion müßte dann so lauten, wie in Bild 1 ersichtlich ist.

Wolfgang Rotschek

```
for i:=Length(Low_String) downto 1 do
  insert(UpCase(copy(Low_String,i,1),Up_String,1);
```

Bild 1: Die verkürzte Fassung der Routine. Die Umlaute werden nicht beachtet, da die CASE-Anweisung fehlt. Lediglich die Klein- und Großschreibung wird auf gemeinsame Großschreibung umgewandelt.

```
PROGRAM STRINGS;
TYPE STRING_BUFFER   = STRING(.30.);
PROCEDURE UPSTRING (LOW_STRING   :   STRING_BUFFER;
                    VAR
                      UP_STRING   :   STRING_BUFFER);
VAR
  I             :  BYTE;
  HOLD          :  CHAR;

BEGIN
  UP_STRING:='';
    FOR I:=LENGTH(LOW_STRING) DOWNTO 1 DO BEGIN
      HOLD:=COPY(LOW_STRING,I,1);
        CASE HOLD OF
             'ä','Ä': INSERT('AE',UP_STRING,1);
             'ö','Ö': INSERT('OE',UP_STRING,1);
             'ü','Ü': INSERT('UE',UP_STRING,1);
             'ß': INSERT('SS',UP_STRING,1);
             ELSE INSERT(UPCASE(HOLD),UP_STRING,1);
        END;
   END;
END;

VAR
  NAME,NAME_UP        : STRING(.30.);
  E                   : CHAR;

BEGIN
  E := CHR(27);
  CLRSCR;
  REPEAT
    WRITE (E,'pNAME EINGEBEN:',E,'q ');
    READLN (NAME);
    UPSTRING (NAME,NAME_UP);
    WRITELN;
    WRITELN (NAME_UP);
    WRITELN;
  UNTIL NAME = '*';
END.
```

Bild 2: Die gesamte Routine "in ihrer vollen Länge". Neben der Aufhebung der Kleinschreibung wird auch noch auf die Umlaute eingegangen.

```
*****************************************
*                                       *
*            Funktionenplotter          *
*                                       *
*****************************************
```

Wenn man den Computer dazu bringen möchte, Funktionen mit einigem Bedienerkomfort graphisch auszuwerten, dann bedarf es schon eines größeren Aufwandes. Das unten abgedruckte Programm ist ungefähr 12 KByte lang. Dafür bietet es auch viele Features. Fangen wir jedoch von grund auf mit der Erklärung an:

Ziel eines Funktionenplotters ist es, eine beliebige Funktion auf dem Schirm darzustellen. Dazu gibt man den Funktionsterm ein, der Computer berechnet dann daraus die einzelnen Punkte. Je mehr verschiedene Elemente ein Term haben darf, desto mannigfaltiger werden die Graphen. In unserem Programm darf man natürlich alle BASIC-Operatoren und BASIC-Funktionen verwenden. Die frei verwendbaren Variablen sind auf die Zahl 10 eingeschränkt. Sie werden K0 bis K9 genannt. Zusätzlich zum Term kann man noch angeben, ob man polare oder kartesische Koordinatendarstellung möchte. Näheres über diese beiden Darstellungsarten kann man in der 3D-Graphik-Nachlese erfahren.

Um größtmögliche Flexibilität bei der Anzeige des Graphen zu haben, werden einige Parameter abgefragt. Am Anfang des Programms fragt der Computer, ob das Datensichtgerät Farbe hat oder Schwarz/Weiß ist. Danach wird geklärt, ob ein Monitor oder ein Fernseher verwendet wird. Der Computer stellt das Verhältnis der Bildschirmentzerrung nach dem soeben genannten Kriterium ein. Wie schon oben erwähnt, darf man dann die Funktion eingeben. Auf die Bedienungsanleitung kommen wir später zurück. Nach der Eingabe des Terms folgen die Änderung der Konstanten K0 bis K9 und ein Fragenkatalog:

1) Sollten die polaren Koordinaten gewählt werden, kann der Startwert für X eingegeben werden.

Bei allen weiteren Fragen darf man mit ENTER antworten, der Computer nimmt dann Standardwerte.

2) Werteinkrement: Setzt fest, in welchen Intervallen die Laufvariable erhöht wird.

3) Werteanzeige an/ aus: Am unteren Rand erscheinen beim Zeichnen der Funktion die jeweiligen Arbeitswerte.

4) Koordinatenachsen: Entscheidet, ob die Achsen angezeigt werden oder nicht.

5) Gleicher Maßstab für X- und Y-Koordinaten, ja oder nein.

6) Wahl, ob ein Ton, der äquivalent zur Höhe des Funktionswertes ist, erzeugt werden soll oder nicht.

Zum Schluß darf man noch den sichtbaren Ausschnitt des Graphen bestimmen, der Y-Wert wird auf Wunsch selber berechnet. Danach zeichnet der Computer den Graphen.

Das Programm besticht neben seiner Leistung vor allem durch die sehr ausführlichen Help-Funktionen. Neben einer 7 Bildschirmseiten umfassenden Bedienungserklärung, die kein Geheimnis offen läßt, gibt es auch noch während des Zeichnens die Möglichkeit, mit einem Tastendruck alle 14 "Befehle" anzeigen zu lassen, die während des Zeichnens verwendet werden dürfen. Hier braucht man nichts erklären, das Programm erklärt sich wirklich selbst. Nebenbei ist auch die Kontrolle der Eingaben vorbildlich.

Durch die vielfachen Anweisungen während des Zeichnens kann man sehr schöne Effekte erzielen, zum Beispiel Sinuskurven in verschiedenen Amplituden und Frequenzen.

Sehen wir uns das Programm etwas an:

Ursprünglich bestand das Programm nur aus BASIC. Da jedoch die Routine zum Ausrechnen eines Funktionsterms im BASIC langwierig und kompliziert ist, haben wir sie gegen das Maschincode-Äquivalent aus Heft 6/84 ausgetauscht. Wir werden das MC-Programm am Ende des Textes noch kurz genauer betrachten. Durch diese Änderung wurde ein Abfrage, ob die Konstanten K0 bis K9 heißen, praktisch unmöglich gemacht. Im übrigen kann man nette Effekte erzielen, wenn man im Funktionsterm interne Variablen einsetzt (dies ist in der ursprünglichen Version unmöglich).

Das BASIC-Programm selber ist, obwohl es sehr komprimiert programmiert ist, klar gegliedert und mit einigen REM-Zeilen versehen. Man kann sich daher relativ leicht in das Programm einschalten und es manipulieren. Die Helproutine am Ende der Entwicklung ist etwas abgesetzt. Um sich unnötige Schreibarbeit zu ersparen, kann man die Helproutine weglassen und in Zeile 1660 ein RETURN einfügen. Das Programm stürzt deswegen nicht gleich ab, die Bedienungsanleitung muß allerdings entbehrt werden.

Die Anleitung ist in einem interessanten Prinzip abgelegt. Der Text steht in DATA-Zeilen, die von einer Schleife eingelesen werden. Da der Bildschirmaufbau im Großen und Ganzen bei jedem Screen gleich ist, kann man eine Schleife verwenden. Die "@"-Zeichen (CHR$(64)) simulieren leere PRINTS, ein Dollarzeichen zeigt dem Computer das Ende der Seite.

Kommen wir nun zum MC-Programm.

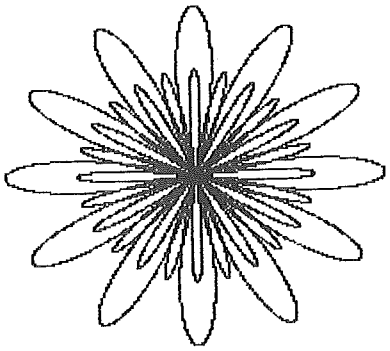
Es ist im Wesentlichen die Routine aus Heft 6/84. Der einzige Unterschied, der besteht, ist in der zweiten Hälfte zu finden. Um sich die beiden BASIC-Programme zu ersparen, die im Dezember vorigen Jahres noch notwendig waren, richtet sich ein MC-Loader alle Zeiger und verschiebt die Arbeitsroutine nach &H8000. Da Programmierer meistens sehr schreibfaul sind (uns geht es zumindestens so), wollten wir unbedingt das nachfolgende Ladekommando für das BASIC-Programm im MC-Code unterbringen. Um die nötige Flexibilität zu gewährleisten (CLOAD, LOAD"CAS:",R und so weiter), legen wir den Befehl einfach in den Eingabebuffer des Computers, wandeln ihn mit einer ROM-Routine in eine BASIC-Zeile um und lassen ihn mit einer zweite ROM-Routine gleich ausführen. Wie im Artikel über die ROM-Sprünge in diesem Heft ersichtlich ist, springt der Computer von der Adresse &H165 aus nicht zu einer Lade-Routine, sondern führt nur eine BASIC-Zeile aus. Man kann nun ganz einfach den entsprechenden Ladebefehl selber eintragen. Lediglich die Null am Ende muß bleiben. Die Länge der Zeile wird in Zeile 680 selber erzeugt und in BC geladen. Programmierer ohne einen Assembler müssen BC selber richten!

Wir haben derzeit folgendes angenommen: BASIC-File wurde mit SAVE "1:FUNPLT" abgespeichert, die Disk muß in Laufwerk 1 sein.

Man kann übrigens die Routine 165H auch mit CALL anspringen, dann kommt sie wieder in das MC-Programm zurück. Auf diese Weise kann man BASIC-Befehle während des MC-Programms abarbeiten lassen, wenn man selber zu faul ist, eine Routine als Ersatz zu schreiben.

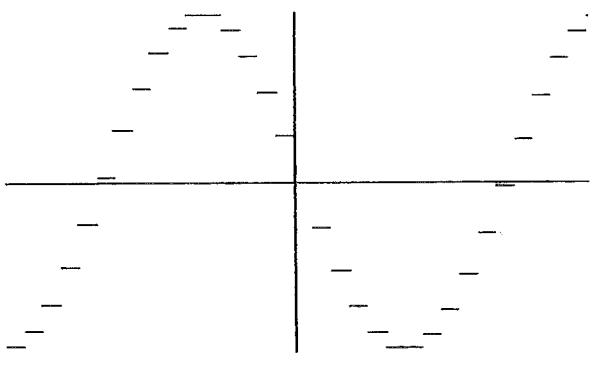
Der Funktionenplotter ist sowohl auf Disk als auch mit Kassette lauffähig.

```
10 ' ***    Funktionenplotter
20 ' *** Von Marcus Hopfer  (C) 1985
30 '
40 DEFUSR=&H8000:SCREEN0,0:WIDTH39:CLEAR2000
:DEFSTRE,F,V:DEFINTA-D,L-R:DIME(13):XM=255:Y
M=180:J1=YM/45:ZF=1.08:R=RND(-TIME):GOTO60
50 Y=USR(F):RETURN
60 PRINT:PRINT"   Mit welchem Geraet arbeite
n Sie :":LOCATE11,6:PRINT"1) Farbe":LOCATE11
,8:PRINT"2) Schwarzweiss":LOCATE9,12:PRINT"E
rbitte Eingabe : ";
70 E=INPUT$(1):IFE="2"THENC1=15:C2=1:C3=1:EL
SEIFE="1"THENC1=15:C2=4:C3=5ELSE70
80 LOCATE11,6:PRINT"1) Monitor":LOCATE11,8:P
RINT"2) Fernseher   ":LOCATE27,12
90 E=INPUT$(1):IFE="1"THEN110ELSEIFE="2"THEN
ZF=1.3142ELSE90
100 ' ***  Titeldisplay
110 COLORC1,C2,C3:SCREEN1:PRINT" ";STRING$(4
0,210);:LOCATE6,40:PRINTSTRING$(40,210);:FOR
Z=8TO32STEP8:LOCATE6,Z:PRINT"R";:LOCATE240,Z
:PRINT"R";:NEXTZ:E1="Funktionenplotter":FORZ
=1TOLEN(E1):LOCATEZ*13+7,20+5*SIN(Z*18):PRIN
TMID$(E1,Z,1);:NEXTZ
120 SPRITE$(1)="<B"+CHR$(157)+CHR$(161)+CHR$
(161)+CHR$(157)+"B<":LOCATE20,70:PRINT"Copyr
ight Hopfer Software      1985":PUTSPRITE1,(1
84,69),C1,1:LOCATE44,110:PRINT"Marcus Hopfer
 wuenscht Ihnen":LOCATE68,126:PRINT"viel Ver
gnuegen !!!"
130 LOCATE44,170:PRINT"Bitte eine Taste druc
ken...";:E=INPUT$(1)
140 RESTORE150:FORZ=0TO13:READE(Z):NEXT
150 DATA0 - Zeichenvorgang fortsetzen,1 - Ne
uen Startwert fuer X setzen,2 - Neuer Werteb
ereich fuer X,3 - Neuer Wertebereich fuer Y,
4 - Werteinkrement aendern,5 - Bildschirm lo
eschen,6 - Werteanzeige  An / Aus
160 DATA7 - Konstanten aendern,8 - Koordinat
enachsen  An,9 - Neue Funktion eingeben,> -
Bild auf Kassette speichern,F - Funktion auf
listen,T - Ton  An / Aus,M - Gleicher Massta
b  An / Aus
```



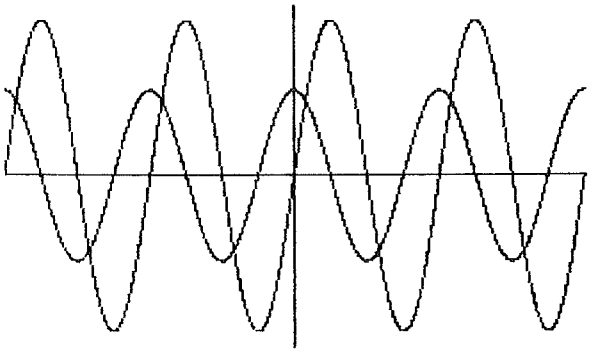
Formel: Y=COS(K0*X), K0 bestimmt Anzahl der Blätter, polare Kordinaten

```
170 SCREEN0:LOCATE0,8:PRINT"Brauchen Sie Ins
truktionen (J/N) ? ";
180 E=INPUT$(1):IFE="J"ORE="j"THENGOSUB1660
190 ' *** Eingabe der Funktion
200 F="":GOSUB1400:LOCATE9,5:PRINT"Koordinat
ensystem :":LOCATE11,8:PRINT"1) Polar":LOCAT
E11,10:PRINT"2) Kartesisch":LOCATE8,15:PRINT
"Bitte waehlen Sie : ";
210 E=INPUT$(1):IFE="1"THENK0=1ELSEIFE="2"TH
ENK0=2ELSEK0=0:GOTO210
220 GOSUB1400:LOCATE0,5:PRINT"Funktion :":LO
CATE4,8:PRINTF;:LOCATE0,8:LINEINPUT"y = ";F:
PRINT:IFLEN(F)>120THENCLS:PRINT"Eingegebene
Funktion ist zu lang.":PRINT:PRINT:GOSUB1420
:F="":GOTO220
230 IFF=""THENCLS:PRINT"Lassen Sie den Unsin
n !!!":PRINT:PRINT:PRINT:GOSUB1420:GOTO220
240 Z1=0:Z2=0:FORZ=1TOLEN(F):V=MID$(F,Z,1):I
FV="("THENZ1=Z1+1:NEXTZELSEIFV=")"THENZ2=Z2+
1:NEXTZELSENEXTZ
250 IFZ1<>Z2THENCLS:PRINT"Fehler in Klammern
setzung :":PRINT:PRINT:PRINTF:PRINT:PRINT:GO
SUB1420:GOTO220
260 FORZ=1TOLEN(F):AA=ASC(MID$(F,Z,1)):IF(AA
>96ANDAA<122)THENAA=AA-32:MID$(F,Z,1)=CHR$(A
A):NEXTZELSENEXTZ
270 ONERRORGOTO1610
280 ' *** Eingabe der Nebeninformationen
290 GOSUB50:FORZ=0TO9:K(Z)=0:NEXTZ:GOSUB1480
:GOSUB1400:LOCATE0,5:PRINT" Alle Konstanten
haben den Wert=0 .":PRINT:PRINT"        Wolle
n Sie die Werte":PRINT"          aendern (J/N)
 ? ";
300 E=INPUT$(1):IFE="n"ORE="N"THEN320ELSEIFE
<>"j"ANDE<>"J"THEN300
310 PRINTE;:LOCATE0,11:FORZ=0TO9:PRINTTAB(5)
;"Konstante  K";MID$(STR$(Z),2,1);" = ";:INP
UTK(Z):NEXTZ:GOSUB1480
320 GOSUB1400:IFK0=1THENLOCATE0,5:SX$="":INP
UT"Startwert fuer x = ";SX$:IFSX$=""THEN320E
LSESX=VAL(SX$)
330 LOCATE0,7:PRINT"Bei druecken von <Enter>
, werden":PRINT"die Fragen vom Computer":PRI
NT"selbst beantwortet :":PRINT:WI=0:INPUT"We
rteinkrement = ";WI:PRINT:PRINT"Werteanzeige
 : 1=An / 2=Aus ? ";
340 E=INPUT$(1):IFE="1"THENWA=1ELSEWA=2:E="2
"
350 PRINTE:PRINT:PRINT"Koordinatenachsen : 1
=Ja / 2=Nein ? ";
360 E=INPUT$(1):IFE="2"THENKA=2ELSEIFK0=1THE
NKA=2:E="2"ELSEKA=1:E="1"
370 PRINTE:PRINT:PRINT:PRINT"Gleichen Massta
b fuer X u. Y (J/N) ? ";
380 E=INPUT$(1):IFE="j"ORE="J"THENMA=1ELSEIF
E="N"ORE="n"THENMA=2ELSEIFK0=1THENMA=1:E="J"
:ELSEMA=2:E="N"
390 PRINTE:PRINT:PRINT:PRINT"Tonausgabe (J/N
) ? ";
400 E=INPUT$(1):IFE="J"ORE="j"THENJ=1ELSEJ=2
410 GOSUB1400:LOCATE0,5:PRINT"Bitte definier
en Sie die Groesse":PRINT"des Bildschirmauss
chnittes :":PRINT:PRINT:PRINT"Wertebereich f
uer X :":LOCATE0,11:INPUT"Von : ";XA$:IFXA$=
""THEN410ELSEXA=VAL(XA$):LOCATE17,11:INPUT"
 Bis : ";XE$
420 IFXE$=""THEN410ELSEXE=VAL(XE$)
430 IFXA=XETHEN410ELSEIFK0=2THENSX=XA
440 IFWI=0THENWI=(ABS(XA-XE))/500
450 IFMA=1THENYA=XA:YE=XE:GOTO560
460 LOCATE0,14:PRINT"Wollen Sie einen Werteb
ereich":PRINT"fuer Y angeben (J/N) ? ";
470 E=INPUT$(1):IFE="J"ORE="j"THENGOTO540ELS
EE="N"
480 ' *** Errechnen des Y-Wertebereiches
490 ONERRORGOTO1610:PRINTE:PRINT:PRINT:PRINT
"Bitte warten.... ";:LOCATE,,0:Z1=(ABS(XA-XE
))/100:X=XA:GOSUB50:YA=Y:YE=Y:FORZ=XATOXESTE
PZ1:X=Z:GOSUB50:LOCATE0,20,0:PRINTCSNG(Z);"
    ";:IFK0=1THENY=Y*SIN(X)
500 IFY<YATHENYA=Y:NEXTZELSEIFY>YETHENYE=Y:N
EXTZELSENEXTZ
510 Z1=0:X=0:Y=0:YA=YA*1.1:YE=YE*1.1:GOSUB14
00:LOCATE0,5,1:PRINT"Computer waehlt fuer Y
Bereich :":PRINT:PRINT"Von : ";YA:PRINT:PRIN
T"Bis : ";YE:PRINT:PRINT:PRINT
520 PRINT"Sind Sie einverstanden (J/N) ? ";
530 E=INPUT$(1):IFE="J"ORE="j"THEN560ELSEIFE
```



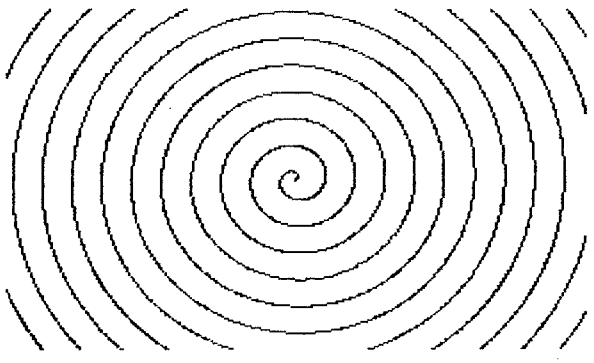
Formel: Y=SIN(6*INT(X))
kartesische Koordinaten

```
  "N"ANDE<>"n"THEN530
540 GOSUB1400:LOCATE0,5:PRINT"Wertebereich f
uer Y :":LOCATE0,8:INPUT"Von : ";YA$:IFYA$="
"THEN540ELSEYA=VAL(YA$):LOCATE17,8:INPUT"  B
is : ";YE$:IFYE$=""THEN540ELSEYE=VAL(YE$):IF
YE<YATHENSWAPYE,YA
550 IFYA=YETHEN540
560 IFC2=4THENC1=1:C2=7:C3=4:COLORC1,C2,C3
570 SCREEN1:IFYA=YETHENY=YE+1E-03
580 ' *** Errechnen von Koordinatenachsen
590 XB=ABS(XA-XE):YB=ABS(YA-YE):S1=XM/XB:S2=
YM/YB
600 IFXA=0THENX0=0:GOTO610ELSEIFXE=0THENX0=X
M:GOTO610ELSEIF(XA>0ANDXE>0)THENX0=-(XA*S1):
GOTO610ELSEX0=ABS(XA*S1)
610 IFYA=0THENY0=YM:GOTO620ELSEIFYE=0THENY0=
0:GOTO620ELSEY0=(YE*S2)
620 IFY0<0THENYF=YM/YB:GOTO640ELSEIFY0>YMTHE
NYF=YM/YB:GOTO640ELSEIFY0=YMTHENYF=YM/YB:GOT
O640ELSEIFY0=0THENYF=YM/YB:GOTO640ELSEIFY0>(
YM-Y0)THENYF=Y0/YEELSEYF=(YM-Y0)/YA
630 ' *** Errechnen der Multiplikatoren
640 YF=ABS(YF):IFX0<0THENXF=XM/XBELSEIFX0>XM
THENXF=XM/XBELSEIFX0=XMTHENXF=XM/XBELSEIFX0=
0THENXF=XM/XBELSEIFX0>(XM-X0)THENXF=X0/XAELS
EXF=(XM-X0)/XE
650 XF=ABS(XF)
660 IFMA=1ANDXF>YFTHENXF=YFELSEIFMA=1ANDYF>X
FTHENYF=XF
670 IFMA=1THENXF=XF/ZF
680 IFQF=5THENRETURN
690 GOSUB1120:ONERRORGOTO1440
```



Formel: Y=(K0*SIN(X+K1))
kartesische Koordinaten

```
700 ' *** Funktionswerte berechnen etc
710 X=SX
720 GOSUB50:IFKO=1THENX1=Y*COS(X):Y1=Y*SIN(X
)ELSEX1=X:Y1=Y
730 IFWA=1THENGOSUB1580
740 X2=X0+(X1*XF):Y2=Y0-(Y1*YF):IFX2<00RX2>X
MORY2<00RY2>YMTHENGOTO760ELSEPSET(X2,Y2),C1
750 IFJ<>1THEN760ELSEJ2=INT(Y2/J1):J2=ABS(J2
-45)+15:J$="164n"+STR$(J2):PLAY J$
760 X=X+WI:IFINKEY$=""THEN720
770 FORZ=0TO13:FORZ1=1TOLEN(E(Z)):LOCATE6*(Z
1-1)+12,184:COLORC2:PRINT"I";:LOCATE6*(Z1-1)
+12,184:COLORC1:PRINTMID$(E(Z),Z1,1);:GOSUB7
90:NEXTZ1:Z2=38-LEN(E(Z)):COLORC2:PRINTSTRIN
G$(Z2,201);:FORT=1TO500:NEXTT:GOSUB790:NEXTZ
:GOTO770
780 ' *** Tastaturabfrage bei Zeichnen
790 E=INKEY$:IFE=""THENRETURNELSEIF(E<"0"ORE
>"9")ANDE<>">"ANDE<>"f"ANDE<>"F"ANDE<>"T"AND
E<>"t"ANDE<>"M"ANDE<>"m"THENRETURNELSEGOSUB1
560:IFE=">"THEN1190ELSEIFE="F"ORE="f"THEN125
0ELSEIFE="T"ORE="t"THEN1300
800 IFE="M"ORE="m"THEN1340ELSEONVAL(E)+1 GOT
0720,830,860,930,1000,1020,1040,1080,1110,11
60
810 STOP
820 ' *** Eingabe: Neuer Startwert fuer X
830 LOCATE12,184:PRINT"X =";X:LOCATE102,184:
COLORC2:PRINTSTRING$(24,201);:COLORC1:LOCATE
126,184:PRINT"Neu : ";:M=162:Z5=12:F3="12345
67890.-+":GOSUB1500:IFF2=""THEN840ELSEX=VAL(
F2)
840 GOSUB1560:GOTO770
850 ' *** Eingabe: Neuer Wertebereich X
860 LOCATE12,184:PRINT"XA =";XA:LOCATE102,18
4:COLORC2:PRINTSTRING$(24,201);:COLORC1:LOCA
TE126,184:PRINT"Neu : ";:M=162:Z5=12:F3="123
4567890.-+":GOSUB1500:IFF2=""THEN870ELSEXA=V
AL(F2)
870 GOSUB1560:LOCATE12,184:PRINT"XE =";XE:LO
CATE102,184:COLORC2:PRINTSTRING$(24,201);:CO
LORC1:LOCATE126,184:PRINT"Neu : ";:M=162:Z5=
12:F3="1234567890.-+":GOSUB1500:IFF2=""THEN8
80ELSEXE=VAL(F2)
880 IFXA>XETHENSWAPXA,XE
890 IFXA=XETHEN860
900 IFMA=1THENYA=XA:YE=XE
910 GOSUB1560:QF=5:GOSUB590:QF=0:GOTO770
920 ' *** Eingabe: Neuer Wertebereich Y
930 LOCATE12,184:PRINT"YA =";YA:LOCATE102,18
4:COLORC2:PRINTSTRING$(24,201);:COLORC1:LOCA
TE126,184:PRINT"Neu : ";:M=162:Z5=12:F3="123
4567890.-+":GOSUB1500:IFF2=""THEN940ELSEYA=V
AL(F2)
940 GOSUB1560:LOCATE12,184:PRINT"YE =";YE:LO
CATE102,184:COLORC2:PRINTSTRING$(24,201);:CO
LORC1:LOCATE126,184:PRINT"Neu : ";:M=162:Z5=
12:F3="1234567890.-+":GOSUB1500:IFF2=""THEN8
80ELSEYE=VAL(F2)
950 IFYA>YETHENSWAPYA,YE
960 IFYA=YETHEN930
970 IFMA=1THENXA=YA:XE=YE
980 GOSUB1560:QF=5:GOSUB590:QF=0:GOTO770
990 ' *** Eingabe: Neues Werteinkrement
1000 LOCATE12,184:PRINT"WI =";WI:LOCATE102,1
84:COLORC2:PRINTSTRING$(24,201);:COLORC1:LOC
ATE126,184:PRINT"Neu : ";:M=162:Z5=10:F3="12
34567890.":GOSUB1500:IFF2=""THEN1010ELSEWI=V
AL(F2)
1010 GOSUB1560:GOTO770
1020 CLS:GOTO770
1030 ' *** Werteanzeige  An / Aus
1040 LOCATE12,184:PRINT"Wertanzeige : 1=An /
 2=Aus";
1050 E=INPUT$(1):IFE="1"THENWA=1ELSEIFE="2"T
HENWA=2ELSE1050
1060 GOSUB1560:GOTO770
1070 ' ***  Konstanten aendern
1080 FORT5=0TO9:GOSUB1560:LOCATE12,184:PRINT
"K";MID$(STR$(T5),2,1);" =";K(T5);:LOCATE102
,184:COLORC2:PRINTSTRING$(24,201);:COLORC1:L
OCATE126,184:PRINT"Neu : ";:M=162:Z5=12:F3="
1234567890.+->":GOSUB1500:IFF2<>""ANDF2<>">"
THENK(T5)=VAL(F2)
1090 IFF2=">"THENGOSUB1560:GOSUB1480:GOTO770
ELSENEXTT5:GOSUB1560:GOSUB1480:GOTO770
1100 ' *** Koordinatenachsen zeichnen
1110 KA=1:GOSUB1120:GOTO770
1120 IFKA=1AND(X0>=0ANDX0<=XM)THENLINE(X0,0)
-(X0,YM)
1130 IFKA=1AND(Y0>=0ANDY0<=YM)THENLINE(0,Y0)
-(XM,Y0)
1140 RETURN
1150 ' *** Neue Funktion eingeben
1160 E="":F3="":L=0:SX=0:WI=0:XA$="":XE$="":
YA$="":YE$="":IFC3=4THENC1=15:C2=4:C3=5
1170 COLORC1,C2,C3:GOTO110
```



Formel: Y=X
polare Koordinaten

```
1180 ' *** Bild auf Kassette speichern
1190 LOCATE12,184:PRINT"Untertitel: ";:M=78:
Z5=30:F3=" ()Xx.+-*/^\Kk0123456789SINsinCOco
TAta@RqrGgL1EePpBbFfMmDdHhJjUuVvWwYyZz<>#!%"
:GOSUB1500:GOSUB1560
1200 LOCATE12,184:E="Bitte Record und Play d
ruecken...":FORZ=1TOLEN(E):PRINTMID$(E,Z,1);
:NEXTZ:FORT=1TO500:MOTOROFF:NEXTT:GOSUB1560
1210 LOCATE12,184:MOTOROFF:PRINT"Wenn bereit
, bitte > Taste druecken.";:MOTOROFF
1220 E=INKEY$:IFE<>">"THENMOTOROFF:GOTO1220
1230 GOSUB1560:LOCATE12,184:PRINTF2;:CSAVE"B
ild",S:GOTO770
1240 ' ***  Funktion Auflisten
1250 IFLEN(F)<35THENLOCATE12,184:PRINT"Y = "
;F;:E=INPUT$(1):GOSUB1560:GOTO770
1260 IFLEN(F)<71THENLOCATE12,184:PRINT"Y = "
;LEFT$(F,33);"...";:E=INPUT$(1):GOSUB1560:LO
CATE12,184:PRINT"...";MID$(F,34,37);:E=INPUT
$(1):GOSUB1560:GOTO770
1270 IFLEN(F)<105THENLOCATE12,184:PRINT"Y =
";LEFT$(F,33);"...";:E=INPUT$(1):GOSUB1560:L
OCATE12,184:PRINT"...";MID$(F,34,34);'...";:
E=INPUT$(1):GOSUB1560:LOCATE12,184:PRINT"...
";MID$(F,68,37);:E=INPUT$(1):GOSUB1560:GOTO7
70
1280 LOCATE12,184:PRINT"Tut mir leid, Funkti
on zu lang.":E=INPUT$(1):GOSUB1560:GOTO770
1290 ' *** Tonausgabe An / Aus
1300 LOCATE12,184:PRINT"Ton : 1=An / 2=Aus";
1310 E=INPUT$(1):IFE="1"THENJ=1ELSEIFE="2"TH
ENJ=2ELSE1310
1320 GOSUB1560:GOTO770
1330 ' *** Gleicher Masstab  An / Aus
1340 LOCATE12,184:PRINT"Gleicher Masstab : 1
=An / 2=Aus";
1350 E=INPUT$(1):IFE="1"THENMA=1ELSEIFE="2"T
HENMA=2:GOTO1380ELSE1350
1360 IFXA<YATHENYA=XAELSEXA=YA
1370 IFXE>YETHENYE=XEELSEXE=YE
1380 GOSUB1560:0F=5:GOSUB590:0F=0:GOTO770
1390   *** Unterroutine fuer Ueberschrift
1400 SCREEN0:PRINTCHR$(27);"p      ***  Funkt
ionenplotter  ***      ";CHR$(27);"q";:RETURN
1410 ' *** Unterroutine fuer Tastendruck
1420 LOCATE,,0:PRINT"Bitte eine Taste drueck
en...";:E=INPUT$(1):LOCATE,,1:RETURN
1430 ' *** Behandlung von Fehlern 1
1440 IFERL=50AND(ERR=5 OR ERR=11)THEN1450ELS
ESCREEN0:PRINT"Fehler in Zeile : ";ERL:PRINT
"Code : ";ERR:STOP
1450 IFWA<>1THENRESUME760
1460 EW="("+STR$(X)+" / Invalid )":GOSUB1590
:RESUME760
1470 ' *** Anpassung der Konstanten
1480 K0=K(0):K1=K(1):K2=K(2):K3=K(3):K4=K(4)
:K5=K(5):K6=K(6):K7=K(7):K8=K(8):K9=K(9):RET
URN
1490 ' *** Eingabe fuer Screen 1 etc.
1500 F1="":F2="":LOCATEM,184:PRINT"I";
1510 F1=INPUT$(1):IFF1=CHR$(8)THEN1530ELSEIF
F1=CHR$(13)THEN1560ELSEIFLEN(F2)=Z5THEN1510E
LSEIFINSTR(F3,F1)THEN1520ELSEGOTO1510
1520 F2=F2+F1:LOCATE(LEN(F2)-1)*6+M,184:COLO
RC2:PRINT"I";:LOCATE(LEN(F2)-1)*6+M,184:COLO
RC1:PRINTF1;"I";:GOTO1510
1530 IFLEN(F2)=1THENF2="":COLORC2:LOCATEM,18
4:PRINT"II";:COLORC1:LOCATEM,184:PRINT"I";:G
OTO1510ELSEIFF2=""THEN1510
1540 F2=LEFT$(F2,LEN(F2)-1):COLORC2:LOCATELE
N(F2)*6+M,184:PRINT"II";:COLORC1:LOCATELEN(F
2)*6+M,184:PRINT"I";:GOTO1510
1550 ' *** Unterste Zeile loeschen
1560 COLORC2:LOCATE0,184:PRINTSTRING$(42,201
);:COLORC1:RETURN
1570 ' *** Werteanzeige fuer X und Y
1580 EW="("+STR$(X)+" / "+STR$(Y)+")"
1590 FORR=1TOLEN(EW):LOCATE6*(R-1)+12,184:CO
LORC2:PRINT"II";:LOCATE6*(R-1)+12,184:COLORC
1:PRINTMID$(EW,R,1);"<";:NEXTR:R1=38-LEN(EW)
:COLORC2:PRINTSTRING$(R1,201);:LOCATE6*(R-1)
+12,184:COLORC2:PRINT"I";:COLORC1:RETURN
1600 ' *** Behandlung von Fehlern 2
1610 IFERR=2ANDERL=50THENRESUME1630ELSEIFERL
=50AND(ERR=5 OR ERR=11)THENRESUMENEXTELSEPRI
NT"Fehler in Zeile : ";ERL:PRINT"Code : ";ER
R:STOP
1620 COLOR15,4,5:SCREEN0:LOCATE0,0,1:END
1630 CLS:PRINT"Syntax error in Funktion :":P
RINT:PRINT:PRINTF:PRINT:PRINT
1640 GOSUB1420:GOTO220
1650 ' *** Anzeigen der Instruktionen
1660 RESTORE1660:FOR1=1TO6:GOSUB1400:LOCATE0
,4:FORT=4TO22:READE:IFE="@"THENPRINTELSEIFE=
"$"THEN1680ELSEPRINT" "E
1670 NEXTT
1680 PRINT" ";:GOSUB1420:NEXTI
1690 DATADas Programm "Funktionenplotter","e
rmoeglicht es Ihnen, eine Funktion", Ihrer W
ahl am Bildschirm graphisch, darzustellen.,@
1700 DATA Zuerst koennen Sie zwischen,dem 'P
olaren' und dem 'Kartesischen', Koordinatens
ystem waehlen.,@
1710 DATA Danach erfolgt die Eingabe der, Fu
nktion.,@,"Haben Sie 'polar' gewaehlt, dann"
, steht 'y' fuer den Radius r und, x fuer den
Winkel Phi.
1720 DATA@,@,$
1730 DATA@,Bei der Eingabe der Funktion ist,
auch die Verwendung von Konstanten,erlaubt.,
@,Diese sind k0 - k9 und,koennen waehrend de
s Zeichenvorgangs
1740 DATAgeaendert werden.,@,"z.B : Y = K4 *
 Sin (X)',@,ermoeglicht das Zeichnen von bel
iebig  vielen Sinusschwingungen mit,verschie
dener Amplitude in einer,Zeichnung.
1750 DATA@,@,$
1760 DATA"Folgende Eingaben sind erlaubt :",
@,"Rechenzeichen :,+ - * / \ ^",@,"Logische
Ausdruecke :",> < = Or And Xor Eqv Imp Not,@
,"Mathematische Funktionen :
1770 DATASin Cos Tan Atn Sqr Log,Exp Int Fix
 Sgn Abs Rnd,@,"Sowie :","Klammern : ( ) , <
Leerzeichen> und .",@,"Variablendefinitionen
 : % ! #",@,@,$
1780 DATA@,Als naechstes folgt die Eingabe d
er,Werte der einzelnen Konstanten.,@,Bei ein
er polaren Funktion wird dann,der Startwert
fuer X eingegeben.,@,Bei einer kartesischen
setzt ihn
1790 DATAder Computer selbst.,@,Als naechste
s folgt die Eingabe,"des Werteinkrements, da
s den Faktor","angibt, um den der X-Wert jew
eils",erhoeht wird.
1800 DATA@,@,$
1810 DATA@,"Wird eine Werteanzeige gewuensch
t,",wird der jeweilige X- und Y-Wert,waehren
d des Zeichnens unten am,Bildschirm angezeig
t.,@,Auch die Anzeige der Koordinatenachsen
ist wahlfrei.
1820 DATA@,Dann muessen Sie noch den,Wertebe
reich fuer X festlegen.,Er gibt den fuer Sie
 sichtbaren,Ausschnitt aus dem Funktionsgrap
hen,an.,@
1830 DATA@,@,$
1840 DATADie Wahl des Wertebereichs fuer Y,k
oennen Sie auch dem Computer,ueberlassen.,@,
@,Wird waehrend des Zeichenvorgangs,"irgende
ine Taste gedrueckt, so",werden am unteren B
ildschirmrand
1850 DATAalle erlaubten Befehle angezeigt.,@
,Wird bei der Eingabe der Konstanten,waehren
d des Zeichenvorgangs die,"Taste '>' gedruec
kt, dann bleibt ",der Wert aller folgenden K
onstanten,unveraendert.,@,@,$
1860 GOSUB1400:LOCATE1,3:PRINT"Befehlssatz w
aehrend dem Zeichnen :":PRINT:PRINT:FORZ=0TO
13:PRINTTAB(3);E(Z):NEXTZ:PRINT:PRINT:PRINTT
AB(3);:GOSUB1420
1870 CLS:LOCATE0,8,0:PRINT"Viel Spass beim F
unktionenzeichnen!!!":FORT=1TO1000:NEXTT:RET
URN
```

Auf der nächsten Seite befindet sich das Assemblerlisting des Maschincodeprogramms und das dazugehörige Hexdump. Das MC-Programm dient auf der einen Seite als Loader, es installiert einen Teil von sich selbst ab 8000H und lädt das BASIC-Programm ab 8140H in den Speicher. Auf der anderen Seite ist es mit der Rechenroutine aus Heft 6/84 ident.

Abspeichern muß man es mit:

```
BSAVE"'Name'"&HC000,&HC100,&HC03B.
```

```
100 REM          ORG C000H
101 REM    ;RT1: codiert String zu einer
102 REM    ;     BASIC-Zeile
103 REM    ;RT2: Fehler 'Type mismatch'
104 REM    ;RT3: berechnet Wert aus Zeile
105 REM    ;RT4: wandelt Ergebnis in DBL-
106 REM    ;Zahl um
107 REM    ;RT5: Platz fuer 'BASIC-Zeile'
108 REM    ;RT6: Type des Ergebnisses
109 REM    ;RT7: BASIC-Akkumulator
110 REM    ;BASANF: Adresse, ab der wir das
111 REM    ;BASIC-Programm abgelegen.
112 REM    ;BASBEG: gibt an, ab wann das
113 REM    ;BASIC-Programm abgelegt wird.
114 REM    ;LINBUF: Eingabebuffer
115 REM    ;RUNLIN: Romroutine, führt BASIC-
116 REM    ;Zeile aus.
117 REM RT1       EQU 0B44H
118 REM RT2       EQU 0905H
119 REM RT3       EQU 14CAH
120 REM RT4       EQU 5765H
121 REM RT5       EQU F54FH
122 REM RT6       EQU F793H
123 REM RT7       EQU F923H
124 REM BASANF    EQU 8140H
125 REM BASBEG    EQU F54AH
126 REM LINBUF    EQU F68EH
127 REM RUNLIN    EQU 165H
128 REM           CP 3
129 REM           JP NZ,RT2
130 REM           INC HL
131 REM           INC HL
132 REM           LD E,(HL)
133 REM           INC HL
134 REM           LD D,(HL)
135 REM           EX DE,HL
136 REM           LD C,(HL)
137 REM           LD B,0
138 REM           INC HL
139 REM           LD E,(HL)
140 REM           INC HL
141 REM           LD D,(HL)
142 REM           EX DE,HL
143 REM           PUSH BC
144 REM           PUSH HL
145 REM           PUSH DE
146 REM           LD DE,STRING
147 REM           LDIR
148 REM           EX DE,HL
149 REM           LD (HL),0
150 REM           POP DE
151 REM           POP HL
152 REM           POP BC
153 REM           LD DE,0
154 REM           LD HL,STRING
155 REM           CALL RT1
156 REM           LD HL,RT5
157 REM           CALL RT3
158 REM           CALL RT4
159 REM           LD A,(RT6)
160 REM           LD HL,(RT7)
161 REM           RET
162 REM STRING    DEFB 00
163 REM BEGIN     LD HL,C000H
164 REM           LD DE,8000H
165 REM           LD BC,003AH
166 REM           LDIR
167 REM           LD HL,BASANF
168 REM           LD (BASBEG),HL
169 REM           DEC HL
170 REM           LD (HL),0
171 REM           LD HL,0
172 REM           LD HL,CLOAD
173 REM           LD DE,LINBUF
174 REM           LD BC,CLDEND-CLOAD+1
175 REM           LDIR
176 REM           LD HL,LINBUF-1
177 REM           JP RUNLIN
178 REM     ;CLOAD: BASIC-Zeile, die abge-
179 REM     ;arbeitet werden soll, wird hier
180 REM     ;abgelegt. Abschluß mit Null.
181 REM     ;Länge in BC eintragen!
182 REM CLOAD     DEFM "LOAD"
183 REM           DEFB 34
184 REM           DEFM "1:FUNPLT"
185 REM           DEFB 34
186 REM           DEFM ",R"
187 REM CLDEND    DEFB 0
188 REM           END
```

```
C000 FE 03 C2 05 09 23 23 5E .....##^
C008 23 56 EB 4E 06 00 23 5E #V.N..#^
C010 23 56 EB C5 E5 D5 11 3A #V.....:
C018 C0 ED B0 EB 36 00 D1 E1 ....6...
C020 C1 11 00 00 21 3A C0 CD ....!:..
C028 44 0B 21 4F F5 CD CA 14 D.!O....
C030 CD 65 57 3A 93 F7 2A 23 .eW:..*#
C038 F9 C9 00 21 00 C0 11 00 ...!....
C040 80 01 3A 00 ED B0 21 40 ..:...!@
C048 81 22 4A F5 2B 36 00 21 ."J.+6.!
C050 00 00 21 63 C0 11 8E F6 ..!c....
C058 01 11 00 ED B0 21 8D F6 .....!..
C060 C3 65 01 4C 4F 41 44 22 .e.LOAD"
C068 31 3A 46 55 4E 50 4C 54 1:FUNPLT
C070 22 2C 52 00 74 75 76 77 ",R.tuvw
```

****
*
* Ergänzung für CURSOR.COM
*
****

Wie erst jetzt bekannt wurde, kann es beim Programm CURSOR (wurde in Heft 4/85 veröffentlicht) unter dem Betriebssystem 2.23 zu Fehlfunktionen kommen. Dies äußert sich dadurch, daß bei einem Aufruf von CURSOR ohne Angabe von Parametern der Cursor am Monitor komplett ausgeschaltet wird.

Die Ursache liegt in der Initialisierung der 80-Zeichenkarte durch das Betriebssystem. Darin wird die Cursor-Endzeile (der Zeichenmatrix) nämlich auf 7 eingestellt, das Programm erwartet allerdings das Cursorende in Zeile 8.

Eine Behebung dieses Fehlers kann auf zwei Arten erfolgen:

Zum Ersten kann man die Tabelle für die Monitorinitialisierung ändern (dies hat den Vorteil, daß dabei gleichzeitig ein stabiles Bild erzeugt wird), die andere Möglichkeit liegt in der Änderung des ersten Befehls im Programm CURSOR.

Ich möchte nun erklären, wie die Änderung im Programm zu erfolgen hat. Diese muß man mit einem Debugger (in meinem Beispiel ZSID80) folgendermaßen durchführen:

Aufruf des Debuggers und laden des Programms CURSOR.

```
ZSID80 CURSOR.COM
```

Ändern des ersten Assemblerbefehls

```
A100
LD   B,67
.
```

Verlassen des Debuggers und Rückkehr in das Betriebssystem

```
CTRL-C  oder  G0
```

Abspeichern des geänderten Programms CURSOR

```
SAVE 1 CURSOR.COM
```

Diese Änderung bewirkt, daß bei der Betriebsart UNDERLINE-Cursor die Anfangszeile auf 7 gesetzt wird. Danach sollte diese Option ebenfalls klaglos funktionieren.

Eine Beschreibung über die Änderung in der Initialisierungstabelle kann gemeinsam mit einem Bericht über die Verkürzung der Nachlaufzeit der Diskettenlaufwerke dem Artikel 'Kürzere Nachlaufzeiten für Diskettenlaufwerke' entnommen werden.

Rotschek Wolfgang

Leserbriefe

Frage: Gibt es einen Befehl im MSX-BASIC des SVI-728, der ähnlich dem 'SAVE "Name",S' des SVI-328 Bilder abspeichern kann?

Antwort: Es gibt ihn laut Philips Handbuch in der Form BSAVE "Name",'A','B',S. 'A' und 'B' sind die Start- und die Endadresse im Video-RAM. Dieser Befehl soll jedoch nur mit Diskette funktionieren. Da derzeit keine Diskettenstation in der Redaktion war, konnten wir dies nicht nachprüfen. Zu laden ist das Bild durch BLOAD"Name",S. Der Name enthält dabei auch die Kennung des Diskettenlaufwerkes. Sobald eine Disk in die Redaktion geliefert wird, werden wir auch diesen Befehl genauer überprüfen. Auf Kassette funktioniert dieser Befehl nicht (davon konnten wir uns schon überzeugen).

Für Kassette muß man sich mit einem kleinen (Unter-) Programm behelfen, das das Video-RAM in den Speicher transferiert:

```
1000 REM VRAM ABSPEICHERN
1010 FOR I=0 TO 16383
1020 POKE I+&HA800, VPEEK(I):NEXT
1030 BSAVE "CAS:BILD",&HA800,&HE800
1040 RETURN
```

Hereinladen kann man das Ganze mit der folgenden Subroutine:

```
1100 REM BILD EINLESEN
1110 BLOAD "CAS:BILD"
1120 FOR I=0 TO 16383
1130 VPOKE I, VPEEK(I+&HA800):NEXT
1140 RETURN
```

Angesprungen werden die Routinen mit GOSUB, wie gehabt. Bediener, die schon mit Maschincode arbeiten, können das Programm auch leicht in MC-Code abfassen, der Computer wird das Bild dann deutlich schneller abspeichern, da die lange Blocktransferierphase verkürzt wird.

Frage: Gibt es eine Möglichkeit, MSX-Programme auf dem SVI-328 und umgekehrt zu laden, wenn ja, welche?

Antwort: Es gibt ein Programm, mit welchem man die Baudrate der Ladegeschwindigkeiten verändern kann. Das jeweilige Programm muß im ASCII-Code abgespeichert werden und kann dann transferiert werden. Das Programm wurde von einem unserer Clubmitglieder geschrieben. Wir werden es in der nächsten Ausgabe veröffentlichen.

Frage: Wie kann man in die Bank 22 switchen, ohne SWITCH zu verwenden?

Antwort: Bevor wir genau erklären, wie das vor sich geht, möchten wir darauf hinweisen, daß dies nur dann geht, wenn eine 64K-RAM-Karte vorhanden ist, die folgende Schalterstellung aufweist:

S21	OFF
S22	ON
S31	ON
S32	OFF
S02	OFF
S48/32	OFF

Wenn der Computer sich mit etwa 58000 Bytes freien Speicher meldet, kann das Experiment beginnen. Laut Heft 5/85 muß ja das Register 15 im PSG folgenden Wert haben

```
 1    1    X    1    1    0    1    1
ROM0 ROM1 CAPS BK32 BK31 BK22 BK21 BK01
```

Die LED bei CAPS-LOCK stellen wir der Einfachheit halber auf 1. Dann ergibt sich das Byte &HFB. Im BASIC kann man mit der Zeile 'OUT &H88,15:OUT &H8C,&HFB' in die Bank 22 switchen. Dies ist natürlich kein lupenreines Kommando, da sich der Interpreter nach dem switchen im RAM nicht mehr auskennt, da seine Systemvariablen nicht mehr stimmen. Er steigt daher mit einer (manchmal unsinnigen) Fehlermeldung aus.

Der BASIC-Interpreter liefert uns angenehmere Möglichkeiten, um in die Bank 22 zu kommen. Man kann, wenn man per Maschincode die Bank ansprechen möchte, einen ROM-Sprung in die SWITCH-Routine durchführen. Der Computer hat SWITCH ab der Adresse 337FH abgelegt. Die Routine selber beginnt bei 3387H, A muß dann schon 0 enthalten. Bei 1 wird ein SWITCH STOP ausgeführt. Der Interpreter arbeitet dann wie nach dem Befehl SWITCH und landet, das ist der einzige Nachteil, wie SWITCH im BASIC der zweiten Bank.

Will man über Maschincode in die zweite Bank gelangen, kann man die Routine &H5F verwenden. In B übergibt man das gewünschte Port, HL enthält die Adresse, zu der man in der zweiten Bank will. Die Interrupts sind übrigens nach dem SWITCH freigegeben. Wenn die RAM-Karte schon durch ein SWITCH installiert wurde, steht in der Speicherstelle FE76H die Adresse des Stackpointers, die gebraucht wird (sowohl in Bank 02 als auch in Bank 22). Man kann also für die jeweilige Bank immer ins BASIC zurückkehren, wenn man den Stackpointer von FE76 holt. Will man über Maschincode initialisieren, kann man dies mit der ROM-Routine ab 7B9F, die auch vom SWITCH-Befehl gegebenenfalls verwendet wird.

Ein paar Daten können bei jedem SWITCH in den Registern des Z80A übergeben werden. Will man vom Maschincode aus ein ganzes Programm oder Tabellen in die Bank 22 mitnehmen (oder von der BK22 mitbringen), dann muß man den Umweg über die Bank 21 nehmen. Der Weg von 02 nach 22 sieht dann folgendermaßen aus:

```
ROM-Interpreter wegswitchen, dafür BK21 her
Daten und Programm von 02 nach 21 laden
02 wegswitchen, dafür 22 her
Daten und Programm nach 22 laden
ROM-Interpreter wieder herswitchen, BK21 weg
```

Um die Bank 02 von der Bank 22 zu erreichen, muß man die Schritte analog verwenden, mit dem Unterschied, daß 02 und 22 vertauscht werden müssen.

Kleinanzeigen

Verkaufe wegen Umstieg auf Festplatte
Super Expander SVI 605
(1 * 160K Laufwerk, auch mit SVI-328), ein Jahr alt, Richard Schwaninger,
Buerserberg 63, 6700 BLUDENZ

Super Expander zu verkaufen:
SVI-601 mit Floppy-Controller SVI-801 und Floppy SVI-902, Kaufdatum April 1984, sehr gepflegt, zwecks Umstieg abzugeben, nähere Informationen im Clublokal oder bei der Redaktion. VB 10000 Schilling

Auskünfte und die Weiterleitung von Anfragen in Sachen "Kleinanzeigen" übernimmt die Redaktion des SVI-Journals. Clubmitglieder können Kleinanzeigen (bis zu sechs Zeilen) kostenlos im SVI-Journal veröffentlichen. Für andere Personen kostet eine Kleinanzeige 50 Schilling.

Spectra Video Club Austria

Was bietet Ihnen der Spectra Video Club Austria?

- regelmäßige Clubabende mit Gelegenheit zum Informaitonsaustausch
- Möglichkeit zum kostenlosen Arbeiten an SVI-Computern während der Clubtreffen und zum Ausdrucken von Programmlistings
- außerordentliche Clubabende mit Vorträgen über Themen rund um Hard- und Software der Spectravideo-Computer und über MSX
- kostenloser Bezug der monatlich erscheinenden Clubzeitschrift SVI-JOURNAL
- verbilligter Einkauf von Spectravideo-, MSX- und Bondwell-Produkten
- NEU! Bondwell- und MSX-Besitzer sind ebenfalls willkommen.

Mitgliedsbeitrag: Jahresbeitrag S 500,-
für Schüler, Studenten, Lehrlinge S 250,-

Nähere Informationen beim

Spectra Video Club Austria
c/o Computer-Studio
1040 Wien, Paniglgasse 18-20
Telefon (0222) 65 88 93

IMPRESSUM:

Chefredakteur: Gerhard Fally

Ständige freie Mitarbeiter: Constantin Gaganas, Philipp Ott, Rafael Razim, Wolfgang Rotschek, Christoph Sator, Heinz Schmid, Stephan Traxler, Georg Wolfbauer

Medieninhaber (Verleger): Spectra Video Club Austria, p.A. Computer-Studio, A-1040 Wien, Paniglgasse 18-20, Tel (0222) 65 88 93

Hersteller: HTU-Wirtschaftsbetriebe Ges. m. b. H., 1040 Wien

Herausgeber: Spectra Video Club Austria, p.A. Computer-Studio, A-1040 Wien, Paniglgasse 18-20, Tel. 65 88 93

Erscheinungsweise: monatlich, jeweils zur Monatsmitte, Einzelheft S 15,-

Abonnementpreise:
jährlich S 150,-
halbjährlich S 80,-

Erscheinungsort Wien
Verlagspostamt 1040 Wien

Bankverbindung:
Creditanstalt: 0964-34147/00
(Spectra Video Club Austria)