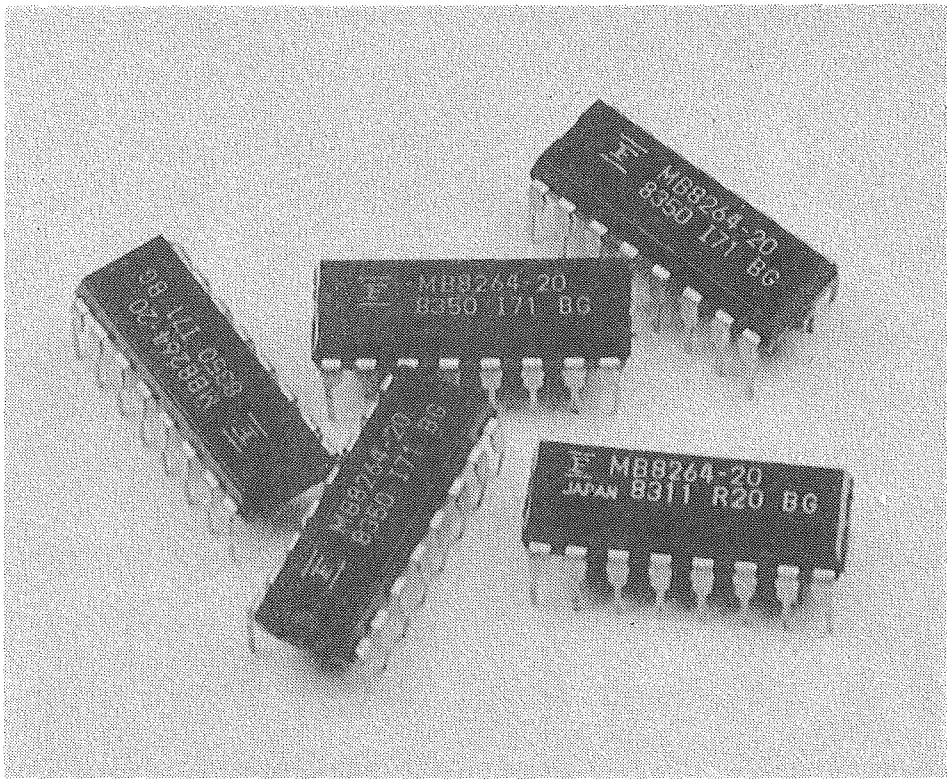
SVI JOURNAL

Die Zeitschrift des Spectra-Video-Club Austria

Mehr BASIC-Speicher

für die SVI-Computer



März 1985

Heft 3/85

S 15.-

Lieber SVI-Journal-Leser!

Wir haben schon wieder erweitert. Die Informationen für die SVI-Computer kommen nun in Hülle und Fülle zu uns. Die 24 Seiten vom letzten Mal haben nicht gereicht, wir mußten schon auf 28 Seiten ausdehnen. Trotzdem hoffen wir noch immer auf rege Teilnahme in unserer Zeitschrift. Wenn Sie also irgendein interessantes Programm auf Lager haben oder eine kleine Serie schreiben wollen, dann teilen Sie uns dies mit.

Im Zuge unserer Neuerungen haben wir uns ebenso entschlossen, das Inhaltsverzeichnis ins Innere des Heftes zu holen. Nun haben wir sowohl für das "Directory" unseres Heftes als auch für die Headlines am Titelblatt mehr Platz zur Verfügung. Es ist nicht mehr notwendig, alle Programme lakonisch mit dem Titel "Programme" zu versehen, sondern wir können Ihnen die genauen Überschriften bieten. Neu ist übrigens auch unser Bestreben, viel über den SVI-728 zu schreiben. Da aber noch sehr wenig Personen diesen Computer besitzen, bitten wir gerade die Leser mit einem SVI-728, daß sie uns Informationen liefern.

Sehr schmeichelhaft für uns war das Angebot des Generalimporteurs für Österreich, der Firma Monacor. Ab sofort bekommt jeder neugekaufte Computer einen Sammelband mit allen Heften des vorigen Jahres beigelegt. Da das erste Heft schon vergriffen ist, und das zweite in absehbarer Zeit ebenso aus sein wird, mußte dafür natürlich eine Neuauflage unserer Zeitschrift erzeugt werden.

In diesem Heft wird eine Story Wirklichkeit, von der man schon lange geredet hat. Wir zeigen, wie die Software-Speichererweiterungen funktionieren. Bekanntlich haben findige Programmierer auch die zweiten 32K RAM der Grundversion fürs BASIC erreichbar gemacht, die ja bis dahin ein "nutzloses Dasein" fristeten. Natürlich drucken wir noch ein zweites Programm ab. Das Erste wurde ja schon, wenn auch etwas versteckt, in Heft 1/85 veröffentlicht ("Befehlserweiterungen ohne CMD").

Da wir zum Teil recht lange Programme im SVI-Journal abdrucken, wollen wir nun das Disketten-/Kassettenservice einführen, das schon vor einiger Zeit in unserem Journal vorgestellt wurde.

Wir halten für Sie die Programme auf dem gewünschten Speichermedium zur Verfügung. Sie können sich nun diese Programme bei uns holen oder per Post anfordern. Natürlich kann es unsere noch kleine Redaktion nicht auf sich nehmen, die Disketten oder Kassetten zu verschenken, der Originalpreis einer Disk oder Kassette muß in diesem Fall schon berappt werden. Es gibt hier allerdings noch eine dritte Möglichkeit: Sie kommen persönlich und bringen dabei Ihre eigenen Massenspeicher mit. Im Clublokal dürfen Sie dann die "ersehnten" Programme des SVI-Journals kopieren.

Neuigkeiten auf dem SVI-Sektor gibt es übrigens auch. So soll für die Spectravideo-Computer ein Expander mit Namen SVI-609 erscheinen, mit dem die Computer netzwerkfähig werden. Dieser Expander hat zusätzlich noch ein Winchester-Festplattenlaufwerk eingebaut. Mit dieser Erweiterung werden sowohl SVI-328 als auch SVI-728 zu einem Netzwerk mit bis zu 32 Terminals und einem Zentralrechner kombiniert. Das Festplattenlaufwerk hat eine Speicherkapazität von 10 MegaByte und ist wahrscheinlich fix neben einer Floppydisk-Station untergebracht. Preise sind uns noch nicht bekannt.

Ebenso noch ganz inoffiziell ist die Information, daß ein MSX-Portable von Spectravideo auf den Markt kommen soll. Wir konnten aber bis jetzt noch nichts näheres darüber erfahren.

Die Firma Wehsner in Wien bietet seit neuester Zeit ein Service für SVI-Käufer an. Man vertauscht dort die Z- und die Y-Taste. So dürfen Leute aufatmen, die auf diesem Gerät maschinschreiben wollen. Z liegt dann also neben dem T und Y hat seinen Platz links vom X. Der Umbau erfolgt kostenlos, ist aber derzeit nur bei Neukäufen möglich.

Ihr SVI-Journal-Chefredakteur Gerhard Fally!

INHALT

Die nächsten Clubabende:

Mi, dem 20. März 1985, ab 19 Uhr
Di, dem 26. März 1985, ab 19 Uhr
Sa, dem 13. April 1985, ab 17 Uhr
Mi, dem 17. April 1985, ab 19 Uhr
Di, dem 23. April 1985, ab 19 Uhr
Sa, dem 4. Mai 1985, ab 17 Uhr

wie immer 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.

In dieser Folge widmen wir uns intensiv den Ein-/Ausgabe-Anweisungen. Wir hatten ja schon im vorigen Heft mit BLOAD und BSAVE begonnen. Wir werden aber einige Befehle auslassen, denn die Befehle zur Dateiverarbeitung wollen wir gesondert behandeln.

Ein unkomplizierter Befehl ist CLICK ON/OFF. Mit ihm können wir den Tastaturklick ein- oder ausschalten. Der Computer schaltet nach der Initialisierung diesen Klick automatisch ein. Es ist recht praktisch, ein Signal zu hören, welches einem bestätigt, daß man eine Taste gedrückt hat. Da die Spectravideo-Tastatur jedoch gute Druckpunkte hat, kann man sich auch nach seinem "Fingerspitzengefühl" richten. Besonders in einer Umgebung, in der unnötige Geräuscherzeugung nicht erwünscht ist, wird der Befehl CLICK OFF (schaltet Klick aus) sehr strapaziert werden.

Etwas interessanter ist die Kombination CLOAD/CSAVE. Wir brauchen sie ausschließlich zur Kommunikation mit Kassetten. Das Prinzip ist wieder einfach: Mit CSAVE 'Dateiname' wird ein Programm oder ein Bild abgespeichert (beim Bild wird ein ",S" an den Befehl angehängt). Mit CLOAD 'Dateiname' holt man sich sein Programm wieder in den Speicher. Setzt man hinter das Befehlswort (CLOAD) ein Fragezeichen, so wird der Inhalt des Speichers mit dem auf der Kassette verglichen. Haben wir also gerade etwas abgespeichert, dann können wir testen, ob das Programm auch wirklich fehlerfrei auf der Kassette "gelandet" ist. Kassetten sind bekanntlich nicht so zuverlässige Datenträger wie Disketten. Ein sogenanntes "Verify" (mit CLOAD? 'Dateiname') nach jedem Abspeichern ist daher sinnvoll.

CSAVE "TEST" speichert das Programm TEST
CSAVE "BRUEGH",S speichert das Bild BRUEGH
CLOAD? "TEST" vergleicht die Datei TEST mit dem BASIC-Programm im Speicher
CLOAD "TEST" lädt die Datei TEST in den Speicher

Man kann bei CLOAD den Dateinamen auch weglassen. Dann lädt der Computer das nächste Programm, das gefunden wird, herein.

Beim Spectravideo haben wir einige Möglichkeiten, unseren Bildschirm zu löschen. Wir haben einerseits SCREEN. Dieser Befehl wird etwas später erklärt. Ebenso haben wir eine Taste CLS und einen gleichnamigen Befehl. Die Anweisung CLS löscht sowohl Text-, als auch Graphikbildschirme. Der Modus bleibt der Gleiche, es verschwinden lediglich alle Zeichen und Figuren. Der Cursor wandert dabei in die linke obere Ecke des Schirms. Die Control-Sequenz "CTRL-L" bewirkt allerdings nur ein Löschen des Textbildschirms, ebenso ?CHR$(12).

Nur für Disk-BASIC wird COPY gebraucht. Mit dieser Anweisung kann man Disketten kopieren. Mit COPY 2 FROM 1 wird zum Beispiel die Disk im "Einser"-Laufwerk auf die im Laufwerk 2 kopiert. Will man nur einzelne Dateien kopieren, so stellt man hinter das FROM den Dateinamen der jeweiligen Datei, etwa: COPY 1 FROM "2:MONIT" (Programm MONIT wird vom Laufwerk 2 nach Laufwerk 1 kopiert).

Es ist sogar möglich, Sektoren zu kopieren. Hier muß man allerdings etwas mehr angeben. Das Format für den Befehl sieht dann so aus:

COPY Laufwerk; (Track,Sektor) FROM Laufwerk; (Track,Sektor)-(Track,Sektor)

Wieder wird hinter dem FROM die Quelle angegeben, während direkt hinter dem Befehlswort das Ziel steht. Man gibt folgendes an:

1) Von welchem Laufwerk wird auf welches Laufwerk kopiert?

2) Von welchem Track und von welchem Sektor wird zu kopieren angefangen?

3) Bis zu welchem Track, beziehungsweise Sektor wird "gearbeitet"?

4) Ab welchem Track, beziehungsweise Sektor wird der Inhalt am Ziel abgelegt?

Wenn wir also von Disk 1 von Track 24 die Sektoren 3-6 auf Disk 2 ab Track 2, Sektor 8 kopieren wollen, dann sieht der Befehl folgendermaßen aus:

COPY 2;(2,8) FROM 1;(24,3)-(24,6)

Beim Kopieren von Sektoren muß man natürlich aufpassen, daß keine Files vernichtet werden. Es kann nämlich nur all zu leicht passieren, daß ein Sektor auf einen Teil der Disk kopiert wird, auf dem sich schon ein Programm befindet.

Ebenso ist es verboten, von Track 0 auf andere Tracks zu kopieren, denn diese erste Track hat eine andere Formatierung.

In Österreich ist es bis jetzt leider noch nicht gestattet, das SVI-Modem zu verwenden. Aus diesem Grund ist der Befehl DIAL auch relativ unwichtig. Mit ihm kann man lediglich per Modem eine Telefonnummer wählen. Man kann zwar mit einer RS 232 und einem Akkustikkoppler ein Modem ersetzen, doch ist dies eine sehr teure Lösung. Sollte ein SVI-Modem, welches auch in Österreich erlaubt ist, auf den Markt kommen, dann werden wir uns sowieso im SVI-Journal aktuell damit und mit den notwendigen Befehlen befassen.

Es ist oft nützlich, nach dem Einschalten des Computers gleich ein Programm zu laden, welches einem bestimmte Parameter von vornherein einstellt, zum Beispiel das Verstellen der Funktionstasten. Dies geht allerdings nur mit einer Diskette. Im BASIC gibt es den Befehl IPL, der auf der Diskette einen String speichert. Dieser String wird kurz nach dem Einschalten des Computers ausgelesen und als Befehlsfolge interpretiert. Normalerweise wird man in diesem String immer ein kleines Programm aufrufen, welches dann beliebig viele andere Befehle abarbeitet. In einem einzigen String lassen sich ja bekanntlich weniger Befehle unterbringen als in einem Programm. Theoretisch darf man jedoch jeden beliebigen Befehl einspeichern. Sehen wir uns deshalb den Befehl allgemein an:

Hinter dem Befehlswort IPL wird der String angefügt, der später gebraucht wird. Die Länge dieses Strings ist lediglich durch die

255 Zeichen-Grenze beschränkt. Ähnlich den Strings der unten folgenden Funktionstasten dürfen auch hier wieder alle Zeichen, das heißt auch nicht druckbare, verwendet werden. Der Befehl IPL funktioniert nur auf der Diskette des Laufwerks 1.

Der Spectravideo ist einer der wenigen Computer unter 10.000 Schilling, der sogenannte Soft-Keys besitzt. Mit diesen frei programmierbaren Funktionstasten können Befehlsfolgen bis zu 15 Zeichen definiert werden. Viele andere Computer haben nämlich unter dem Titel "Funktionstasten" lediglich Tasten, die mit einem Zeichen belegt werden dürfen, oder umfangreiche und komplizierte Änderungsbefehle für ihre Tasten.

Beim SVI wird eine Funktionstaste mit dem Befehl KEY geändert. Hierzu gibt man zuerst das Befehlwort, danach die Nummer der Funktionstaste und zum Schluß die Zeichenfolge ein. Sollte der String länger als 15 Zeichen sein, dann werden die restlichen abgeschnitten. Auch nicht druckbare Zeichen werden verwertet. So kann man Cursor-Bewegungen, INSERT, CLS, DEL und viele andere Funktionen integrieren. Wenn man zum Beispiel auf eine Taste "REM ***" legen will, dann sieht dies so aus:

```
KEY 4,"REM ***"
```

Wollen wir diesen Text jedoch irgendwo einfügen, setzen wir ein INS davor. Hinter das Befehlswort stellen wir noch ein ENTER, damit die Zeile abgeschlossen ist:

```
KEY 4,CHR$(18)+"REM ***"+chr$(13)
```

Das INS brauchen wir nicht extra abzuschließen, es wird durch das ENTER (CHR$(13)) beendet.

Es gibt übrigens 10 Funktionstasten. Die ersten fünf werden durch Drücken von einer der fünf Tasten aktiviert, die zweiten fünf sind mit SHIFT kombiniert.

Der Inhalt der zehn Funktionstasten wird im Textmodus als Fußleiste am Bildschirm ausgegeben. Dabei sind aber aus Platzgründen nur die ersten 7 Zeichen abgebildet. Möchte man die gesamte Belegung sehen, kann man KEY LIST eingeben. Nun wird der vollständige Inhalt aller Funktionen gezeigt. Wer die 80-Zeichenkarte besitzt, hat auf der Leiste am Bildschirm übrigens alle 15 Zeichen pro Belegung liegen.

Wieder für die Handhabung mit Disk wird der Befehl KILL gebraucht. Mit ihm können wir Programme oder Files löschen. Nach KILL 'Dateiname' verschwindet die nicht mehr benötigte Datei von der Disk.

Zur Eingabe von Strings haben wir schon INPUT und INPUT$ kennengelernt. Doch beide haben ihre Nachteile. Es gibt aber noch einen anderen Befehl, der die Vorteile dieser beiden Anweisungen kombiniert (und einige Nachteile damit kompensiert). LINE INPUT ist speziell auf die Eingabe von Strings spezialisiert. Dieses Kommando gibt kein Fragezeichen aus, wie bei INPUT üblich, sondern überläßt es dem Programmierer, was er in der Stringkonstanten hinter dem Befehlswort ausgeben lassen möchte. Jedes eingegebene Zeichen wird angezeigt, der Abschluß wird mit ENTER durchgeführt.

Ebenso läßt LINE INPUT einen Beistrich im eingegebenen String zu, ohne daß man Anführungszeichen machen bräuchte. Bei INPUT fungiert das Komma ja als Trennzeichen zwischen den einzelnen Variablen. Will man bei INPUT einen Beistrich eingeben, muß man den ganzen String unter Anführungszeichen stellen. Sonst verhält sich der Befehl weitgehend analog zum INPUT.

Einige kleine Beispiele geben wir natürlich auch noch:

LINE INPUT A$: man darf in A$ Zeichen eingeben, keine Anzeige auf dem Bildschirm vor der Eingabe.
LINE INPUT "NAME";A$: "NAME" wird am Bildschirm ausgegeben, danach wartet der Computer auf die Eingabe.
LINE INPUT A$;B$ ist irregulär. Die Stringkonstante muß ein konstanter String sein.

Das MSX-BASIC ist reich an Speicher-/Ladebefehlen. Wir haben schon BLOAD und CLOAD kennengelernt, aber es gibt auch LOAD (und SAVE als Speicherkommando dazu). Mit diesem Befehl können wir sowohl auf Disk als auch auf Kassette arbeiten. Wollen wir mit der Floppy kommunizieren, müssen wir lediglich "1:" oder "2:" vor den Dateinamen stellen. Ansonsten funktioniert LOAD/SAVE genauso wie CLOAD/CSAVE.

Es gibt allerdings noch eine Option bei SAVE, die CSAVE nicht hat. Man kann ein ",A" hinter den Namen stellen, dann wird die Datei im ASCII-Code abgespeichert. Man braucht diese Art des Speicherns für den uns bekannten Befehl MERGE. Es gibt jetzt übrigens schon einen Ersatz für MERGE. In der vorigen Ausgabe des SVI-Journals wurde ein MC-Programm vorgestellt, mit dem man zwei mit CSAVE abgespeicherte Programme mischen kann, dies jedoch nur nebenbei. Tatsache ist, daß man mit SAVE Programme im ASCII-Code speichern kann, was manchmal sehr nützlich ist.

Auch LOAD hat eine Option. Wenn wir ein ",R" anhängen, wird das Programm nach dem Laden gleich gestartet. Es wird also kein RUN mehr benötigt. Beim Arbeiten mit Disketten kann man sein Programm auch mit einem anderen Befehl kombiniert laden und starten lassen. Bei RUN "1:'Dateiname'" lädt und startet der Computer das Programm.

Sehen wir uns nun einen recht interessanten Befehl an, LSET. Mit dieser Anweisung kann man Daten in einen Zwischenspeicher laden, der für eine Random-Datei gebraucht wird. Zusätzlich dazu kann man diesen Befehl auch ohne Disk-BASIC verwenden, um zum Beispiel Strings zu formatieren oder formatiert abzuspeichern.

Wir werden uns jedoch nicht mit den Geheimnissen der Random-Dateien befasen, die kommen später an die Reihe. Für uns sind momentan nur die Formatiermöglichkeiten interessant. Im übrigen gibt es auch ein Äquivalent zum LSET, nämlich das RSET. Welcher Unterschied hier besteht, das wird später geklärt.

Wenn wir einfach einen String auf eine gewisse Länge stutzen oder erweitern wollen, dann nehmen wir eine Variable, definieren sie auf die bestimmte Länge und speichern dann mittels LSET (RSET) den gewünschten String in der Puffervariablen. Der String wird daraufhin gestutzt, wenn er länger ist, oder links- (bei LSET), beziehungsweise rechtsbündig (bei RSET) abgelegt. Das Definieren auf eine bestimmte Länge kann man am Einfachsten mit folgender Formel vornehmen:

```
'Variable'=SPACE$('Anzahl')
```

In der nächsten Folge beschäftigen wir uns noch ein wenig mit diesem Befehl und seinen Marotten. Danach werden so wichtige Befehle wie PLAY, SOUND oder LOCATE erklärt.

Assemblerfortsetzungskurs Nr.: 009

Diesmal werden keine Ladebefehle sondern arithmetische Anweisungen besprochen. Bisher haben wir nur die ADD (ADC)-, SUB (SBC)- und (CP)-Funktionen kennengelernt. Doch von den arithmetischen Befehlen gibt es noch einige mehr, die das bitweise "Verknüpfen" von Registern und Daten ermöglichen. Dafür besitzt der Z80 drei Befehle: OR (Oder) AND (Und) und XOR (exklusives Oder). Obwohl damit nur 8-Bitdaten verwertbar sind, sind diese Kommandos doch ausreichend. Bei diesen Befehlen wird wie bei den schon besprochenen arithmetischen Befehlen ein Operand und ein Operator angegeben. Dabei wird dann das siebente Bit des Operators mit dem siebenten Bit des Operanden verknüpft. Daraufhin kommen die sechsten Bits dran, dann die fünften und so weiter. Doch was die drei Befehle im Einzelnen bewirken, besprechen wir jetzt.

```
AND:      0 AND 0 = 0
          0 AND 1 = 0
          1 AND 0 = 0
          1 AND 1 = 1
```

Die obere Tabelle zeigt auf, was passiert, wenn man zwei Bits mit der logischen Funktion AND verknüft. Das Ergebnis ist nur dann Eins, wenn Operand und Operator den Wert Eins haben. Hat einer von ihnen den Wert Null, so ist auch das Ergebnis Null.

```
OR:       0 OR 0 = 0
          1 OR 0 = 1
          0 OR 1 = 1
          1 OR 1 = 1
```

OR ist im Grunde genommen die Gegenoperation zu AND. Hat der Operand oder der Operator den Wert Eins, so ist auch das Ergebnis Eins. Haben jedoch beide den Wert Null, so ist auch das Ergebnis Null.

```
XOR:      0 XOR 0 = 0
          0 XOR 1 = 1
          1 XOR 0 = 1
          1 XOR 1 = 0
```

Bei XOR ist das Ergebnis nur dann Null, wenn Operand und Operator den gleichen Wert haben, das heißt, daß beide entweder Null oder Eins sind. Nur wenn die Werte von Operand und Operator nicht identisch sind, hat das Ergebnis den Wert Null.

Die AND und OR Funktionen sind vor allem in der Digitalelektronik sehr wichtig. Die meisten integrierten Schaltkreise sind nur aus solchen AND und OR Funktionen aufgebaut. Auch in der Mathematik sind solche Funktionen anzutreffen. Die "Bool'sche Algebra" beschäftigt sich mit ihnen. Hierbei gibt es nur zwei Zustände, Null und Eins. Wie sieht das nun aber mit der Verwendung von Registern aus? Wir wollen gleich einmal zwei Zahlen logisch mit AND, OR und dann mit XOR verknüpfen. Dabei sollen A und B vorgegebene Werte haben.

```
A = 00110101
B = 01011110

A AND B:      00110101  AND
              01011110  =
              --------
              00010100

A OR B:       00110101  OR
              01011110  =
              --------
              01111111

A XOR B:      00110101  XOR
              01011110  =
              --------
              01101011
```

Wozu verwendet man nun diese Befehle, außer zu komplexen Rechenroutinen? Da gibt es einen "ganzen Haufen" von Möglichkeiten. Zuerst ist noch zu sagen, daß die logischen Befehle die gleiche Syntax wie die SUB-Befehle haben, das heißt, daß alle Operationen immer den Akkumulator betreffen und daher der Akkumulator nicht angeschrieben wird. Doch nun ein paar Möglichkeiten:

1.) OR A oder AND A. Verändert den Inhalt des Akkumulators nicht. Es wird das CARRY-Flag gelöscht, und die anderen Flags werden gemäß des Inhaltes gesetzt. Anwendung: Löschen des CARRY-Flags oder zum Testen des Akkumulators, ob der Inhalt positiv, negativ oder Null ist.

2.) XOR FFh. Invertiert den Inhalt des Akkumulators. Alle gesetzten Bits werden Null, und die ungesetzten bekommen den Wert Eins.

3.) XOR A. Löscht den Inhalt des Akkumulators. Da beim "Exklusivem OR" alle Bits, die den gleichen Wert haben, nach der Verknüpfung den Wert Null haben, ist es recht einfach zu begreifen, wieso mit dem Befehl XOR A der Akkumulator den Wert Null erhält. Dabei wird der Akku mit sich selber verknüpft und daher haben auch Operand und Operator den gleichen Wert. Der Befehl XOR A bewirkt das Gleiche wie der Befehl "LD A,00", er ist nur etwas schneller und eleganter.

4.) AND: Mit diesem Befehl lassen sich recht einfach einzelne Bits im Akkumulator abfragen. Will man zum Beispiel wissen, ob das fünfte Bit im Akku gesetzt ist oder nicht, so löscht man mit dem AND-Befehl alle anderen Bits und fragt daraufhin den Akkumulator auf Null ab. Da alle nicht gesetzten Bits gelöscht werden, muß der Befehl, um das fünfte Bit zu testen, so aussehen: AND 00100000b. Im Operand bleiben nur die Bits unverändert, die im Operator gesetzt sind. In unserem Fall ist es das fünfte Bit. Es soll ja unverändert bleiben, da wir es testen wollen. War das fünfte Bit im Akkumulator gesetzt, so ist das Ergebnis 00100000b. Hat das Bit den Wert Null gehabt, so hat auch der Akku jetzt den Wert Null, und das ZERO-Flag ist gesetzt. AND wird, wie wir nun gesehen haben, dazu verwendet, bestimmte Bits im Akkumulator zu löschen und die anderen unverändert zu lassen.

5.) OR. OR ist wie schon erwähnt das Gegenstück zu AND. Die Bits, die im Operator gesetzt sind, werden auch im Operand gesetzt. Die Bits, die Null sind, werden im Operanden unberührt gelassen.

Daraus ist zu sehen, daß es mit OR nicht möglich ist Bits zu testen. OR wird verwendet, um Bits im Akkumulator auf Eins zu setzen. Um die Bits drei und sieben im Akku zu setzen, muß folgender Befehl durchgeführt werden: OR 10001000b.

6.) XOR. Dieses Kommando wird dazu verwendet, um einzelne Bits zu invertieren. Dort, wo im Operanden ein Bit auf Null gesetzt ist, wird nichts im Akkumulator verändert. Hat jedoch ein Bit im Operanden den Wert Null, so wird das Bit des Akkus, mit dem es verknüpft wird, invertiert. Daher ist es auch klar, daß XOR FFh den Akkumulator gänzlich invertiert. Jedoch ist es möglich, anstatt XOR FFh den CPL- (complement Akku) Befehl zu verwenden. Mit CPL wird der Akku invertiert, das heißt, es wird das Einerkomplement zum Akkumulator erstellt. Aber es gibt natürlich auch noch einen Befehl, der das Zweierkomplement zum Akkumlator erstellt. Dieser Befehl heißt NEG (negiere Akku). NEG betrifft wiederum nur den Akkumulator.

Normalerweise erstellen wir für neue Befehle immer ein Programm, um die neu gelernten Befehle besser verstehen zu können. Jedoch ist es etwas schwierig, ein Programm zu erstellen, das sich nur auf logische Befehle bezieht. Aber im nächsten Programm werden auch die logischen Befehle zur Verwendung kommen.

Oben wurde gerade besprochen, wie man mit Hilfe von logischen Befehlen Bits setzen, löschen und abfragen kann. Um nun ein einzelnes Bit zu setzen, zu löschen oder abzufragen, hat der Z80 eigene Befehle. Beginnen wir mit dem SET (setzen)-Kommando. Man muß hier die Bitnummer des Bits, das auf Eins gesetzt werden soll, angeben. Dabei können alle 8-Bit Register (ausgenommen natürlich das Flag-Register), sowie (IX+nn) und (IY+nn) verwendet werden.

Hat nun das Rgister L den Wert Null und will man das zweite Bit setzen, so ist dies mit folgendem Befehl möglich SET 2,B. Nun schaut das Bitmuster von B folgendermaßen aus: 00000100b.

Es ist ja nun fast schon klar, daß auch ein Gegenbefehl dazu existert. Das ist der RES (reset=zurücksetzen)-Befehl. Er ist genauso zu handhaben wie der SET Befehl. Zuerst kommt die Bitnummer und dann das Register.

Nun, wie schon gesagt wurde, gibt es auch ein Kommando zum Testen eines Bits in einem Register. Die Syntax ist dieselbe wie bei den zwei soeben erwähnten Befehlen: BIT n,r (n ist eine Zahl von 0-7, r ein Register). Der Wert des Bits n wird inventiert und dann ins ZERO-Flag geschrieben. Ist das Bit Null, so wird das ZERO-Flag gesetzt, und ist das Bit Eins, so hat das ZERO-Flag den Wert Null. Der Vorteil des BIT-Befehls gegenüber der AND-Anweisung liegt darin, daß beim BIT-Kommando der Inhalt des Akkumulators nicht verändert oder zerstört wird.

Zur besseren Übung wird ein Programm erstellt, das eine 16-Bit Adresse binär auf dem Bildschirm ausgibt.

Dieses Programm gibt die Adresse binär aus, die indirekt durch FFC0h adressiert wird. Das heißt, daß in FFC0h die Adresse steht, bei welcher wiederum die Adresse steht, die ausgegeben werden soll. Am Ende des Programms ist ein neuer Befehl erkennbar: DS 8. Dieses Kommando ist wie DB oder DW nur ein reines Assemblerkommando. Es bewirkt, daß acht Bytes an Speicher für das Label

```
100 ' Dieses Programm gibt eine 16-Bit
110 ' Adresse binaer auf den Bild-
120 ' schirm aus.
130 '
140 REM START   org  C800h
150 REM PRT_A   equ  0018h ;Ausgabe
160 REM ;eines Zeichens
170 REM ADR     equ  FFC0h ;Adresse
180 REM ;wo die auszugebende 16-Bit
190 REM ;Adresse steht
200 REM
210 REM         ld   hl,(ADR)
220 REM         inc  hl
230 REM ;(HL)=die auszugebende 16-Bit
240 REM ;Zahl
250 REM         ld   b,(hl)
260 REM         call PRT_BIN ;Printe
270 REM ;das Highbyte der Adresse
280 REM ;binaer
290 REM         dec  hl
300 REM         ld   b,(hl)
310 REM         call PRT_BIN ;Printe
320 REM ;das Lowbyte der Adresse
330 REM ;binaer
340 REM         ret
350 REM ;Beende Programm und kehre
360 REM ;ins BASIC zurueck
370 REM
380 REM
390 REM ;Die folgende Routine PRT_BIN
400 REM ;druckt das Register B binaer
410 REM ;auf dem Bildschirm aus.
420 REM
430 REM PRT_BIN ld   c,1 ;Bitzaehler
440 REM         ld   de,BUFFER
450 REM PRT_B1  ld   a,b
460 REM         and  c
470 REM         ld   a,"0"
480 REM         jr   z,PRT_B2
490 REM         ld   a,"1"
500 REM PRT_B2  ld   (de),a
510 REM         inc  de
520 REM         ld   a,c
530 REM         add  a,a ;verdopple
540 REM ;den Bitzaehler
550 REM         ld   c,a
560 REM         jr   nc,PRT_B1
570 REM
580 REM         ld   b,8 ;Schleifen-
590 REM ;zaehler zum Ausgeben des
600 REM ;Buffers von hinten
610 REM         dec  de
620 REM PRT_B3  ld   a,(de)
630 REM         call PRT_A
640 REM         dec  de
650 REM         djnz PRT_B3
660 REM         ret
670 REM
680 REM BUFFER  ds   8 ;reserviere
690 REM ;acht Bytes fuer das Label
700 REM ;BUFFER
710 REM         end
```

BUFFER freigehalten werden. DS steht für "Define Space" (definiere Raum).

Doch jetzt zur genaueren Programmbeschreibung. Am Beginn wird festgelegt, daß das Programm bei C800h beginnen soll. Zuerst wird dann nach HL die Adresse geladen, bei der das auszugebende 16-Bit Wort steht. HL wird erhöht, da ja am Bildschirm zuerst das Highbyte und dann das Lowbyte ausgegeben werden soll. Nachdem ins Register B das Highbyte geladen und ausgegeben wurde, wird das Gleiche mit dem Lowbyte gemacht.

Doch nun zum wichtigsten Teil unseres Programmes. Die Routine PRT_BIN soll das Register B binär auf dem Bildschirm darstellen. In der ersten Schleife wird das Register bitweise zerlegt und die auszugebenden Nullen und Einsen in einen Zwischenbuffer geschrieben. Der Zwischenbuffer ist notwendig, da bei der Zerlegung mit dem rechten

Fortsetzung auf Seite 13

Nach dem im vorigen Heft nur in Kurzform beschriebenen Boot-Vorgang sehen wir uns den Kalt- und Warmstart nun genauer an. Als Grundlage für alle weiteren Quelldateien wird, obwohl CP/M 2.22 am weitesten unter SVI-Benützern verbreitet ist, das CP/M 2.23 herangezogen. Der Grund liegt in der Erfassungsmöglichkeit aller bis dato von Spectravideo durchgeführten Erweiterungen. Eine sich für den Anwender einer anderen CP/M-Version als 2.23 ergebende Adreßumrechnung halte ich für ein kleineres Übel, als sich interessante Details entgehen zu lassen. Nicht zuletzt erhöht ein "Nachwassern" des Programmes Pfade, dessen Verständnis.

Nach dem Durchlauf der bereits beschriebenen Boot-Routine befindet sich nun das BIOS (Basic input/output system) im Speicher. Es erstreckt sich von Adresse E600H bis F07FH und kann mit "Grundsystem zur Ein- und Ausgabe" oder mit "hardwareabhängiger Teil des CP/M" übersetzt werden. Tatsächlich ist bei CP/M dieser Abschnitt der einzige, welcher an den Rechner angepaßt werden muß. Alle anderen Betriebssystemteile sind, abgesehen von den absoluten Adressen, bei jedem CP/M-Computer gleich.

Unter jenen, in weiteren Kapiteln beschriebenen Unterprogrammen des BIOS, befinden sich zwei, für den Systemstart lebensnotwendige Programmabschnitte. Die Kalt- und die Warmstart-Routine. Die Kaltstart-Routine wird nur einmal, und zwar beim Systemstart, die Warmstart-Routine nach einem Kaltstart und bei jedem Control-C durchlaufen.

3.3 Der Kaltstart

Die Kaltstart-Routine wird unmittelbar nach dem Verlassen der Boot-Routine, wenn das BIOS bereits im Speicher steht, angesprungen. Sie hat nicht nur die Aufgabe der Initialisierung der Hardware, wie z.B. des Video-Display-Prozessors, der 80Zeichen-Karte oder der seriellen Schnittstelle. Sie muß auch Software-Parameter setzen, die für den eigentlichen Betrieb des Computers unbedingt notwendig sind. Dazu gehört das Ablegen der Interruptadressen in den dafür vorgesehenen Speicherstellen, das Wählen der Bildschirmbreite und der Farbe für den Fernsehbetrieb, die gewünschte Funktionstastenbelegung u.a. Nicht zuletzt ist sie auch für das ordentliche Weiterreichen des Programmzählers (JP WSTART) verantwortlich.

Da es sich bei der, beim SVI-328 zur Anwendung kommenden, Interruptbehandlung um ein wohldurchdachtes, auch zeitliches Zusammenspiel von Programmteilen handelt, werden wir dieser besondere Aufmerksamkeit widmen.

Die vom Video-Display-Prozessor (VDP) alle 20 Millisekunden abgegebenen Interrupts veranlassen den Z80 das gerade bearbeitete Programm zu unterbrechen, um zu der Adresse 38H zu verzweigen und nach Abarbeitung der dort beginnenden Befehle wieder, genau wie bei einem Unterprogrammaufruf, zum ursprünglichen Programm zurückzukehren. Vorausgesetzt ist natürlich die Software-Erlaubnis, auf Interrupts überhaupt zu regieren. Siehe "DI" bzw. "EI" (Disable - Sperre bzw. Enable - Erlaube Interrupts).

Das CP/M-Betriebssystem bedient sich über die Interruptroutine eigentlich nur der Tastaturabfrage und des, durch den Kaltstart-Vorgang an diese angebundene, Unterprogramms zur Begrenzung der Nachlaufdauer der Laufwerksmotoren.

Und trotzdem, obwohl die im BASIC-ROM ruhende Interrupt-Routine auch den Zusammenstoß von Sprites und den Druck auf einen Triggerknopf registriert, den Zeitzähler erhöht und einige weitere Aufgaben erfüllt, die jedoch nur bei BASIC verarbeitet werden, wird sie angesprungen.

Die Erfassung und Entprellung der jeweils gedrückten Tasten erfolgt während mehrerer Interruptphasen, asynchron zur tatsächlichen Abfrage, und ist zeitlich genau abgestimmt. Es wäre Unsinn, durch eine neue Routine wertvollen Speicherplatz zu verschwenden, wenn man problemlos mittels "Bank-Switching" zur bereits bestehenden gelangt, zumal sowieso auch wegen der Ausgabe am Fernsehschirm bereits ins BASIC-ROM "getaucht" wird.

Um aber die im BASIC-ROM liegende Interrupt-Routine während des Normal-Betriebs zu erreichen, müssen beim Kaltstart einige Vorbereitungen getroffen werden, um beim Auftreten dieser Interrupts, den Programmverlauf in die dafür vorgesehenen Bahnen zu lenken.

Zu diesem Zweck wird bei Speicherstelle 38H, also jener Adresse die bei einem Interrupt angesprungen wird, ein Sprungbefehl zu jener Routine abgelegt, die dann ihrerseits mittels "Bank-Switching" dafür sorgt, den Interrupt-Programmverlauf in die BASIC-ROM-Interruptroutine, die sich ja auch auf Speicherstelle 38H befindet, umzuleiten.

Bei Durchsicht der nun erreichten ROM-Routine fallen zwei wichtige Austrittspunkte auf. Der erste stellt einen Unterprogrammaufruf nach FE79H dar. Dieser Aufruf findet gleich am Beginn, noch vor der Tastaturabfrage statt. Er wird aber auch bei einem einfachen, nicht durch Interrupt bedingten, in das BASIC-ROM stattfindenden Sprung in die Interrupt-Routine ausgeführt. Anschliessend aber wird die Befehlsfolge vorzeitig abgebrochen.

Da es aber noch einen zweiten Austrittspunkt gibt, der nur bei einem vorausgegangenen, durch Hardware ausgelösten Interrupt aktiv ist, bleibt der erstere mit einem schon bei der BASIC-Initialisierung gesetzten RET-Befehl praktisch unbeachtet.

Die absolute Adresse des zweiten Unterprogrammaufrufs ergibt sich durch die von der Kaltstart-Routine durchgeführten Addition von E1H zur Adresse des ersten oben erwähnten Unterprogrammaufrufs, also FF5AH. Um zur Adresse des ersten, jedoch nicht verwendeten Aufrufs zu kommen, genügt das Lesen der Speicherstellen 26H und 27H, nach einem kurzen Umschalten auf das BASIC-ROM. Diese dienen nun als Basis zur Berechnung des zweiten, verwendeten Austrittspunktes.

Der errechnete Unterprogrammaufruf aktiviert eine Routine, die einen, vor jeder Diskettenschreib- bzw. Leseoperation gesetzten Zähler regelmäßig um 1 vermindert, um bei Erreichen des Nullstandes die Laufwerksmotoren auszuschalten.

Im Folgenden finden Sie ein kommentiertes Listing der Kaltstart-Routine.

```
                        TITLE  'KALTSTART-ROUTINE FÜR DAS CP/M-BETRIEBSSYSTEM'

                               .Z80
0000'                          ASEG
                               ORG     0EF06H

0004                    CDISK  EQU     4               ;Aktuelles Laufwerk
F083                    LSTDSK EQU     0F083H          ;Zuletzt angesprochenes Laufwerk
F084                    LTRKA  EQU     0F084H          ;Zuletzt gewählte Spur von Lfwk. A
E66A                    IIOBYT EQU     0E66AH          ;I/O-Byte für Initialisierung
0003                    IOBYTE EQU     3               ;Aktuelles I/O-Byte
E87B                    CPMINT EQU     0E87BH          ;Beginn der CP/M-Interrupt-Service-Routine
0038                    INTJMP EQU     38H             ;Inhalt von INTJMP = Jump-Befehl "C3H"
0039                    INTADR EQU     39H             ;Beinhaltet Adresse von CPMINT
0026                    INTROM EQU     26H             ;Inhalt von INTROM ist die Basisadresse zur
                                                       ;Berechnung des Zeigers auf DTIMER
E8A1                    DTIMER EQU     0E8A1H          ;Kontrolliert die Nachlaufzeit der Laufwerke
E665                    I25LIN EQU     0E665H          ;Fernseher-Fußleiste (25. Zeile) ein/aus
FA06                    LINE25 EQU     0FA06H          ;Fernseher-Fußleiste ein/aus (ROM-Zugriff)
E664                    CRTWID EQU     0E664H          ;Zeichenanzahl für Fernseher
F543                    WIDCRT EQU     0F543H          ;Zeichenanzahl für Fernseher (ROM-Zugriff)
E661                    COLTAB EQU     0E661H          ;Farben für Fernseher
FA0A                    CRTCOL EQU     0FA0AH          ;Farben für Fernseher (ROM-Zugriff)
                                                       ;1.Byte:       Farbe der Zeichen
                                                       ;2.Byte:       Farbe des Hintergrundes
                                                       ;3.Byte:       Farbe des Rahmens
E675                    KEYTAB EQU     0E675H          ;Beginn der vorgewählten Keybelegung
FA1E                    KEYMEM EQU     0FA1EH          ;Beginn der aktuellen Keybelegung (ROM-Zugriff)
E84E                    BASROM EQU     0E84EH          ;Schnittstelle zum Basic-ROM
EBCE                    INI80Z EQU     0EBCEH          ;Initialisierung der 80Zeichen-Karte
EC30                    INISER EQU     0EC30H          ;Initialisierung der seriellen Schnittstelle
F06C                    MSGOUT EQU     0F06CH          ;Ausgabe-Routine für Zeichen bis 00H
EFF0                    WSTART EQU     0EFF0H          ;Warmstartadresse

EF06                    CSTART:
EF06   F3                      DI                     ;Sperre Interrupts
EF07   31 0080                 LD      SP,80H         ;Setze Stapel auf 80h

EF0A   AF                      XOR     A
EF0B   32 0004                 LD      (CDISK),A      ;Laufwerk nach CSTART ist A  (A=0, B=1)
EF0E   32 F083                 LD      (LSTDSK),A     ;Letztgewähltes Laufwerk ist A
EF11   DB 31                   IN      A,(31H)        ;Speichere letzteingestellte Spurnummer
EF13   32 F084                 LD      (LTRKA),A      ;von Laufwerk A

EF16   3A E66A                 LD      A,(IIOBYT)     ;Lese I/O-Byte für Initialisierung (INIT)
EF19   32 0003                 LD      (IOBYTE),A     ;Initialisiere I/O-Byte

EF1C   3E C3                   LD      A,0C3H         ;Setze Zeiger auf die CP/M-Interrupt-Service-
EF1E   21 E87B                 LD      HL,CPMINT      ;Routine, welche bei zugelassenen Interrupts
EF21   32 0038                 LD      (INTJMP),A     ;alle 20 Millisekunden angesprungen wird
EF24   22 0039                 LD      (INTADR),HL

EF27   3E 0F                   LD      A,0FH          ;Ermögliche Zugang zum Basic-ROM
EF29   D3 88                   OUT     (88H),A        ;und lese aus INTROM jene Adresse, die
EF2B   DB 90                   IN      A,(90H)        ;als Basis zur Berechnung der endgültigen
EF2D   F6 02                   OR      2              ;Adresse, die bei jedem Interrupt vom
EF2F   D3 8C                   OUT     (8CH),A        ;BASIC-ROM aus angesprungen wird
EF31   2A 0026                 LD      HL,(INTROM)
EF34   E6 FD                   AND     0FDH           ;Schalte wieder aufs Ram zurück
EF36   D3 8C                   OUT     (8CH),A
EF38   11 00E1                 LD      DE,0E1H        ;Bilde entgültige ROM-Interruptadresse
EF3B   19                      ADD     HL,DE          ;und lege in dieser einen Sprung zur
EF3C   36 C3                   LD      (HL),0C3H      ;Routine DTIMER ab, welche die Nachlauf-
EF3E   23                      INC     HL             ;dauer der Laufwerksmotoren kontrolliert
EF3F   11 E8A1                 LD      DE,DTIMER      ;Im Gegensatz zu der direkt aus dem Basic-
EF42   73                      LD      (HL),E         ;ROM gelesenen Adresse, wird die letztere
EF43   23                      INC     HL             ;nur bei einem durch Hardware ausgelösten
EF44   72                      LD      (HL),D         ;Sprung nach 38H = Interrupt angesprungen
EF45   AF                      XOR     A              ;Verhindere vorerst den Aufbau der Fußleiste
EF46   32 FA06                 LD      (LINE25),A

EF49   3A E664                 LD      A,(CRTWID)     ;Lese vorgewählte Zeichenanzahl für
EF4C   B7                      OR      A              ;Fernseher (INIT)
EF4D   CA EF55                 JP      Z,I40CHR
EF50   3E 27                   LD      A,39D          ;CRTWID = 00 ==> 40 ZEICHEN
EF52   C3 EF57                 JP      I39CHR         ;CRTWID > 00 ==> 39 ZEICHEN
```

```
EF55                    I40CHR:
EF55    3E 28                   LD      A,40D
EF57                    I39CHR:
EF57    32 F543                 LD      (WIDCRT),A      ;Setze gewünschte Zeichenanzahl (ROM-Zugriff)

EF5A    21 E661                 LD      HL,COLTAB       ;Transferiere Farbdaten in jene Speicher-
EF5D    11 FA0A                 LD      DE,CRTCOL       ;stellen, auf die die Screen 0-Routine im ROM
EF60    06 03                   LD      B,3             ;(Initialisierung auf Textmodus) zugreift
EF62    7E              COLTRN: LD      A,(HL)          ;COLTAB = (INIT)
EF63    12                      LD      (DE),A
EF64    23                      INC     HL
EF65    13                      INC     DE
EF66    05                      DEC     B
EF67    C2 EF62                 JP      NZ,COLTRN

EF6A    21 E675                 LD      HL,KEYTAB       ;Transferiere die Belegung der Funktions-
EF6D    11 FA1E                 LD      DE,KEYMEM       ;tasten dorthin, wo die Tastaturabfrage im
EF70    06 0A                   LD      B,10D           ;ROM deren Inhalt vorraussetzt (ROM-Zugriff)
EF72    7E              KEYTRN: LD      A,(HL)          ;KEYTAB = (INIT)
EF73    12                      LD      (DE),A
EF74    23                      INC     HL
EF75    13                      INC     DE
EF76    05                      DEC     B
EF77    C2 EF72                 JP      NZ,KEYTRN

EF7A    3A E665                 LD      A,(I25LIN)      ;Lese, ob Fußleiste ein oder aus (INIT)
EF7D    32 FA06                 LD      (LINE25),A      ;Wähle dementsprechend (ROM-Zugriff)

EF80    21 0047                 LD      HL,47H          ;Initialisiere VDP auf Textmodus
EF83    CD E84E                 CALL    BASROM

EF86    CD EBCE                 CALL    INI80Z          ;Initialisiere 80Zeichen-Karte

EF89    CD EC30                 CALL    INISER          ;Initialisiere serielle Schnittstelle

EF8C    21 EF95                 LD      HL,ININSG       ;Gib Kaltstart-Systemmeldung aus
EF8F    CD F06C                 CALL    MSGOUT

EF92    C3 EFF0                 JP      WSTART          ;übergib weitere Kontrolle der Warmstart-
                                                        ;Routine
EF95                    ININSG:
EF95    0C 1A 53 70             DB      0CH,1AH,'Spectravideo CP/M-80 Reversion '
EF99    65 63 74 72
EF9D    61 76 69 64
EFA1    65 6F 20 43
EFA5    50 2F 4D 2D
EFA9    38 30 20 52
EFAD    65 76 65 72
EFB1    73 69 6F 6E
EFB5    20
EFB6    32 2E 32 33             DB      '2.23',0DH,0AH,'For SV-605B',0DH,0AH
EFBA    0D 0A 46 6F
EFBE    72 20 53 56
EFC2    2D 36 30 35
EFC6    42 0D 0A
EFC9    43 6F 70 79             DB      'Copyright (C) by Digital Research',0DH,0AH,00H
EFCD    72 69 67 68
EFD1    74 20 28 43
EFD5    29 20 62 79
EFD9    20 44 69 67
EFDD    69 74 61 6C
EFE1    20 52 65 73
EFE5    65 61 72 63
EFE9    68 0D 0A 00

                        ;Abkürzungen:   VDP             Video-Display-Prozessor

                        ;Erklärungen:   (INIT)          Diese Daten können auf der Systemdiskette
                        ;                               eigenen Vorstellungen angepaßt werden

                        ;               (ROM-Zugriff)   Die hier abgelegten Daten werden von
                        ;                               einzelnen ROM-Routinen benützt

                                END                     ;Ende des Quelldatei
No  Fatal error(s)
```

Die 80 Zeichen-Karte ist für jene SVI-Besitzer unentbehrlich, die mit ihrem Computer auch Schreibarbeiten mittels eines Textverarbeitungsprogramms erledigen wollen. Auch andere Programme für den geschäftlichen Bereich verlangen eine Ausgabe von 80 Zeichen pro Zeile auf dem Bildschirm. Wie das funktioniert, wollen wir in dieser Folge zeigen.

Nach einer Zwangspause (aus Platzmangel mußte der Hardware-Artikel im Heft 2/85 entfallen) wenden wir uns diesmal der 80 Zeichen-Karte zu. Der angekündigte Bericht über den Floppy-Controller wird in einem der kommenden Hefte ausführlich nachgeholt.

Die 80 Zeichen-Karte wird vor allem in jenen Fällen Anwendung finden, wo es auf eine übersichtliche Darstellung auf dem Bildschirm ankommt und wo man eine Ausgabe auf einen Drucker mit einer Zeilenbreite von 80 Zeichen wünscht. In der Regel wird es sich dabei vor allem um geschäftliche Anwendungen handeln. Wegen der deutlicheren Lesbarkeit der Zeichen und der geringeren Ermüdung der Augen wird die 80 Zeichen-Karte aber auch dem eifrigen Programmierer wertvolle Dienste leisten.

Wegen der hohen Auflösung ist der Betrieb der 80 Zeichen-Karte nur in Verbindung mit einem monochromen Monitor (grün oder bernstein) möglich, die Darstellung auf einem Fernseher oder einem Farbmonitor ist unleserlich. Lediglich Farbmonitore der gehobenen Preisklasse (ab ca. 20.000,- S) könnten ein brauchbares Bild liefern.

Man wird also bei Anschaffung einer 80 Zeichen-Karte einen monochromen Monitor wählen. Diese Monitore sind teilweise auch mit Tonausgabe erhältlich. Die 80 Zeichen-Karte verfügt über einen eigenen Video-Ausgang, der direkt an den Monitor angeschlossen wird. Der TV- bzw. Videoausgang am Computer kann gleichzeitig an ein Fernsehgerät oder einen Farbmonitor angeschlossen bleiben.

Die 80 Zeichen-Karte kann nur in Zusammenhang mit einer Floppy-Station betrieben werden. Es gab offensichtlich im BASIC-ROM keinen Platz für die notwendigen Routinen zum Betrieb dieser Karte. Daher werden diese erst mit dem Disk-BASIC in den Computer geladen. Der Befehl "WIDTH 80" ist zwar im BASIC-ROM vorhanden, ein Aufruf ohne Disk-BASIC endet aber mit der Fehlermeldung "Illegal function call".

Die 80 Zeichen-Karte wird übrigens ganz einfach in einen der verfügbaren Slots des Expanders gesteckt und wie beschrieben mit einem Monitor verbunden. Man schaltet den Computer mit einer eingelegten Disk-BASIC-Diskette ein. Im BASIC meldet sich der Computer in der Regel auf dem 40 Zeichen-Schirm (z.B. dem Fernseher). Wenn man nicht beide Sichtgeräte gleichzeitig angeschlossen hat, muß man nun den Befehl "WIDTH 80" eingeben ("blind", also ohne Anzeige auf dem Schirm). Hat man dies getan, meldet sich der Computer auf dem Monitorschirm mit "Ok".

Man kann diese zugegeben mühsame Art der Inbetriebnahme vereinfachen, wenn man mit dem Befehl "IPL" ein Programm zum Selbststart bei der Initialisierung des Systems vorsieht. Man gibt die Diskette in Laufwerk 1 und gibt im direkten Modus folgendes ein:

```
IPL "RUN" + CHR$(34) + "1:INIT" + CHR$(34)
```

Dies bewirkt, daß beim Einschalten des Systems (beim "booten") das Programm INIT von dieser Diskette (im Laufwerk 1) geladen und auch gleich gestartet wird. CHR$(34) ist übrigens das Anführungszeichen, das bei diesem Befehl "verschlüsselt" eingegeben werden muß.

Nun benötigen wir also nur noch das Programm INIT auf der Diskette, damit die Angelegenheit auch funktioniert. Das Programm könnte zum Beispiel so aussehen:

```
10 FOR I% = 0 TO 15
20 READ A% : OUT &H50,I% : OUT &H51,A%
30 NEXT I%
40 WIDTH80
50 KEY1,CHR$(21)+"LOAD"+CHR$(34)+"1:"
60 DATA 108,80,89,8,33,8,24,29,2,8,96,8,0,
   0,0,0
```

Es ist zweckmäßig, bei dieser Gelegenheit gleich auch die Funktionstasten umzubenennen und mit jenen Befehlen zu belegen, die man beim Arbeiten mit Disketten häufiger benötigt. Wir haben hier aus Platzgründen nur eine Tastenbelegung geändert.

Die geheimnisvolle Zahlenfolge in der DATA-Zeile wird für das Laden der Register des Video-Controllers auf der 80 Zeichen-Karte verwendet. Darauf kommen wir später noch zurück. Und ganz besonders wichtig ist natürlich der Befehl "WIDTH 80" in Zeile 40, der auf 80 Zeichen-Darstellung umschaltet.

Bei Inbetriebnahme des CP/M-Betriebssystems ist die Sache einfacher. Das Betriebssystem "sieht" selbst nach, ob eine 80 Zeichenkarte vorhanden ist. Ist dies der Fall, meldet sich der Computer am 80 Zeichen-Schirm, andernfalls am TV-Gerät. Es gibt jedoch auch hier die Möglichkeit, von 80 auf 40 Zeichen-Betrieb umzuschalten.

Wir haben vorhin den Video-Controller erwähnt. Sehen wir uns den Baustein bei dieser Gelegenheit näher an. Es handelt sich um den 6845. Bevor wir auf die speziellen Eigenschaften dieses Chips eingehen, sollten wir wissen, welche Aufgaben ein Video-Controller zu erfüllen hat.

Er erzeugt die Synchronisationssignale für den Monitor, weiters die Adressen für den Bildwiederholspeicher. Auch die Position des Cursors ist zu ermitteln. Die Anzahl der Zeilen, Bildschirmweite und die Synchronisationsart werden ebenso wie beispielsweise die Cursordarstellung vom Mikroprozessor (dem Z80) programmiert und in die Register des Video-Controllers geschrieben.

Der 6845 verfügt über zwei voneinander unabhängige Zählerketten. Die eine zählt von einem programmierbaren Anfangswert bis zu einer Endadresse, damit keine Lücken bei der Adressierung des Bildwiederholspeichers entstehen. Die zweite Zählerkette bestimmt den Ablauf der Synchronisationssignale. Der Baustein kann einen Adreßbereich von 16 KByte ansprechen, ein Blättern im Bildwiederholspeicher wäre so möglich. Auf unserer 80 Zeichen-Karte befinden sich jedoch nur 2 KByte RAM, diese Möglichkeit entfällt somit.

Fortsetzung auf Seite 12

Mehr BASIC-Speicher für den SVI-328

Der SVI-328 besitzt 80K RAM. Bis jetzt standen jedoch nur knappe 30K fürs BASIC zur Verfügung. Erst durch ein kleines Maschincode-Programm konnten auch die zweiten 30K RAM nutzbar gemacht werden. Wir geben neben einem dokumentierten Listing auch eine genaue Bedienungsanleitung heraus.

Der SVI-328 hat 80KByte RAM. Allerdings werden 16KBytes davon als Video-RAM verwendet. Da in der ersten BANK auch noch 32K ROM untergebracht werden müssen, bleiben fürs RAM nur noch 32KByte Platz in der BANK 0. Im Normalfall läßt sich nur die erste BANK vom BASIC ansprechen (es sei denn, man hat eine Hardware-Speichererweiterung), daher waren die 'zweiten 32K der Grundversion bis jetzt nur für Maschincode-Programme erreichbar in BANK 2 untergebracht. Die SVI-Besitzer mußten frustiert feststellen, daß von den angepriesenen 80K RAM nur knappe 32 zur Verfügung standen. Durch ein kleines Maschincode-Programm kann man nun die Inhalte der beiden großen RAM-Blöcke vertauschen. Damit hat man die gesamten 64K RAM zur Verfügung. Wir wollen uns etwas näher mit diesen Programmen beschäftigen.

Wir wollen hier nicht näher auf die Feinheiten der Programmiertechniken eingehen, diese wurden ja schon im Heft 1/85 erklärt. Wichtig ist für uns, wie man die Programme bedient. Als konkretes Beispiel verwenden wir dazu jenes Programm, welches auf der BASIC-Diskette "serienmäßig" mitgeliefert wird. Für alle, die dieses Programm noch nicht mitgeliefert bekommen haben, steht unten das ausgedruckte Hex-Dump.

Nach dem Einspeichern und Starten des Programms installiert es sich automatisch. Ab nun kann man mit CMD SWITCH von einer BANK in die andere schalten. Wir wollen alle Eigenheiten des Programms klären, deshalb geben wir nach der Installation der MC-Entwicklung ein kleines BASIC-Programm ein (bitte noch nicht CMD SWITCH eingeben!).

```
10 REM BANK 0
20 PRINT "DIES IST EIN TEST/BANK 1
30 CMD SWITCH
40 GOTO20
```

Nun tippen wir LIST. Das Programm ist, wie zu erwarten war, im Speicher. Aber jetzt kommt CMD SWITCH an die Reihe. Wir geben ein:

```
CMD SWITCH und nachher LIST
```

Es erscheint jedoch nur ein Ok. Unsere logische Folgerung: Wir haben die Banken getauscht. Nun befinden wir uns in der zweiten Hälfte des RAMS. Wir geben jetzt CMD SWITCH ein, um wieder zurückzukommen, doch oh Schreck, wir springen immer wieder in die BANK 2.

Wir haben ja in der BANK 0 ein GOTO stehen, welches den Computer immer wieder in die zweite Hälfte des RAMs springen läßt. Da der Inhalt sämtlicher Zeiger und Variablen für die einzelnen Bänke unverändert bleibt, arbeitet der Computer nach dem SWITCH automatisch die Zeile ab, die unmittelbar nach der Absprungstelle in die andere BANK steht. Dieses Prinzip funktioniert fast genauso wie das des Unterprogramms. Wir werden später noch darauf zurückgreifen.

Nun gibt es zwei Möglichkeiten, um aus dieser Misere zu entkommen. Methode 1, wir schalten den Computer ab und fangen alles von vorne an. Diese Methode ist jedoch nicht sonderlich elegant. Also müssen wir uns etwas Anderes suchen.

Zu unserem Glück hat der Programmierer dieser Befehlserweiterung daran gedacht, in der Anweisung noch eine Option zu implementieren, nämlich STOP. Wir geben CMD SWITCH STOP ein. Der Computer "switcht" wieder in die Bank 0, bricht aber seinen Programmablauf ab.

Fassen wir zusammen:

Mit CMD SWITCH schaltet der Computer auf die andere BANK um. Sämtliche Zeiger und Variablen werden in der jeweiligen BANK mitgerettet. Daher setzt der Computer den Programmablauf bei einem neuerlichen CMD SWITCH dort fort, wo er unmittelbar vor dem Absprung aufgehört hatte.

Bei CMD SWITCH STOP unterbricht der Computer den Programmablauf nach dem Wechsel der Banken.

Natürlich muß man bestimmte Variablen übergeben können. In einem Programm werden immer einige "Standard"-Daten gebraucht. Hier kann man das Video-RAM verwenden. Es sind ja immer einige Teile des Bildschirmspeichers unbenutzt. Wenn man relativ wenige Daten überspielen muß, kann man die 32 Sprites, die ja auch im Video-RAM liegen, dazu verwenden. Man definiert einfach die Sprites mit den Variablen, die man braucht. Nach dem Wechsel liest man die Formen dann wieder aus. Man kann auch numerische Variable "überliefern". Hier wandelt man mittels MK?$ die numerische Variable in einen String um. Beim Auslesen wird CV? zum Wiederherstellen der Zahlen verwendet. Für das Fragezeichen kann man die Buchstaben S, D oder I einsetzen, je nach Variablentyp.

Wir können dies in unserem Programm probieren. Dazu ergänzen wir es wie folgt (Sollten Sie sich noch in BANK 2 befinden, geben Sie einfach CMD SWITCH STOP ein):

```
15 INPUT A
16 SPRITE$(0)=MKD$(A)
```

Jetzt tippen wir RUN. Der Computer frägt nach der Variablen, danach erscheint ein Ok, wir sind in der BANK 2. Wir holen uns nun mit ?CVD(SPRITE$(0)) den Wert wieder aus dem Video-RAM. Wir können natürlich statt dem Fragezeichen auch irgendeine Variablenzuweisung davor setzen. Man kann also die Variablennamen der übergebenen Variablen beim "switchen" ändern. Dies führt uns zu einem weiteren Anwendungsgebiet:

Da sich das "BANK-SWITCHING" ungefähr wie ein Unterprogrammaufruf verhält, kann man eine Subroutine in die zweite BANK laden. Einen Vorteil gibt es hier natürlich auch wieder. Die Variablen gelten im normalen BASIC das ganze Programm hindurch, sowohl in Unter- als auch in Hauptprogrammen. Wenn in einem Unterprogramm die Variable D verändert

wird, dann übernimmt sie das Hauptprogramm verändert.

Durch das "BANK-SWITCHING" kann man die Variablen des Unterprogramms unabhängig von denen im Hauptprogramm machen. Jede der Banken hat ihre eigenen Variablen, es werden nur die Variablen übergeben, die gewünscht sind. Man muß nicht mehr darauf aufpassen, daß eine Rechenvariable im Unterprogramm den weiteren Arbeitsverlauf des Hauptprogramms stört.

Dies ist eigentlich ein wichtigerer Grund, das Bankenwechseln zu befürworten, denn eine "bloße" Speichererweiterung wird für den Großteil der Computerprogrammierer sowieso nicht gebraucht. Es gibt sehr wenige Programme, die wirklich über 29K Speicher brauchen.

Kommen wir noch kurz zu den Variablen zurück: Sollte man mehr als 32 Variablen übergeben wollen, kann man im Video-RAM mittels VPOKE Daten abspeichern. Knappe 16K RAM dürften ja als Datenspeicher genügen!

Es gibt übrigens sowohl in dem unten abgebildeten Hex-Dump als auch im Programm aus Heft 1/85 einen weiteren Befehl, CMD COPY. Er kopiert den Inhalt der BANK 0 in den zweiten RAM-Block. So kann man schnell eine Version seines Programms sichern, ohne auf einen Massenspeicher (Disk oder Kassette) zugreifen zu müssen.

Wir probieren dies gleich an Ort und Stelle aus. Sobald wir wieder in der BANK 0 angelangt sind, tippen wir CMD COPY ein. Nun löschen wir unser Programm mittels NEW und "switchen" mittels CMD SWITCH STOP in die zweite Bank. Hier erkennen wir unser altes Programm. Bei einem neuerlichen CMD SWITCH ist das Programm wieder verschwunden, was aber klar ist, da wir es mit NEW vorhin gelöscht hatten.

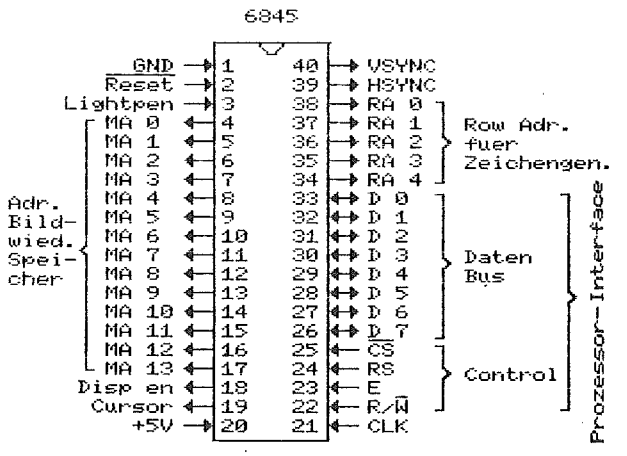
```
D000  FE D6 28 07 FE C9 28 03
D008  C3 00 00 C1 FE D6 F5 D7
D010  F1 28 01 7E FE 90 37 20
D018  02 D7 AF E5 ED 73 76 FE
D020  F5 F3 3E 0F D3 88 DB 90
D028  0E FD CB 57 20 02 0E F7
D030  A1 4F DB 90 08 21 EB 33
D038  18 0D 1A CB BA 12 CB FA
D040  13 10 F7 08 D3 8C 08 5E
D048  23 56 23 46 23 79 D3 8C
D050  7A B3 20 E6 F1 38 05 3E
D058  03 32 2B 7E 01 00 80 21
D060  FF 7F 55 5D FE D6 20 05
D068  EB ED B8 18 07 1A ED A8
D070  02 EA 6D 80 ED 7B 76 FE
D078  08 D3 8C CD 50 00 E1 FB
D080  C9 00 01 82 00 11 00 80
D088  21 00 D0 ED B0 ED 53 4A
D090  F5 CD 71 01 2A 26 00 11
D098  23 01 19 E5 01 03 00 11
D0A0  08 80 ED B0 E1 36 C3 23
D0A8  70 23 36 80 21 BB D0 CD
D0B0  8D 00 21 9C 00 E3 3E D6
D0B8  C3 0C 80 0D 0A 0A 53 70
D0C0  65 69 63 68 65 72 20 66
D0C8  65 72 74 69 67 20 21 20
D0D0  20 20 20 20 20 20 20 0D
D0D8  0A 20 20 20 20 20 20 20
D0E0  20 20 20 20 20 20 20 20
D0E8  20 20 20 20 20 20 20 20
D0F0  20 20 20 20 20 20 20 20
D0F8  0D 0A 0A 00 00 00 00 00
```

Das Maschincode-Programm, das oben als Hex-Dump abgebildet ist, kann mit BSAVE "CMD",&HD000, &HD0FF,&HD082 auf Kassette abgespeichert werden. Für Disketten stellt man einfach ein "1:" vor "CMD", also "1:CMD". Die Speichererweiterung startet ab &HD082, vorher wird lediglich eine Tabelle abgelegt. Sollten Sie also per DEFUSR starten wollen, aufpassen!

Und nun viel Spaß beim BANK-SWITCHING!

SVI-Hardware
Fortsetzung von Seite 10

Die folgende Darstellung zeigt die PIN-Belegung des 6845.



Der 6845 belegt nur zwei I/O-Adressen. Beim SVI-Computer handelt es sich um die Adressen 50H und 51H. Zur Auswahl dient der Eingang RS (register select). Der direkte Zugriff auf das Video-RAM der 80 Zeichen-Karte wird auf der Adresse 58H mit FFH ermöglicht, mit 0 wird das Video-RAM wieder weggeschaltet (CRT Bank Control).

Der 6845 verfügt über 18 Register. Die Auswahl der Register erfolgt durch Schreiben der Registeradresse auf die Portadresse 50H (RS = 0). Danach können bei der Portadresse 51H (RS = 1) Daten in dieses Register geschrieben bzw. gelesen werden.

Welche Aufgaben die einzelnen Register zu erfüllen haben, zeigt die folgende Tabelle:

Register	Text	R/W
0	Horizontal gesamt	W
1	Horizontal angezeigt	W
2	HSYNC-Position	W
3	HSYNC-Breite	W
4	Vertikal gesamt	W
5	Vertikal-Abgleich	W
6	Vertikal angezeigt	W
7	VSYNC-Position	W
8	Interlace (Zeilensprung)	W
9	Anzahl der Bildzeilen pro Zeichen	W
10	Zeilenstartadresse des Cursors	W
	und Darstellungsart des Cursors	W
11	Zeilenendadresse des Cursors	W
12	Startadresse des Speichers H	W
13	Startadresse des Speichers L	W
14	Cursorposition H	R/W
15	Cursorposition L	R/W
16	Lichtgriffel-Adresse H	R
17	Lichtgriffel-Adresse L	R

An anderer Stelle finden Sie in diesem Heft ein Programm, mit dem man den Bildschirminhalt bei 80 Zeichen-Darstellung direkt zum Drucker schicken kann. Dies kann besonders dann wünschenswert sein, wenn man eine erstellte Bildschirmmaske samt Inhalt zum Drucker schicken möchte. Es entfällt dann die mühsame Generierung einer entsprechenden Druckerausgabe durch LPRINT-Befehle.

Tips und Tricks für den SVI-728

Besitzer des MSX-Computers SVI-728 können die meisten der im SVI-Journal für SVI-328 publizierten Programme ohne Änderung übernehmen. Gelegentlich gilt es jedoch Abweichungen des MSX-BASIC zu beachten. Wir bringen daher wieder Tips zu diesem Thema.

Im letzten Heft ist uns im Programm zur Ausgabe von Texten im Graphikmodus in der Eile ein Schönheitsfehler passiert. Das Programm funktioniert zwar, statt PSET sollte man jedoch PRESET verwenden. PSET setzt einen Punkt in der Zeichenfarbe, während PRESET nur einen unsichtbaren Cursor an die entsprechende Stelle des Bildschirms setzt, da in der Hintergrundfarbe gezeichnet wird. Ersetzen Sie also bitte in dem Programm alle PSETs durch PRESET.

Der SVI-728 verfügt zwar über insgesamt rund 256 Zeichen (Buchstaben und Ziffern, diverse Graphikzeichen und Symbole, bzw. auch Buchstaben aus anderen Sprachen), die inverse Zeichendarstellung wird man jedoch vergeblich suchen. Das folgende Programm ermöglicht Ihnen auch die Darstellung inverser Zeichen. Allerdings muß man dabei auf einen Teil der Graphikzeichen und Sonderbuchstaben verzichten. Der geänderte Zeichensatz bleibt bis zum Abschalten des Computers erhalten.

Und hier nun das Programm:

```
10 FOR I% = 2304 TO 3055
20 A% = VPEEK(I%)
30 A% = A% XOR 255
40 VPOKE I%+768,A%
50 NEXT
```

Wenn Sie bei der Umwandlung der Zeichen zusehen wollen, fügen sie ganz einfach folgende Zeile in Ihr Programm ein (ehe Sie das Programm das erste Mal laufen lassen):

```
5 FOR I = 32 TO 255 : ? CHR$(I); : NEXT
```

Die Zeile 5 zeigt Ihnen den größten Teil des verfügbaren Zeichenvorrats zuerst in der ungeänderten Version an. Danach folgt die Umwandlung.

Die %-Zeichen bedeuten, daß wir mit ganzen Zahlen rechnen, dadurch können wir den Programmlauf beschleunigen.

Innerhalb der FOR-NEXT-Schleife des Programms wird nun Byte für Byte der Zeichendarstellung aus dem Video-RAM gelesen. In Zeile 30 wird jeder eingelesene Wert mit der Zahl 255 XOR-verknüpft. Dies bedeutet nichts anderes, als daß jedes Bit genau in sein Gegenteil verkehrt wird. Ist es gesetzt, so wird es gelöscht, ist es hingegen null, dann wird es auf eins gesetzt. Anschließend schreiben wir den Wert wieder in den Zeichengenerator hinein, allerdings 768 Byte versetzt. Der Wert, der dort eingetragen wird, ist der Kehrwert des ursprünglichen Werts, also unser inverses Zeichenmuster.

Jetzt muß man nur noch wissen, wohin wir die neuen Werte geschrieben haben. Da jedes dargestellte Zeichen aus acht Byte besteht, müssen wir die Zahl 768 durch acht dividieren, um zu wissen, um wieviele Zeichen die inverse Darstellung verschoben ist. Es ergibt sich also 96, d.h. die inverse Darstellung des "A" befindet sich um 96 Zeichen verschoben im Zeichengenerator.

Um ein inverses "A" darzustellen, können wir jetzt schreiben:

```
PRINT CHR$(ASC("A")+96)
```

Da wir wissen, daß der ASCII-Code von "A" 65 ist, könnten wir auch gleich schreiben:

```
PRINT CHR$(161)
```

Um einen String in inverser Schrift darzustellen, könnte man folgendes Programm verwenden:

```
10 A$ = "WORT"
20 FOR I = 1 TO LEN(A$)
30 MID$(A$,I,1)=CHR$(ASC(MID$(A$,I,1))+96)
40 NEXT
50 PRINT A$
```

Wenn Sie diese Programm laufen lassen, erhalten Sie sofort "WORT" in inverser Schrift.

Beim SVI-328 erzielt man diese Darstellung wie folgt:

```
PRINT CHR$(27)"p"A$chr$(27)"q".
```

Z 80 - Programmieren in Assembler
Fortsetzung von Seite 6

(=nullten) Bit begonnen wird, aber am Bildschirm die Ziffern in umgekehrter Reihenfolge aufscheinen sollen.

Im Register C steht das Bit, welches vom Register B getestet werden soll. In C ist immer nur ein Bit gesetzt, sodaß immer sieben Bits von B ausgeblendet werden, und das zu testende Bit erhalten bleibt. War das Bit Null, so wird das ZERO-Flag gesetzt und eine Null in den Buffer geschrieben. Hat das Bit den Wert Eins gehabt, so ist ZERO-Flag nicht gesetzt. Es wird eine Eins in den Buffer geschrieben. Danach wird C mit sich selber addiert, was zur Folge hat, daß sich das gesetzte Bit in C um eine Stelle nach rechts schiebt. Ist jedoch schon das 7. Bit erreicht worden, wird beim Addieren eines Registers mit sich selber das gesetzte Bit ins CARRY-Flag geschoben. So erkennt das Programm auch, ob die Schleife schon acht mal durchlaufen wurde. Wenn nicht, so ist das CARRY-Flag nicht gesetzt, und die Schleife wird noch einmal durchlaufen.

In dieser Schleife wird das Registerpaar als Zeiger auf den Buffer verwendet. Immer nachdem eine Eins oder eine Null in den Buffer geschrieben wurde, wird der Pointer auf ihn, das Register DE, um Eins erhöht. Im letzten Programmteil wird dann der Inhalt des Buffers von hinten ausgegeben. Wenn die Schleife begonnen wird, hat DE durch die vorherige Schleife bereits den Wert des Endes des Buffers+1. Da aber DE um Eins zu hoch ist, muß es zuvor noch dekrementiert werden.

Fortsetzung folgt

Nachdem ich in der letzten Folge die FOR-Schleife und den GOTO-Befehl erklärt habe, will ich in diesem Teil die übrigen Verzweigungsbefehle von PASCAL vorstellen.

Im Gegensatz zu BASIC gibt es in PASCAL noch andere Befehle, welche die Programmierung einer Schleife ermöglichen. Eine Schleife ist allgemein ein Programmteil, der je nachdem, ob eine Bedingung erfüllt ist oder nicht, durchlaufen wird. Die FOR-NEXT-Schleife ist ein Sonderfall, bei der die Bedingung folgendermaßen lautet:

```
Scheifenvariable = Schleifenvariable + 1
Solange Schleifenvariable <= Endwert
   führe Schleifenkörper aus
```

Manchmal will man aber die Schleifenvariable nicht automatisch um Eins erhöhen oder erniedrigen, sondern sie zum Beispiel mit einer Formel im Schleifenkörper berechnen. Um diese Art von Schleifen zu programmieren, stellt PASCAL zwei Befehle zur Verfügung:

```
REPEAT
  .
  .   Schleifenkörper
  .
UNTIL Bedingung = wahr;
```

und

```
WHILE Bedingung = wahr DO
  BEGIN
    .
    .   Schleifenkörper
    .
  END;
```

Der größte Unterschied zwischen diesen Befehlen liegt darin, daß eine REPEAT-UNTIL Schleife mindestens einmal durchlaufen werden muß, eine WHILE Schleife nicht unbedingt. Bevor ich auf diese Schleifen näher eingehe, muß ich noch erklären, wie man in PASCAL eine Bedingung formuliert. Es gibt in PASCAL mehrere sogenannte "logische Operatoren". Diese Operatoren führen einen Vergleich aus, der zwei verschiedene Endergebnisse haben kann : wahr oder falsch, bzw. TRUE oder FALSE in PASCAL. Logische Operatoren können folgende Zeichen sein:

= :Das Gleichheitszeichen vergleicht zwei Ausdrücke. A = B liefert TRUE, wenn A gleich B ist, und FALSE, wenn A ungleich B ist.

<> :Das Ungleichheitszeichen ist das Gegenteil des Zeichens "=". A <> B ergibt TRUE wenn A ungleich B ist. Wenn A gleich B ist, ergibt diese Operation FALSE.

> :Der Vergleich A > B liefert TRUE wenn A größer als B ist, und FALSE wenn A kleiner oder gleich B ist.

>= :Die Operation A >= B ist ähnlich. TRUE wird erzeugt, wenn A größer oder gleich ist. Wenn A kleiner B ist, ergibt dieser Vergleich FALSE.

< :Dieses Zeichen ist das Kleinerzeichen. Der logische Vergleich A < B ergibt TRUE, wenn A kleiner B ist, und FALSE, wenn A größer oder gleich B ist.

<= :A <= B liefert TRUE, wenn A kleiner oder gleich B ist, und FALSE, wenn A größer B ist.

IN :Dieser logische Operator ist etwas komplexer. Er überprüft, ob ein Wert in einer Menge enthalten ist. A IN (. 12..20 .) gibt TRUE, wenn A größer gleich 12 und kleiner gleich 20 ist, und FALSE, wenn A nicht im Interval von 12 bis 20 liegt. Dieser Operator wird später noch genauer behandelt.

Um diese Vergleiche einfacher zu erfassen, wurde in PASCAL ein eigener Variablentyp geschaffen, der nur die zwei Zustände TRUE und FALSE annehmen kann, nämlich den Typ BOOLEAN. Mit dem Befehl:

```
VAR  variable:BOOLEAN;
```

wird eine boolsche Variable mit dem Namen "variable" definiert. Ihr können auf zwei Arten Werte zugewiesen werden. Man kann der Variable direkt einen Wert zuweisen:

```
variable:=TRUE;
variable:=FALSE;
```

Die Variable kann aber auch direkt das Ergebnis einer logischen Operation übernehmen:

```
variable:= A <= B;
```

Um boolsche Variablen miteinander zu verknüpfen, gibt es noch einige Spezialbefehle:

NOT variable "dreht" die Variable um, d.h. aus TRUE wird FALSE und umgekehrt.

A OR B liefert folgende Ergebnisse:

	Wert von A: TRUE	FALSE
Wert von B: TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

A AND B liefert folgendes:

	Wert von A: TRUE	FALSE
Wert von B: TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

Zusätzlich gibt es in TURBO-PASCAL noch die Verknüpfung XOR (nicht Standard-PASCAL):
A XOR B liefert folgendes:

	Wert von A: TRUE	FALSE
Wert von B: TRUE	FALSE	TRUE
FALSE	TRUE	FALSE

Für diese Operatoren gibt es folgende Vorrangregeln: NOT wird vor AND behandelt, das wiederum vor OR ausgeführt wird, und zuletzt kommt XOR.

Nach diesem Exkurs wieder zurück zu den Schleifen. Zuerst will ich die REPEAT - UNTIL Schleife behandeln. Dazu möchte ich ein kleines Programm schreiben, das den Sinus von 0 - 2*PI im Abstand von PI/5 berechnet:

```
PROGRAM sinus;

VAR wert:REAL;
    step:REAL;
 BEGIN
   CLRSCR;
   WRITELN('SINUS-WERTE 0-2*PI');
   WRITELN('');
   WRITELN('');
   WRITELN('! Wert      ! Sinus     ');
   WRITELN('-----------------------');
   step := PI/5;
   wert := 0;

     REPEAT
WRITELN('! ',wert:1:6,' ! ',sin(wert):1:6);
       wert := wert+step;
     UNTIL wert >= 2*PI

 END.
```

Der Aufbau des Programmes ist einfach. Zuerst werden zwei Variable, "wert" und "step" als REAL definiert. Die Variable PI ist in TURBO-PASCAL vordefiniert, muß also nicht mehr im Deklarationsteil eingeführt werden. Das Programm druckt zuerst einen Tabellenkopf auf dem Bidschirm aus, dann werden die Variablen "wert" und "step" auf ihre Anfangswerte gesetzt. Als nächstes wird eine REPEAT - UNTIL Schleife eingeleitet. Im Schleifenkörper lernen wir zwei neue Befehle kennen: Die Sinusberechnung und die formatierte Ausgabe von Zahlen. In TURBO-PASCAL sind folgende Winkelfunktionen vorhanden:

SIN (variable):	Sinus
COS (variable):	Cosinus
TAN (variable):	Tangens
ARCTAN (variable):	Arcustangens

Dabei ist noch zu beachten, daß alle Winkelfunktionen im Bogenmaß berechnet werden.

Betrachten wir den WRITELN-Befehl der Scheife nocheinmal:

```
WRITELN('! ',wert:1:6,' ! ',sin(wert):1:6);
```

Nachdem eine Variable ausgedruckt wurde steht noch ":1:6" im Listing. Dieser Befehl bewirkt daß eine Zahl formatiert ausgedruckt wird und das Format hat:

```
WRITE(a:vorkommastellen:nachkommastellen);
```

In unserem Beispiel wird festgelegt, daß die Zahlen mit einer Stelle vor dem Komma und sechs Stellen nach dem Komma ausgedruckt wird. Dadurch kann man eine exponentielle Form der Darstellung umgehen und eine schöne Tabelle ausdrucken.

Mit dem UNTIL - Befehl wird die Schleifenbedingung festgelegt. Dabei ist "UNTIL wert >= 2*PI" eigentlich "UNTIL (wert >= 2*PI) = TRUE". Bis "wert" größer oder gleich 2*PI ist, wird die Schleife wiederholt.

Ähnlich funktioniert die WHILE-Schleife. Der Unterschied zur REPEAT-UNTIL SChleife besteht darin, daß die Bedingung am Anfang der Schleife steht. Dadurch muß die WHILE-Schleife nicht unbedingt durchlaufen werden. Der Syntax der WHILE-Schleife ist ähnlich der FOR-Schleife. Neben der schon oben erwähnten Normalform gibt es noch eine Kurzform, wenn nur eine Anweisung wiederholt werden soll:

```
WHILE Bedingung DO statment;
```

Um die WHILE-Schleife besser zu demonstrieren, will ich ein kleines Programm zum Zahlenraten schreiben: Man muß eine Zahl zwischen 0 und 1000 erraten, das Programm sagt nur, ob die zu erratende Zahl größer oder kleiner als die eingegebene Zahl ist.

```
PROGRAM zahlenraten;

VAR eingabe,zahl:INTEGER;
    falsch,groesser,kleiner:BOOLEAN;

BEGIN
  CLRSCR;
  zahl:=RANDOM(1000);
  falsch:=TRUE;
    WHILE falsch DO
      BEGIN
        WRITELN('Bitte eine Zahl raten !');
        READLN(eingabe);
        falsch:=NOT(zahl=eingabe);
        groesser:=(zahl eingabe);
        kleiner:=(zahl eingabe);
        IF groesser THEN
          WRITELN('Die Zahl ist groesser');
        IF kleiner THEN
          WRITELN('Die Zahl ist kleiner');
        WRITELN('');
      END;
  WRITELN('Sie haben die Zahl erraten !');
END.
```

Nun zum Aufbau des Programms. Zuerst werden zwei INTEGER-Variablen, "zahl" und "eingabe" definiert. Dann werden 3 BOOLEAN-Variablen mit den Namen "falsch", "groesser" und "kleiner" vereinbart.

Zu Beginn des Hauptblocks wird die Zufallszahl mit dem Befehl "zahl:=RANDOM(1000);" erzeugt. Der Befehl "RANDOM(grenze)" ergibt eine ganzzahlige Zufallszahl zwischen 0 und "grenze". Dann wird die Variable "falsch" auf TRUE gesetzt.

Die While-Schleife wird an die Bedingung "falsch = TRUE" gebunden. Das Statement:

```
WHILE falsch DO
```

bedeutet eigentlich:

```
WHILE falsch = TRUE DO
```

Im Schleifenkörper wird zuerst nach einer Eingabe gebeten. Dann werden die boolschen Variablen "falsch", "groesser", und "kleiner" gesetzt. "falsch" wird TRUE, wenn "zahl" ungleich "eingabe" ist, "groesser" wird TRUE, wenn "zahl" größer als "eingabe" ist, und "kleiner" wird TRUE, wenn "zahl" kleiner als "eingabe" ist. Danach lernen wir einen neuen Befehl kennen: Den IF-THEN-ELSE Befehl. Die Syntax lautet folgendermaßen:

```
IF bedingung=TRUE THEN
  BEGIN
    .
  END;
                ELSE
  BEGIN
    .
  END;
```

oder, wie in unserem Beispiel:

```
IF bedingung=TRUE THEN
  befehl;
```

Das "IF groesser THEN" entspricht wieder "IF groesser = TRUE THEN". Doch mehr über den IF-THEN-ELSE Befehl in der nächsten Folge.

In unserem Beispiel kann man sehr schön sehen, wie man durch BOOLEAN-Variablen ein Programm gut lesbar gestalten kann. "WHILE falsch DO" ist viel leichter verständlich als "WHILE zahl eingabe". Genauso ist es wichtig, die Variablennamen sinnvoll zu wählen, also nicht "a" oder "b" sondern eben "kleiner" oder "falsch". Dadurch kann ein Programm sehr leicht struktuiert werden.

RAFAEL RAZIM

Da die eindimensionalen Listen eine tragende Rolle auf dem Gebiet der Datenstrukturen spielen, widmen wir auch die zweite Folge dieser wichtigen Gruppierung. Danach sehen wir uns eine weitere Struktur an, den Stapel.

Eindimensionale Listen (arrays) können fast immer dazu verwendet werden, andere, komplexere Datenstrukturen zu simulieren. Grund genug, um auch in der zweiten Folge dieser Serie bei den Listen zu verweilen. Wir versuchen vorerst, zwei in der EDV-Programmierung oft verwendete Algorithmen zu beleuchten, die auf der Mengenlehre basieren. In der Folge stellen wir eine neue Datenstruktur vor, nämlich den Stapel (stack).

Auch zur Simulation des Stapels werden wir Listen und die dazugehörigen Zeiger (pointers) benutzen.

Vereinigung und Durchschnitt:

Obwohl es auf den ersten Blick einigen Lesern abwegig erscheinen mag, daß hier Beispiele aus der Mengenlehre herausgegriffen werden (die Mengenlehre wird eigentlich in den Grundschulen gelehrt), gestattet gerade die Mengenlehre, abstrakte Zahlenbegriffe nach logischen Prinzipien durch Mengenbegriffe zu ersetzen.

Unter Vereinigung zweier Mengen M1 und M2 versteht man alle diejenigen Elemente, die entweder in M1 oder in M2 enthalten sind. Dazu ein Beispiel: Nehmen wir an, wir wollen die Namen aller Mitglieder der SVI-Clubs Wien und Linz erfassen, wobei es sein kann, daß eine Person Mitglied bei beiden Clubs ist. Die Liste all dieser Leute würde die Vereinigung darstellen, wobei jede Person nur einmal angeschrieben ist, auch wenn sie in beiden Clubs vorkommt. Der implementierte Algorithmus für Vereinigung handelt nach einem ähnlichem Prinzip.

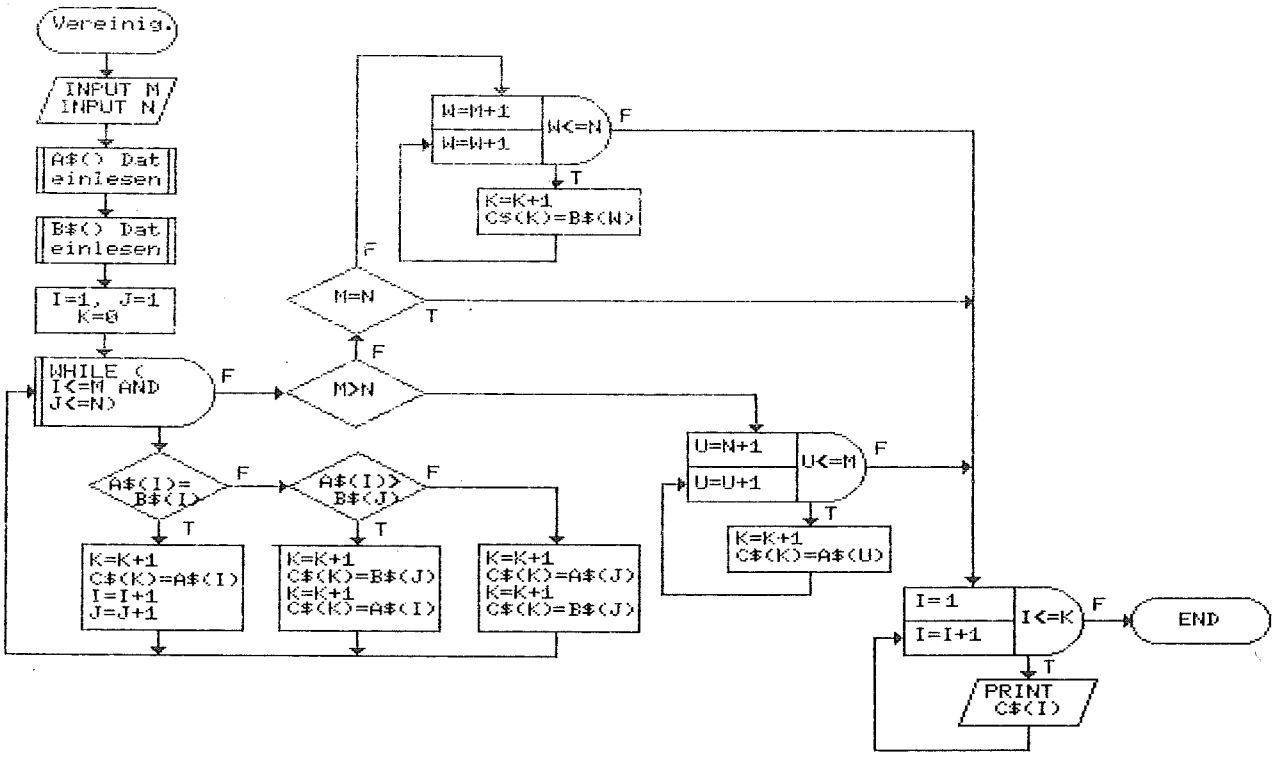
Der Durchschnitt zweier Mengen M1 und M2 umfaßt alle Elemente, die sowohl in M1 als auch im M2 enthalten sind. Um das Beispiel der SVI-Clubs nochmals zu verwenden: Wir würden nach denjenigen Personen suchen, die beiden Clubs angehören.

Zu den Algorithmen für Vereinigung und Durchschnitt selbst ist nicht viel zu sagen. Sie sind ziemlich einfach und bis zu einem gewissen Grad einander ähnlich. Es versteht sich von selbst, daß die Listen A$( ) und B$( ), welche die Daten für die Mengen M1 und M2 beinhalten, in aufsteigender Form sortiert sein müssen.

Stapel (stacks)

Ein Stapel, auch Keller genannt, ist eine Folge gleichartiger Speicherelemente, von denen nur das erste aufgerufen werden kann. Ein Stapel wird am besten durch das englische Kürzel LIFO (Last In First Out) beschrieben. Es besagt, daß das zuletzt eingefügte Element als erstes herauszuführen ist.

Man fügt neue Daten immer an der Spitze des Stapels dazu und bekommt jedesmal nur das oberste Element frei. Das Stapelprinzip soll nun anschließend mit einem Programmierbeispiel belegt werden, und zwar mit einem aus der Kompiliertechnik, das "Klammerausdrücke abarbeiten". Hierbei wird aus Platz- und Zeitgründen nur die Konvertierung von INFIX zu POSTFIX eines arithmetischen Ausdrucks gezeigt.



Zu den Begriffen INFIX und POSTFIX:

In der Mathematik redet man von Termen, zum Beispiel A(B+C). Die "normale" mathematische Schreibweise nennt man hier INFIX. Nun sehen wir uns an, wie der Computer diesen Term unter Beachtung aller Prioritäten abarbeitet. Es würde so aussehen: ABC+*=. Diese Art nennt man POSTFIX.

Der interessierte Leser sei auf ein Programm mit dem Titel "Türme von Hanoi" verwiesen, erschienen im SVI-Journal Heft 6/84, welches sehr starken Gebrauch von Stapeln macht. Und nun zu unserem Beispiel mit dem arithmetischen Ausdruck.

In INFIX-Notation ist ein mathematischer Ausdruck in der Form "Y=(A/B)*(-D^E)/F" gegeben. Doch ein Kompiler oder ein Interpreter muß diesen Ausdruck zuerst konvertieren und dann die arithmetischen Operationen ausführen. Wir werden das erstere besprechen, nämlich die INFIX- zur POSTFIX-Konvertierung. Dabei sollen aus Platz- und Zeitgründen folgende Vereinbarungen gelten:

1) Jede Variable oder Konstante im arithmetischen Ausdruck darf nur aus einem Charakter bestehen, z.B Y=A+3*B/C-D.

2) Die Syntax des Ausdrucks wird nicht überprüft.

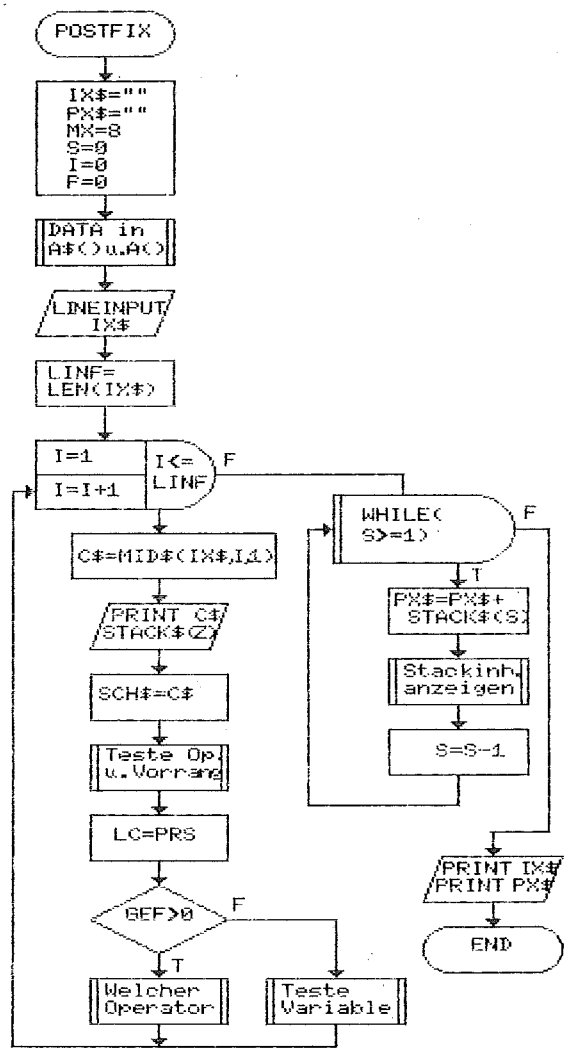
3) Der POSTFIX-Ausdruck sei am Beginn leer (Leerstring). Der INFIX-Ausdruck wird als String betrachtet, der aus einzelnen Substrings besteht, welche entweder Variablen oder Konstanten oder Operatoren sein dürfen. Der String wird nun in seine Bestandteile zerlegt und je nachdem, was dieser Bestandteil darstellt, wie folgt vorgegangen.

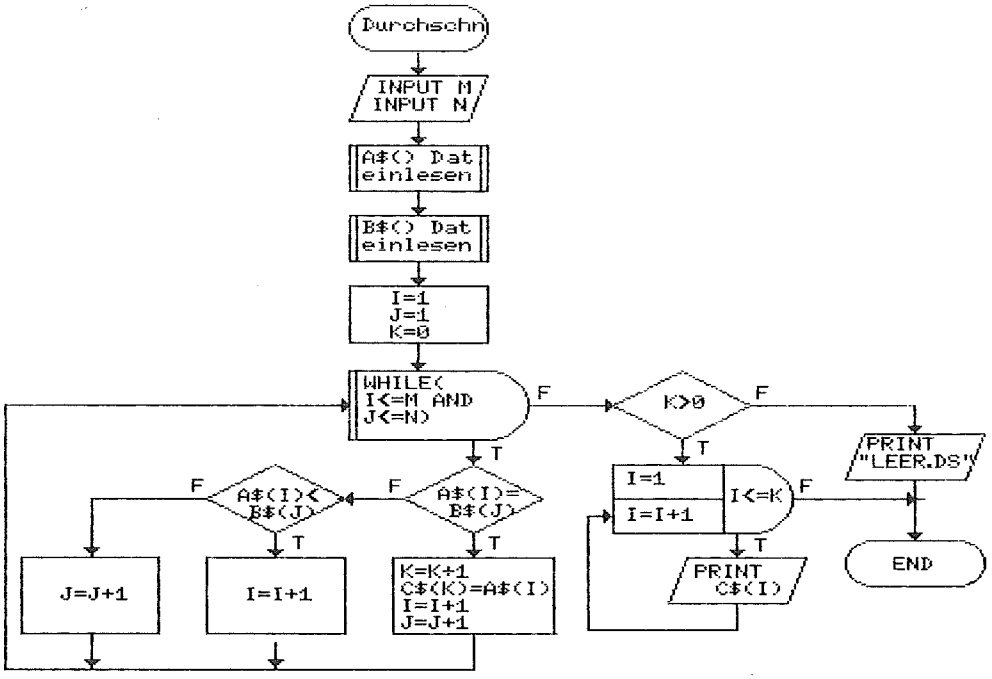
a) Token ist Variable oder Konstante

Es wird automatisch am rechten Ende des POSTFIX-Ausdrucks angehängt.

b) Token ist ein Operator (+,-,*,/,^,=)

Es wird an das offene Ende des Operatorstapels gestellt. Doch bevor der Operator abgelegt wird, findet ein Vergleich zwischen dem neuen Operator und dem bereits an der Spitze





des Stapels befindlichen Operator (wenn überhaupt) statt. Es wird deren Vorrang (Priorität) verglichen. Wenn der Vorrang des bereits am Stapel vorhandenen Operators größer oder gleich (>=) ist, als der des neuen Operators, wird der alte Operator aus dem Stapel geholt und am Ende des POSTFIX-Ausdrucks angehängt. Dieser Vorgang wird solange wiederholt, bis entweder der Stapel leer ist, oder ein Operator mit kleinerem Vorrang gefunden wurde.

c) Wenn keine Tokens mehr auf dem Ausdruckstring vorhanden sind, werden alle Elemente des Operator-Stapels, eines nach dem anderen, herausgeholt und an das Ende des POSTFIX-Ausdrucks gestellt.

Fortsetzung folgt

```
1000 '''''''''''''''''''''''''''''''''
1010 '   Vereinigung Algorithmus      '
1020 '                                '
1030 'written by C.Gaganas,Febr.85    '
1040 '                                '
1050 '''''''''''''''''''''''''''''''''
1060 CLS
1070 DEFINT A-Z
1080 PRINT CHR$(27)+"p"
1090 PRINT "VEREINIGUNG ALGORITHMUS"
1100 PRINT CHR$(27)+"q":PRINT
1110 INPUT "Anzahl der Elemente der List
e A$()";M
1120 INPUT "Anzahl der Elemente der List
e B$()";N
1130 L=M+N
1140 DIM A$(M),B$(N),C$(L)
1150 GOSUB 1310:GOSUB 1360
1160 I=1:J=1:K=0
1170 IF I>M OR J>N THEN GOTO 1200
1180 IF A$(I)=B$(J) THEN K=K+1:C$(K)=A$(
I):I=I+1:J=J+1
       ELSE IF A$(I)>B$(J) THEN K=K+1:C$
(K)=B$(J):J=J+1:K=K+1:C$(K)=A$(I):I=I+1
                           ELSE K=K+1:C$
(K)=A$(I):I=I+1:K=K+1:C$(K)=B$(J):J=J+1
1190 GOTO 1170
1200 IF M>N
       THEN GOSUB 1450
      ELSE IF M<N THEN GOSUB 1460
1210 '
1220 CLS
1230 PRINT "Vereinigte Liste"
1240 PRINT "................."
1250 PRINT:PRINT "(";K;" Elemente)":PRIN
T
1260 FOR I=1 TO K
1270   PRINT C$(I)
1280 NEXT I
1290 END
1300 ' Daten einlesen
1310 RESTORE 1420
1320 FOR I=1 TO M
1330   READ A$(I)
1340 NEXT I
1350 RETURN
1360 RESTORE 1440
1370 FOR I=1 TO N
1380   READ B$(I)
1390 NEXT I
1400 RETURN
1410 '
1420 DATA anton,berta,caesar,dora,erwin,
friedrich,gerhard,harald,ilse,joseph,kon
rad,leopold,mario,normann,otto,peter,qub
ert,robert,stanley,theodor,ulrich,vorman
n,werner,xmas,ypsilon,zero
1430 '
1440 DATA anton,berta,caesar,dalton,emil
,frommelt,gustav,harald,igor,johann,karl
,ludwig,mustaffa,nero,othmar,philip,quad
er,richard,stuntman,thomas,udo,vorarlber
g,werner,xander,ybbs,zenon
1450 FOR U=N+1 TO M:K=K+1:C$(K)=A$(U):NE
XT U:RETURN
1460 FOR W=M+1 TO N:K=K+1:C$(K)=B$(W):NE
XT W:RETURN
```

```
1000 '''''''''''''''''''''''''''''''''
1010 '   Durchschnitt Algorithmus     '
1020 '                                '
1030 'written by C.Gaganas,Febr.85    '
1040 '                                '
1050 '''''''''''''''''''''''''''''''''
1060 CLS
1070 DEFINT A-Z
1080 PRINT CHR$(27)+"p"
1090 PRINT "DURCHSCHNITT ALGORITHMUS"
1100 PRINT CHR$(27)+"q":PRINT
1110 INPUT "Anzahl der Elemente der List
e A$()";M
1120 INPUT "Anzahl der Elemente der List
e B$()";N
1130 L=M+N
1140 DIM A$(M),B$(N),C$(L)
1150 GOSUB 1310:GOSUB 1360
1160 I=1:J=1:K=0
1170 IF I>M OR J>N THEN GOTO 1200
1180 IF A$(I)=B$(J)
       THEN K=K+1:C$(K)=A$(I):
            I=I+1:J=J+1
     ELSE IF A$(I)<B$(J) THEN I=I+1
                         ELSE J=J+1
1190 GOTO 1170
1200 IF K = 0 THEN
    PRINT "Leerer Durchschnitt......":
   END
1210 '
1220 CLS
1230 PRINT "Durchschnitt"
1240 PRINT "............."
1250 PRINT:PRINT "(";K;" Elemente)":
    PRINT
1260 FOR I=1 TO K
1270   PRINT C$(I)
1280 NEXT I
1290 END
1300 ' Daten einlesen
1310 RESTORE 1420
1320 FOR I=1 TO M
1330   READ A$(I)
1340 NEXT I
1350 RETURN
1360 RESTORE 1440
1370 FOR I=1 TO N
1380   READ B$(I)
1390 NEXT I
1400 RETURN
1410 '
1420 DATA anton,berta,caesar,dora,
    erwin,friedrich,gerhard,harald,
   ilse,joseph,konrad,leopold,mario,
  normann,otto,peter,qubert,robert,
 stanley,theodor,ulrich,vormann,
werner,xmas,ypsilon,zero
1430 '
1440 DATA anton,berta,caesar,dalton,
    emil,frommelt,gustav,harald,igor,
   johann,karl,ludwig,mustaffa,nero,
  othmar,philip,quader,richard,
 stuntman,thomas,udo,vorarlberg,
werner,xander,ybbs,zenon
```

```
1000  '''''''''''''''''''''''''''''''''
1010  'INFIX zu POSTFIX Konvertierung  '
1020  'written by C.Gaganas,Maerz 85   '
1030  '                                '
1040  '''''''''''''''''''''''''''''''''
1050  ' no syntax check is performed.  '
1060  ' only one character variables/  '
1070  ' constants allowed.             '
1080  '''''''''''''''''''''''''''''''''
1090 CLS
1100 DEFINT A-Z
1110 IX$=""
1120 PX$=""
1130 MX=8 'Anzahl der Operatoren
1140 DIM A$(MX),A(MX),STACK$(50)
1150 S=0 'Pointer fuer STACK
1160 I=0 'Index fuer INFIX Ausdruck
1170 P=0 'Index fuer POSTFIX Ausdruck
1180 GOSUB 1910
```

```
1190 LINE INPUT "INFIX expression:";IX$:
     PRINT:PRINT
1200 LINF=LEN(IX$)
1210 FOR I=1 TO LINF
1220    C$=MID$(IX$,I,1):
        PRINT "Scanned character-> ";C$
1230    PRINT "Stack contents:";
1240    FOR Z=1 TO S
1250       PRINT CHR$(27)+"p";STACK$(Z);
           CHR$(27)+"q";
1260    NEXT Z
1270    PRINT
1280    SCH$=C$:GOSUB 1410:LC=-1:LC=PRS
1290    IF GEF > 0
           THEN GOSUB 1520
           ELSE GOSUB 1580
1300 NEXT I
1310 PRINT "All characters scanned:"
1320 IF S>=1
        THEN PX$=PX$+STACK$(S):
             GOSUB 1810:S=S-1:GOTO 1320
1330 PRINT:PRINT "INFIX  :";CHR$(27)+"p"
;IX$;CHR$(27)+"q"
1340 PRINT:PRINT "POSTFIX:";CHR$(27)+"p"
;PX$;CHR$(27)+"q"
1350 PRINT:PRINT
1360 END
1370 '''''''''''''''''''''''''''''''''''
1380 ' binary search for operators '
1390 '                             '
1400 '''''''''''''''''''''''''''''''''''
1410 PLTZ=0
1420 GEF=0:PRS=-1
1430 MITTE=0
1440 LO=1
1450 HI=MX
1460 IF HI < LO OR GEF=1
        THEN RETURN
1470 MITTE=(LO+HI)\2
1480 IF SCH$=A$(MITTE)
        THEN GEF=1:PLTZ=MITTE:
             PRS=A(MITTE):GOTO 1500
        ELSE IF SCH$ > A$(MITTE)
                THEN LO=MITTE+1
                ELSE HI=MITTE-1
1490 GOTO 1460
1500 RETURN
1510 ''''''''''''''''''''''''''''''''''''
1520 'check which operator                '
1530 '                                    '
1540 ''''''''''''''''''''''''''''''''''''
1550 IF C$=")" THEN GOSUB 1650:RETURN
 ELSE IF C$="(" OR C$="="
        THEN S=S+1:STACK$(S)=C$:RETURN
        ELSE GOSUB 1730
1560 RETURN
1570 ''''''''''''''''''''''''''''''''''''
1580 'check if variable                   '
1590 '                                    '
1600 ''''''''''''''''''''''''''''''''''''
1610 IF C$=" " THEN RETURN
1620 IF (C$>="A" AND C$<="Z") OR
        (C$>="a" AND C$<="z") OR
        (C$>="0" AND C$<="9")
        THEN PX$=PX$+C$
        ELSE PRINT "wrong variable";C$:
             END
1630 RETURN
1640 ''''''''''''''''''''''''''''''''''''
1650 'add to POSTFIX chars from stack     '
1660 '                                    '
1670 ''''''''''''''''''''''''''''''''''''
1680 IF S>=1 AND STACK$(S)<>"("
        THEN PX$=PX$+STACK$(S):S=S-1
        ELSE GOTO 1700
1690 GOTO 1680
1700 IF S>=1 THEN STACK$(S)=" ":S=S-1
1710 RETURN
```

Fortsetzung auf Seite 21

```
*****************************************
*                                       *
*        Text-Hardcopy für SVI-3x8      *
*       für 40 und 80 Zeichen-Schirm    *
*                                       *
*****************************************
```

Nachdem Ihnen im Heft 4/84 gezeigt wurde, wie Sie eine Hardcopy des Graphikschirmes anfertigen können, möchte ich Ihnen in diesem Artikel zeigen, wie Sie eine Hardcopy des Textbildschirmes bekommen. Sie können so zum Beispiel eine Karteikarte, die mit Maske auf dem Schirm gebildet wurde, ausdrucken, ohne ein kompliziertes Druckprogramm schreiben zu müssen. Die Verwendungsmöglichkeiten finden kein Ende.

Das Programm druckt den Inhalt des Schirmes aus, auf dem Sie gerade arbeiten. Das heißt, daß Sie sowohl den Inhalt der 80 Zeichenkarte ausdrucken können als auch den Inhalt des 40 Zeichen-Textschirmes aus dem Videoprozessor. Deshalb enthält das Programm mehrere wichtige Routinen zum Lesen der beiden verschiedenen Schirme. Weiters können Sie angeben, ab welcher Zeile das Programm den Ausdruck beginnen und bei welcher es dann aufhören soll. Allerdings gibt es hinsichtlich der invers dargestellten Zeichen eine Beschränkung, sie werden normal ausgedruckt (das ist damit zu begründen, daß der Drucker keine inversen Zeichen kennt).

Zum Programm selbst: Es ist komplett in Maschinensprache geschrieben und ist so angelegt, daß es auf dem SVI-328 und SVI-318 läuft. Außerdem können Sie aus dem Programm erkennen, wie die Umrechnung von Cursorkoordinaten in RAM-Adressen erfolgt, wie Sie den Code des Zeichens dort erhalten und wie Sie schließlich den Code in den ASCII-Code umwandeln können. Deshalb ist das Programm auch etwas länger, denn ich hätte ja ebensogut mit der Interpreterroutine ENTER (sie liest eine ganze Zeile ein) und der Druckroutine bei 697DH eine Zeile ausdrucken können.

Die Variablen des Programmes:

MODE enthält den SCREEN-Modus, d.h. wenn Sie SCREEN N eintippen, wird in MODE (=FE3AH) N abgespeichert.

WIDTH enthält die Bildschirmbreite. Ein Wert ungleich 80 in dieser Variable spricht immer den eingebauten Videoprozessor an. Enthält WIDTH die Zahl 80, so wird gegebenfalls die 80 Zeichen-Karte angesprochen.

XPOS und YPOS geben die Cursorpositon an. Sie haben nichts mit dem BASIC-Cursor zu tun und beeinflussen ihn nicht.

FRST_LN und LAST_LN geben an, ab welcher (=FRST_LN) und bis zu welcher Zeile (=LAST_LN) gedruckt werden soll. Sie können diese Werte durch POKE in C804H und C805H verändern.

Die ROM-Routinen:

CHANGE wandelt den Code eines Videozeichens in den entsprechenden ASCII-Code um. Der Akkumulator enthält beim Aufruf den Code des Videozeichens und gibt beim Rückspung den ASCII-Code zurück.

POS_40 errechnet anhand der Cursorkoordinaten im Register HL (H=Spalte und L=Zeile) die VRAM-Adresse (der Ursprung liegt links oben und wird mit X=1 und Y=1 angegeben).

READ_VDP liest den Inhalt der eben ermittelten VRAM-Adresse und übergibt ihn im Akku.

PRNT_CH gibt den Akkumulator an den Drucker aus. Die Routine wartet eventuell auf ON LINE und überprüft auf CTRL-STOP.

Die Programmroutinen:

START druckt ein CR-LF aus und setzt den Cursor in die Zeile, die durch FRST_LN angegeben wird.

LOOP_LN setzt den Cursor links an die Position Null.

LOOP_CL druckt eine ganze Zeile aus.

LINE_DO geht eine Zeile tiefer und hört bei der Zeile, die durch LAST_LN angegeben wird, auf.

GET_CHAR liest ein Zeichen in der Position (XPOS) und (YPOS) ein, wandelt es in ein ASCII-Zeichen um und gibt es dem Akku zurück.

ADRESSE entscheidet, ob das VRAM oder die 80 Zeichen-Karte adressiert werden soll.

POS_80 berechnet aus XPOS und YPOS die Adresse im RAM der 80 Zeichen-Karte.

READ_CH entscheidet, ob aus dem VRAM oder der 80 Zeichen-Karte gelesen werden soll.

READ_80 schaltet auf die 80-Zeichen-Karte um und liest das Zeichen bei der Adresse HL.

Die notwendigen Opcodes für das Programm erhalten Sie aus dem HEX-Dump. Das Programm beginnt bei C800H und endet bei &HC8AFH. Die Speicherzellen C804H und C805H haben eine besondere Bedeutung: C804H enthält FRST_LN und C805H LAST_LN (POKE &HC804,1:POKE &HC805,5:?USR(0) würde die Zeilen 1 bis 5 auf dem Drucker liefern).

```
C800  18 04 00 00 01 18 3A 04  ......:.
C808  C8 3D 32 02 C8 3A 3A FE  .=2..::.
C810  A7 C0 3E 0A CD 15 39 3E  ..>...9>
C818  0D CD 15 39 F3 3E 00 32  ...9.>.2
C820  03 C8 3A 02 C8 6F 26 00  ..:..o&.
C828  7C 32 03 C8 CD 64 C8 E6  ö2...d..
C830  7F CD 15 39 3A 03 C8 3C  ...9:..<
C838  32 03 C8 67 3A 43 F5 BC  2..g:C..
C840  20 E6 3E 0A CD 15 39 3E   .>...9>
C848  0D CD 15 39 3A 02 C8 3C  ...9:..<
C850  32 02 C8 6F 3A 05 C8 BD  2..o:...
C858  20 C8 3E 0A CD 15 39 3E   .>...9>
C860  0D C3 15 39 CD 6E C8 CD  ...9.n..
C868  95 C8 CD 3C 3C C9 3A 43  ...<<.:C
C870  F5 FE 50 CA 7F C8 2A 02  ..P...*.
C878  C8 24 2C CD 8A 3C C9 3A  .$,..<.:
C880  02 C8 4F 87 87 81 6F 26  ..O...o&
C888  00 29 29 29 29 3A 03 C8  .)))):..
C890  4F 06 F0 09 C9 3A 43 F5  O.....:C.
C898  FE 50 CA A1 C8 CD 34 37  .P....47
C8A0  C9 F3 AF 3D D3 58 4E 3C  ...=.XN<
C8A8  D3 58 79 C9 15 39 C9 00  .Xy..9..
```

```
100 REM          org  c800h
101 REM          jr   start
102 ' MODE enthaelt den SCREEN-
103 ' Modus, CHANGE wandelt den
104 ' Video-Code in ASCII-Code um
105 ' WIDTH enthält die Bildschirm-
106 ' breite, POS_40 berechnet aus
107 ' den Cursorkoordinaten die VRAM-
108 ' Adresse, READ_VDP liest
109 ' den Inhalt dieser Adresse in
110 ' den Akkumulator, PRNT_CH gibt
111 ' den Inhalt des Akkus an den
112 ' Drucker aus, XPOS und YPOS
113 ' geben die Position an, FRST_LN
114 ' und LAST_LN sagen dem Programm,
115 ' welche Zeilen gedruckt werden
116 ' sollen.
```

```
117 REM mode      equ  fe3ah
118 REM change    equ  3c3ch
119 REM width     equ  f543h
120 REM pos_40    equ  3c8ah
121 REM read_vdp  equ  3734h
122 REM prnt_ch   equ  3915h
123 REM ypos      db   00
124 REM xpos      db   00
125 REM frst_ln   db   1
126 REM last_ln   db   24
127 REM           ;
128 ' Hier erfolgt der Einsprung,
129 ' es wird ein Zeilenvorschub er-
130 ' zeugt und der Cursor (nicht der
131 ' BASIC-Cursor) nach links oben
132 ' gesetzt.
133 REM start     ld   a,(frst_ln)
134 REM           ld   (ypos),a
135 REM           ld   a,(mode)
136 REM           and  a
137 REM           ret  nz
138 REM           ld   a,10
139 REM           call prnt_ch
140 REM           ld   a,13
141 REM           call prnt_ch
142 REM           ;
143 REM           di
144 REM           ld   a,0
145 REM           ld   (xpos),a
146 ' LOOP_LN ist die äußere
147 ' Schleife, sie bewegt den Cursor
148 ' nach dem Ausdruck einer Zeile
149 ' um eine Zeile nach unten.
150 REM loop_ln   ld   a,(ypos)
151 REM           ld   l,a
152 REM           ld   h,0
153 ' LOOP_CL ist die innere Schleife,
154 ' sie bewegt den Cursor bis zum
155 ' Zeileende und druckt jedes
156 ' Zeichen aus.
157 REM loop_cl   ld   a,h
158 REM           ld   (xpos),a
159 REM           call get_char
160 REM           and  7fh
161 REM           call prnt_ch
162 REM           ld   a,(xpos)
163 REM           inc  a
164 REM           ld   (xpos),a
165 REM           ld   h,a
166 REM           ld   a,(width)
167 REM           cp   h
168 REM           jr   nz,loop_cl
169 ' Drucken von CR-LF und er-
170 ' höhen der Cursorzeile.
171 REM line_do   ld   a,10
172 REM           call prnt_ch
173 REM           ld   a,13
174 REM           call prnt_ch
175 REM           ld   a,(ypos)
176 REM           inc  a
177 REM           ld   (ypos),a
178 REM           ld   l,a
179 REM           ld   a,(last_ln)
180 REM           cp   l
181 REM           jr   nz,loop_ln
182 ' Beenden des Programmes,
183 ' drucken von CR-LF und Ende.
184 REM           ld   a,10
185 REM           call prnt_ch
186 REM           ld   a,13
187 REM           jp   prnt_ch
188 REM           ;
189 ' Einlesen eines Zeichens
190 ' sowohl aus dem VRAM als auch
191 ' aus dem RAM der 80-Zeichen-
192 ' karte. Beim Rücksprung ent-
193 ' hält der Akku den ASCII-Code des
194 ' Zeichens.
195 ' ADRESSE wandelt Bildschirm-
196 ' koordinaten in VRAM-Adressen um.
197 ' READ_CH liefert das Zeichen an
198 ' dieser Position.
199 ' CHANGE wandelt den Code in den
200 ' ASCII-Code um.
201 REM get_char call adresse
202 REM           call read_ch
203 REM           call change
204 REM           ret
205 REM           ;
206 REM adresse   ld   a,(width)
207 REM           cp   80
208 REM           jp   z,pos_80
209 REM           ld   hl,(ypos)
210 REM           inc  h
211 REM           inc  l
212 REM           call pos_40
213 REM           ret
214 REM           ;
215 ' Umrechnen von Koordinaten in
216 ' Adresse.
217 REM pos_80    ld   a,(ypos)
218 REM           ld   c,a
219 REM           add  a,a
220 REM           add  a,a
221 REM           add  a,c
222 REM           ld   l,a
223 REM           ld   h,0
224 REM           add  hl,hl
225 REM           add  hl,hl
226 REM           add  hl,hl
227 REM           add  hl,hl
228 REM           ld   a,(xpos)
229 REM           ld   c,a
230 REM           ld   b,f0h
231 REM           add  hl,bc
232 REM           ret
233 REM           ;
234 REM read_ch   ld   a,(width)
235 REM           cp   50h
236 REM           jp   z,read_80
237 REM           call read_vdp
238 REM           ret
239 REM           ;
240 REM read_80   di
241 REM           xor  a
242 REM           dec  a
243 REM           out  (58h),a
244 REM           ld   c,(hl)
245 REM           inc  a
246 REM           out  (58h),a
247 REM           ld   a,c
248 REM           ret
249 REM           ;
250 REM           end
```

Programm "INFIX zu POSTFIX Konvertierung"
Fortsetzung von Seite 19

```
1720 ''''''''''''''''''''''
1730 'check precedence     '
1740 '                     '
1750 ''''''''''''''''''''''
1760 LS=-1
1770 IF S>=1 THEN SCH$=STACK$(S):GOSUB 1
380:LS=PRS
1780 IF S>=1 AND LC<=LS
        THEN PX$=PX$+STACK$(S):
             S=S-1:GOTO 1770
        ELSE S=S+1:STACK$(S)=C$
1790 RETURN
1800 ''''''''''''''''''''''
1810 ' remaining operators '
1820 'show before appending'
1830 '                     '
1840 ''''''''''''''''''''''
1850 PRINT "Stack contents:";
1860 FOR Z=1 TO S
1870    PRINT CHR$(27)+"p";STACK$(Z);
            CHR$(27)+"q";
1880 NEXT Z
1890 PRINT
1900 RETURN
1910 'Data fuer Operatoren und
1920 'deren Vorrang(precedence)
1930 DATA (,1,),0,*,4,+,3,-,3,/,4,=,2
1940 DATA ^,5
1950 RESTORE 1930
1960 FOR K=1 TO MX
1970    READ A$(K):READ A(K)
1980 NEXT K
1990 RETURN
```

```
10 REM ********************************
20 REM *                              *
30 REM * TIEFKUEHLTRUHENVERWALTUNG    *
40 REM *    (C) BY ANDREAS STOCKER    *
50 REM *      MUHRHOFERWEG 1-5/8/25   *
60 REM *           A-1110 WIEN        *
70 REM *       TEL.:0222/76 18 225    *
80 REM *                              *
90 REM ********************************
100 CLS:SCREEN0,0
105 REM R = RIGHT GRPH 'Y'
110 PRINT"         R R":PRINT
120 PRINT"R   R R      R R      R R"
130 PRINT"R  R  R      R R      R R"
140 PRINT"R R   R      R R      R R"
150 PRINT"RR    R      R RRRRRRR R"
160 PRINT"R R   R      R R      R R"
170 PRINT"R  R  R      R R      R R"
180 PRINT"R   R  RRRRR  R      R RRRRR"
190 PRINT:PRINT
200 PRINT" KUEHLTRUHENVERWALTUNG"
210 PRINT:PRINT:PRINT"  (C) 1984 11 10   BY"
220 PRINT:PRINT"          ANDREAS STOCKER"
230 PRINT:PRINT"          MUHRHOFERWEG 1-5/8/
25"
240 PRINT:PRINT"           A - 1110  W I E N
"
250 PRINT:PRINT"          TEL:0222/ 76 18 225
"
260 FOR I=1TO5000:NEXT
270 CLEAR 5000
280 DIM A$(150)
290 CLS
300 SCREEN0,0
310 PRINT"************ HAUPTMENUE *********
*****":PRINT:PRINT:PRINT
320 PRINT "1 ..... FLEISCH":PRINT
330 PRINT "2 ..... MEHLSPEISEN":PRINT
340 PRINT "3 ..... FERTIGGERICHTE":PRINT
350 PRINT "4 ..... OBST":PRINT
360 PRINT "5 ..... GEMUESE":PRINT
370 PRINT "6 ..... HALTBARKEIT":PRINT
380 PRINT "7 ..... DATEN LESEN":PRINT
390 PRINT "8 ..... DATEN ABSPEICHERN":PRINT
400 PRINT :FOR I=1TO 39:PRINT "*";:NEXT
410 A$=INKEY$:IF A$=""GOTO410ELSEA=VAL(A$)
420 ON AGOTO 430,730,780,830,880,930,1530,14
40
425 CLS:GOTO 310
430 CLS:PRINT "*********** MENUE FLEISCH ***
********"
440 PRINT :PRINT :PRINT
450 PRINT "1 ..... RINDFLEISCH":PRINT
460 PRINT "2 ..... SCHWEINEFLEISCH":PRINT
470 PRINT "3 ..... FISCH":PRINT
480 PRINT "4 ..... GEFLUEGEL UND KANINCHEN":
PRINT
490 PRINT "5 ..... HAUPTMENUE":PRINT
500 PRINT :FOR I=1TO 39:PRINT"*";:NEXT
510 A$=INKEY$:IFA$=""GOTO510ELSEA=VAL(A$)
520 ON AGOTO 530,580,630,680,290
525 GOTO 430
530 CLS :PRINT "*********** RINDFLEISCH ****
*****":X=1:Y=20
540 FOR I=1TO 20
550 PRINT A$(I)
560 NEXT
570 GOTO 1590
580 CLS :PRINT "******* SCHWEINEFLEISCH ****
*****":X=21:Y=40
590 FOR I=21TO 40
600 PRINT A$(I)
610 NEXT
620 GOTO 1590
630 CLS :PRINT "************** FISCH *******
*****":X=41:Y=50
640 FOR I=41TO 50
650 PRINT A$(I)
660 NEXT
670 GOTO 1590
680 CLS :PRINT "**** GEFLUEGEL UND KANINCHEN
 ****":X=51:Y=60
690 FOR I=51TO 60
700 PRINT A$(I)
710 NEXT
720 GOTO 1590
730 CLS :PRINT "********** MEHLSPEISEN *****
*****":X=61:Y=80
740 FOR I=61TO 80
750 PRINT A$(I)
760 NEXT
770 GOTO 1590
780 CLS :PRINT "********** FERTIGGERICHTE **
*****":X=81:Y=100
790 FOR I=81TO 100
800 PRINT A$(I)
810 NEXT
820 GOTO 1590
830 CLS :PRINT "*************** OBST ******
*****":X=101:Y=120
840 FOR I=101TO 120
850 PRINT A$(I)
860 NEXT
870 GOTO 1590
880 CLS :PRINT "************** GEMUESE *****
*****":X=121:Y=140
890 FOR I=121TO 140
900 PRINT A$(I)
910 NEXT
920 GOTO 1590
930 REM ** HALTBARKEIT **
940 CLS:Y=0
950 INPUT "DATUM:";J$
960 FORC=1TO140
970 A$=LEFT$(A$(C),6)
980 R=VAL(LEFT$(J$,2))
990 S=VAL(MID$(J$,3,2))
1000 T=VAL(RIGHT$(J$,2))
1010 U=VAL(LEFT$(A$,2))
1020 V=VAL(MID$(A$,3,2))
1030 W=VAL(RIGHT$(A$,2))
1040 IFY=1THENH=R+1925: GOTO 1060
1050 H=R
1060 G=S: I=T
1070 GOSUB 1150
1080 J=I
1090 IFY=1THENH=U+1925: GOTO 1110
1100 H=U
1110 G=V: I=W
1120 GOSUB 1150
1130 X=I-J
1140 GOTO 1230
1150 IFG-3>0THENZ=-(G-3)*30.6-.5:GOSUB1220:I
=I-Z:GOTO1180
1160 H=H-1
1170 Z=(-(G-3)-12)*30.6-.5:GOSUB1220:I=I-Z
1180 Z=H*365.25:GOSUB1220:I=I+Z
1190 Z=H/100:GOSUB1220:I=I-Z
1200 Z=H/400:GOSUB1220:I=I+Z
1210 I=I-307:RETURN
1220 X=INT(ABS(Z)):Z=SGN(Z)*X:RETURN
1230 IFA$(C)=""GOTO1250
1240 IF X<31THEN BEEP:PRINTA$(C)
1250 NEXT
1260 PRINT "ALLES"
1270 PRINT "BITTE EINE TASTE DRUECKEN"
1280 X$=INKEY$:IFX$="" GOTO 1280
1290 GOTO310
1300 CLS :PRINT "************** EINGABE ****
*********"
1310 FORI=XTOY
1320 IFA$(I)=""THEN 1350
1330 NEXT
1340 PRINT"KEIN PLATZ MEHR":RETURN
1350 INPUTA$(I)
1360 CLS: GOTO 1590
1370 CLS :PRINT "************* HERAUSNEHMEN
*********"
1380 INPUT "ARTIKEL:";A$
1390 FORI=XTOY
1400 IFA$(I)=A$ THENPRINT "LOESCHEN(L)?"ELSE
 NEXT:PRINT"NICHT VORHANDEN": GOTO 1590
1410 B$=INPUT$(1)
1420 IFB$="L" OR B$="l" THENA$(I)="":PRINT"G
ELOESCHT": GOTO 1590
1430 CLS: GOTO 1590
1440 CLS:PRINT"PRESS PLAY AND RECORD ON TAPE
"
1450 MOTOR ON
1460 FORI=1TO5000:NEXT:CLOSE#1
1470 MOTOROFF
1480 OPEN"KUEHL"FOROUTPUTAS#1
1490 FORI=1TO150
1500 PRINT#1,A$(I)
1510 NEXT
```

Fortsetzung auf Seite 23

```
*****************************************
*                                       *
*          Diskettenverwaltung          *
*                                       *
*****************************************
```

Viele BASIC-Programmierer kennen das Problem. Man schreibt schnell irgendein Programm, das man gerade braucht, dann erzeugt man Variationen und Backups - und nach einiger Zeit hat man einen Haufen Disketten, voll mit wenigen Programmen in vielen Versionen. Der Überblick ist allerdings weg. Die nachstehenden Programme helfen Ordnung zu halten.

"dsknam.v01" schreibt einen bis 128 Zeichen langen Namen ins Inhaltsverzeichnis der Diskette neben das Directory in der Spur 20, Sektor 12. Die nicht benötigten Bytes des Sektors werden mit "." angefüllt. Bitte, versuchen Sie nicht, CPM-Disketten mit diesem Programm zu benennen! Sie würden irreparabel geschädigt werden.

"dirtst.v01" erzeugt eine Liste der verfügbaren Files. Man braucht nur die Disketten der Reihe nach zu füttern. Das Programm ist für den SVI-328 geschrieben, User des SVI-318 ohne Speichererweiterung müssen den großzügig angelegten Stringspeicherplatz (Zeile 150) entsprechend verkleinern. In der hier vorgestellten Version kann das Programm etwa 500 Files verwalten, da die Zwischenspeicherung und der Sortiervorgang ausschließlich im Speicher ablaufen. Nach etwa 150 Files macht der Computer eine Pause beim Einlesen der Disketten. Das ist kein Grund zur Panik - er führt nur eine Speicherbereinigung durch. Bitte versuchen Sie nicht, durch irgendwelche Aktionen diesen Vorgang zu beeinflussen.

Nach Einlesen aller Disks werden die Filenamen alphabetisch sortiert, nach Typ (BASIC, Binär, ASCII oder Bildschirmkopie) geordnet und mit der Liste der Disknamen zusammen ausgedruckt. Das Programm ist für einen DELTA 10 eingerichtet, für andere Drucker mag eine Änderung des CHR$(14) (Breitschrift für eine Zeile) nötig sein. In dieser Version werden nur die ersten beiden Zeichen des Diskettennamens ausgedruckt. Durch eine Änderung der Zeile 370 ist das leicht zu ändern.

Das Programm besteht aus folgenden Blöcken:
100-210 Initialisierung, Puffervereinbarung
220-260 Kopfzeile
270-320 Eingabesteuerung
330-380 Diskettenname holen und schreiben
390-590 Filename und Filetyp feststellen
(Nähere Daten siehe Merkblatt: "Files am Spectravideo)
600-700 Sortierer (ein vereinfachter Bubblesort, der sicher zu beschleunigen wäre)
710-1130 Auswahl nach Filetypen, Ausdruck

Programm "TIEFKUEHLTRUHENVERWALTUNG"
Fortsetzung von Seite 22

```
1520 CLOSE#1:MOTOROFF:GOTO 1590
1530 CLS:OPEN "KUEHL"FORINPUTAS#1
1540 IF EOF(1)THENCLOSE#1:END
1550 FORI=1TO150
1560 INPUT#1,A$
1570 A$(I)=A$
1580 NEXT
1590 LOCATE1,22:PRINT"Z=HAUPTMENUE, E=EINGAB
E, H=HERAUS"
1600 I$=INKEY$
1610 IF I$="Z" OR I$="z" THEN 290
1620 IF I$="E" OR I$="e" THEN 1300
1630 IF I$="H" OR I$="h" THEN 1370
1640 GOTO 1600
```

Programm dsknam.v01 --- Diskettenbenennung

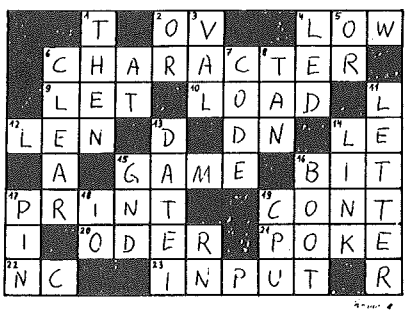
```
100 FIELD #0, 128 AS A$,128 AS B$
110 CLS
120 B1$=STRING$(128,".")
130 PRINT"Dieses Programm dient zur Benennun
g von Basic - Disketten."
140 PRINT"Bitte versuche nicht, CPM - Disket
ten zu benennen!"
150 PRINT
160 PRINT"Bitte Diskette einlegen und belieb
ige Taste drücken!"
170 IF INKEY$="" THEN 170
180 CLS
190 PRINT"Auf der Diskette sind folgende Fil
es:"
200 PRINT
210 FILES
220 PRINT
230 INPUT"Bitte den gewünschten Namen eingeb
en :";A1$
240 A1$=A1$+STRING$(128-LEN(A1$),".")
250 LSET A$=A1$
260 LSET B$=B1$
270 DSKO$ 1,20,12
280 INPUT"Nächste Diskette ";C$
290 IF C$<>"n" AND C$<>"N" THEN 160
```

Programm dirtst.v01 --- Filessortierung

```
100 REM
110 REM Erzeugung einer Liste der verfügbare
n Files
120 REM copyright Wolfgang Tomischko
130 REM
140 CLS
150 CLEAR 15000
160 DIM A$(16),M$(500,1)
170 FIELD #0, 16 AS A$(1),16 AS A$(2),16 AS
A$(3),16 AS A$(4),16 AS A$(5),16 AS A$(6),16
 AS A$(7),16 AS A$(8),16 AS A$(9),16 AS A$(1
0),16 AS A$(11),16 AS A$(12),16 AS A$(13),16
 AS A$(14),16 AS A$(15),16 AS A$(16)
180 ZA%=1
190 REM
200 REM Überschrift
210 REM
220 INPUT"Datum, Uhrzeit";DA$,DU$
230 PRINT
240 LPRINT CHR$(14);"Fileliste vom ";DA$;" u
m ";DU$
250 LPRINT
260 PRINT
270 PRINT"Bitte Disketten füttern und <enter
> drücken. Falls keine weiteren Disketten
  bearbeitet werden, <q> drücken!"
280 PRINT
290 KK$=INKEY$
300 IF KK$="" THEN 290
310 IF KK$=CHR$(13) THEN 360
320 IF KK$="q" THEN 600
330 REM
340 REM Diskettenname holen
350 REM
360 DUMMY$=DSKI$(1,20,12)
370 N$=LEFT$(A$(1),2)
380 PRINT"diskettenname ";N$
390 REM
400 REM Filenamen holen
410 REM
420 FOR S%= 1 TO 3
430 DUMMY$=DSKI$(1,20,S%)
440 FOR I%=1 TO 16
450 IF LEFT$(A$(I%),1)=CHR$(0) THEN 580
460 IF LEFT$(A$(I%),1)=CHR$(255) THEN N$="":
GOTO 260
470 B%=ASC(MID$(A$(I%),10,1))
480 C$=""
490 IF B%>127 THEN C$="."
500 IF B%=0 OR B%=16 OR B%=64 OR B%= 80 THEN
 C$=" "
510 IF B%=32 OR B%=48 OR B%=96 OR B%=112 THE
N C$="#"
520 IF B%=1 OR B%=17 OR B%=65 OR B%= 81 THEN
 C$="*"
```

```
530 F$=LEFT$(A$(I%),6)+C$+MID$(A$(I%),7,3)
540 PRINT "File Nummer ";ZA%;"  ";F$
550 M$(ZA%,0)=F$
560 M$(ZA%,1)=N$
570 ZA%=ZA%+1
580 NEXT I%
590 NEXT S%
600 REM
610 REM sort
620 REM
630 FOR N%=ZA% TO 1 STEP -1
640 C%=1:PRINT N%
650 FOR M%=1 TO N%
660 IF M$(M%,0)>M$(C%,0) THEN C%=M%
670 NEXT M%
680 SWAP M$(N%,0),M$(C%,0)
690 SWAP M$(N%,1),M$(C%,1)
700 NEXT N%
710 REM
720 REM Ausdruck
730 REM
740 PRINT
750 LPRINT
760 PRINT "Basic Programmfiles"
770 LPRINT CHR$(14);"Basic Programmfiles"
780 FOR I%=1 TO ZA%
790 IF MID$(M$(I%,0),7,1)<>"." THEN 830
800 IF M$(I%,0)=M$(I%-1,0) THEN PRINT " ";M$
(I%,1);:LPRINT " ";M$(I%,1);:GOTO 830 ELSE P
RINT:LPRINT
810 PRINT M$(I%,0),M$(I%,1);
820 LPRINT M$(I%,0),M$(I%,1);
830 NEXT I%
840 PRINT:PRINT
850 LPRINT:LPRINT
860 PRINT "ASCII Files"
870 LPRINT CHR$(14);"ASCII Files"
880 FOR I%=1 TO ZA%
890 IF MID$(M$(I%,0),7,1)<>" " THEN 930
900 IF M$(I%,0)=M$(I%-1,0) THEN PRINT " ";M$
(I%,1);:LPRINT " ";M$(I%,1);:GOTO 930 ELSE P
RINT:LPRINT
910 PRINT M$(I%,0),M$(I%,1);
920 LPRINT M$(I%,0),M$(I%,1);
930 NEXT I%
940 PRINT:PRINT
950 LPRINT:LPRINT
960 PRINT "Binaerfiles"
970 LPRINT CHR$(14);"Binaerfiles"
980 FOR I%=1 TO ZA%
990 IF MID$(M$(I%,0),7,1)<>"*" THEN 1030
1000 IF M$(I%,0)=M$(I%-1,0) THEN PRINT " ";M
$(I%,1);:" ";LPRINT M$(I%,1);:GOTO 1030 ELSE
 PRINT:LPRINT
1010 PRINT M$(I%,0),M$(I%,1);
1020 LPRINT M$(I%,0),M$(I%,1);
1030 NEXT I%
1040 PRINT:PRINT
1050 LPRINT:LPRINT
1060 PRINT "Bildschirmkopie"
1070 LPRINT CHR$(14);"Bildschirmkopie"
1080 FOR I%=1 TO ZA%
1090 IF MID$(M$(I%,0),7,1)<>"#" THEN 1130
1100 IF M$(I%,0)=M$(I%-1,0) THEN PRINT " ";M
$(I%,1);:" ";LPRINT M$(I%,1);:GOTO 1130 ELSE
 PRINT:LPRINT
1110 PRINT M$(I%,0),M$(I%,1);
1120 LPRINT M$(I%,0),M$(I%,1);
1130 NEXT I%
1140 FOR I%=1 TO 33:LPRINT:NEXT I%
1150 STOP
```

Auflösung des Rätsels aus Heft 2/85:




```
**********************************************
*                                            *
*              Hexdump-Monitor               *
*                                            *
**********************************************
```

Es kann einen frustieren, wenn man just gerade keinen Assembler besitzt, aber unbedingt ein MC-Programm ausprobieren möchte. Die im SVI-Journal abgedruckten Hexdumps sehen zwar praktisch aus, aber wehe, wenn man probiert, ein 200 Byte-Programm mit dieser Vorlage einzutippen. Man kennt sich in kürzester Zeit nicht mehr aus, weiß nicht mehr, in welcher Zeile man gerade war, etc. Ebenso ist es umständlich, die über die ganze Tastatur verstreuten 16 Hexadezimal-Ziffern zu verwenden. Wie kann man nun Abhilfe schaffen?

Antwort: Man legt die Zeichen "A", "B", "C", "D", "E" und "F" auf die Cursor- und PRINT/SELECT-Tasten über die Zehnertastatur. Man hat nun ein komprimiertes Hexzahlen-Tastenfeld. Wenn man nun die Eingabe des Maschincode-Programms und dessen Korrektur auf das Hexdump-Format zuschneidert, dann kann man die Vorteile so eines Dumps genießen.

Die Bedienung des Programms ist simpel. Am Anfang wird die Startadresse eingegeben. Danach kann man gleich mit dem MC-Programm beginnen. Hat man einen kleinen Fehler begangen, so kann man ihn mit ESC korrigieren. Wird ESC gedrückt, wenn schon ein Nibble eines Bytes eingegeben wurde, so wird nur dieses eine Nibble gelöscht. Andernfalls korrigiert das Programm das ganze soeben definierte Byte.

Nachdem die Hexdump-Eingabe "vollbracht" ist, drückt man ENTER. Nun kann man im Speicher blättern, um noch irgendwelche Korrekturen anzubringen. Dazu fährt man mit dem Cursor zur angezeigten Stelle, ändert sie und tippt ENTER.

Ist man auch mit den Korrekturen fertig, kann man das MC-Programm noch abspeichern. Zum Schluß wird der Anfang des Programms definiert. Man braucht nur A=USR(0) eingeben und sieht die Entwicklung in der Praxis.

Sehen wir uns nun den Aufbau des Programms an. Das Programm ist in drei Teile geteilt. Das Hauptprogramm erledigt die Eingabe und die Speicherung der Bytes. Der zweite Teil sorgt für ein Korrigieren der Eingabe per Bildschirmeditor. Zuletzt gibt es noch den unkompliziertesten Teil, der das Maschincodeprogramm abspeichert und definiert.

Der erste Teil besteht aus einer Schleife, die pro Schleifendurchlauf zweimal eine Subroutine anspringt. Dieses Unterprogramm ist für die eigentliche Verarbeitung der Zeichen zuständig. Zuerst wird ein Zeichen eingelesen. Dies wird einerseits durch INKEY$ erledigt. Da aber die PRINT- und die SELECT-Taste nicht auf INKEY$ reagiert, muß zusätzlich noch die Speicherzelle &HFD7D ausgelesen werden. In dieser Speicherzelle befindet sich die Information codiert, ob die oben erwähnten Tasten gedrückt wurden. Ebenso wird dadurch die Cursor-Taste "Rechts" eruiert. Nachdem eine Taste betätigt wurde, wird dies registriert und getestet, ob eine der Tasten eingegeben wurde, die gebraucht wird. In der Variablen R wird dann der Wert des gewünschten Nibbles erzeugt. Danach zeigt der Computer das Zeichen an und speichert das definierte Byte ab.

Die Tasten ESC und ENTER rufen eigene Routinen ab. ESC (ab Zeile 48) korrigiert eine Fehleingabe und ENTER wechselt in den Bildschirmeditor-Modus über.

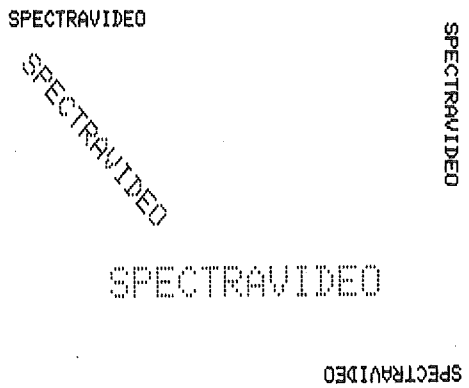
Nach der Eingabe von ENTER frägt der Computer, ob eine Änderung gewünscht sei. Bei einem "J" wird der Editor-Modus aufgerufen, bei dem allerdings die Hex-Tastatur wieder auf das Buchstabenfeld verstreut ist. Man braucht ja die Cursor-Tasten zum Positionieren des Cursors.

Interessant ist vielleicht noch, daß man beliebig lange am Bildschirm editieren kann, erst wenn ein ENTER gedrückt wird, speichert der Computer die Korrekturen ab. Das Programm nimmt beim Abspeichern nur die vorgesehenen Bytes vom Bildschirm. Hat man versehentlich zwischen zwei Bytes Zeichen gesetzt, so werden diese ignoriert.

Der letzte Teil, das Abspeichern, enthält ebenso einen kleinen Trick. Da beim Abspeichern mit BSAVE eine Fehlermeldung entsteht, wenn man als Dateinamen den Leerstring verwendet, kann man dies dazu benützen, um eine Abfrage "Abspeichern JA/NEIN" elegant zu lösen. Man läßt den Computer durch ON ERROR GOTO zum Ende springen. Ist hingegen ein Abspeichern erwünscht, führt der Computer ohnehin das BSAVE ohne Klagen aus.

```
1 REM **********************************
2 REM *                                *
3 REM *        HEX-DUMP-HILFE          *
4 REM *         VERSION 2.1            *
5 REM *        (c) by G. Fally         *
6 REM *                                *
7 REM **********************************
8 DEFFNP$(I)=STRING$(2-LEN(HEX$(PEEK(I))),"0
")+HEX$(PEEK(I))+" ":SCREEN0,0
9 INPUT"Bitte Startadresse angeben";I
10 CLS:ST=I
11 PRINTHEX$(I)" ";:FORT=1TO8:GOSUB14:GOSUB1
4:PRINT" ";:I=I+1:NEXT
12 PRINT:GOTO11
13 REM
14 REM *** ZEICHEN LESEN ***
15 REM
16 R=0:IFPEEK(&HFD7D)<>255THEN16
17 E=PEEK(&HFD7D):IFE=255THEN22
18 IFE=223THENR=15
19 IFE=239THENR=13
20 IFE=127THENR=12:S$=INKEY$
21 IFRTHENGOTO30ELSE16
22 S$=INKEY$:IFS$=""THEN17ELSEQ=ASC(S$):IFQ=
27THENGOSUB54:RETURN
23 IFQ=13THENI=ST:GOTO36ELSEIFQ<28THEN16
24 ONQ-27GOTO25,26,27,28:GOTO29
25 R=12:POKE&HFD7D,120:GOTO30
26 R=10:GOTO30
27 R=14:GOTO30
28 R=11:GOTO30
29 IFQ<58ANDQ>47THENR=Q-48ELSE16
30 PRINTHEX$(R);:IFSTHENM=M*16+R:S=0:POKEI,M
ELSEM=R:S=1
31 RETURN
32 REM
33 REM *** KORREKTUREN ***
34 REM
35 CLS:FORT=1TO20:PRINTHEX$(I)" ";:FORY=1TO8
:PRINTFNP$(I);:I=I+1:NEXT:PRINT:NEXT:PRINT:X
1=6:Y1=0:RETURN
36 GOSUB35:LOCATE0,21:PRINT"AENDERUNGEN GEWU
ENSCHT (J/N)? "+CHR$(27)+"J";:S$=INPUT$(1):P
RINTS$
37 IF S$="J"ORS$="j"THEN39
38 IFS$="N"ORS$="n"THEN46ELSEI=I-160:GOTO36
39 LOCATE0,22,1:PRINT"Vor/Zurueck (V/Z)? "+C
HR$(27)+"J";:ME=Y1*40+X1:VPOKEME,VPEEK(ME)+9
6
40 S$=INPUT$(1):VPOKEME,VPEEK(ME)-96:Q=ASC(S
$):LOCATE,,0:IFQ<80THEN43
41 IFS$="Z"ORS$="z"THENI=I-320:GOSUB35ELSEIF
S$="V"ORS$="v"THENGOSUB35ELSE39
42 GOTO39
43 IFQ>27ANDQ<32THENLOCATEX1,Y1:PRINTS$;:X1=
POS(0):Y1=CSRLINELSEIFQ=13THEN57
44 IF(Q<59ANDQ>47)OR(Q>64ANDQ<70)THENLOCATEX
1-1,Y1:PRINTS$;:X1=X1+1:GOTO39ELSEGOTO39
45 POKESE,IN:I=I-160:GOTO36
46 CLS:PRINT"Name fuer MC-Datei eingeben!
47 PRINT"Wenn Speichern nicht erwuenscht,
    3 ENTER druecken
48 ON ERROR GOTO50
49 S$="":INPUTS$:INPUT"Anfangsadresse";Q:INP
UT"Endadresse";W:BSAVES$,Q,W
50 DEFUSR=ST:PRINT"Um MC-Programm zu starten
, A=USR(0) tippen!":END
51 REM
52 REM *** ESC-ROUTINE ***
53 REM
54 IFSTHENS=0:LOCATEPOS(0)-1,CSRLIN:GOSUB56:
GOSUB16ELSEI=I-1:IFT=1THENLOCATE26,CSRLIN-1:
GOSUB56:T=8ELSELOCATEPOS(0)-3,CSRLIN:T=T-1:G
OSUB56
55 GOTO16
56 PRINTCHR$(27)+"J";:RETURN
57 Y1=0:FORM=I-160TOI-2STEP8:X1=6
58 FORY=0TO7:B=0:FORT=1TO2:A=VPEEK(X1+Y1)
59 GOSUB64:NEXT:POKEM+Y,B:X1=X1+1:NEXT:Y1=Y1
+40:NEXT
60 REM
61 REM *** NIBBLE-BERECHNUNG
62 REM
63 I=I-160:GOTO36
64 IFA>95THENA=A-96
65 A=A-16:IFA>9THENA=A-7
66 B=B*16+A:X1=X1+1:RETURN
```

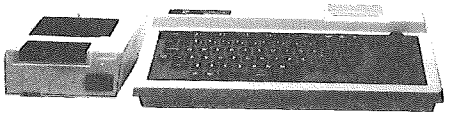
Das von Herrn Banga aus München eingesandte Programm eignet sich zum Schreiben in verschiedenen Schriftrichtungen:




```
10 SCREEN 1,0:DEFINT A-Z
20 PRINT " SPECTRAVIDEO "
30 DEF FNX=240-Y:DEF FNY=X:GOSUB 100
40 DEF FNX=250-X:DEF FNY=190-Y:GOSUB 100
60 DEF FNX=12+X-Y:DEF FNY=12+Y+X:GOSUB 100
70 DEF FNX=50+X*2:DEF FNY=140+Y*2:GOSUB 100
90 GOTO 200
100 FOR X=0 TO 77
110 FOR Y=0 TO 7
120 DOT=POINT(X,Y)
130 IF DOT=0 THEN 150
140 PSET (FNX,FNY),DOT
150 NEXT Y,X
160 RETURN
180 GOTO180
200 K=0:S=0:Z=0:R=8
210 LPRINT CHR$(27)"1"
220 LPRINT CHR$(27)"K"CHR$(1);CHR$(1)
230 FOR X=0 TO 255
240 FOR Y=0 TO 7
250 A=POINT (X,Y+K)
260 IF A=4 THEN 300
270 Z=2^(7-Y)
280 S=S+Z
300 NEXTY
310 LPRINTCHR$(S);
320 S=0
330 NEXTX
350 K=K+R
360 IF K<192 THEN 210
370 END
```

Die Super-Computer.

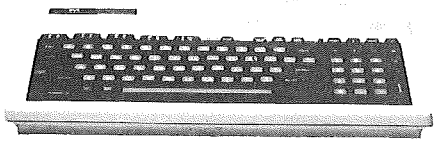
SVI



SVI-318 32 K RAM, erweiterbar bis **144 K RAM.** Erweitertes MICROSOFT-BASIC, integrierte Cursorsteuerung **öS 4.990,—**

SVI-904 Datenrecorder, 1800 Baud, Zählwerk, Laufwerksteuerung durch SVI-318 oder 328 inkl. 2 Spielkassetten **öS 990,—**

SVI-318-Set bestehend aus SVI-318 Basisgerät (32 K RAM, MICROSOFT-BASIC), SVI-904 Datenrecorder und Softwarepaket mit 5 Kassetten **öS 5.890,—**



SVI-328 32 K ROM, 80 K RAM, Erweitertes MICROSOFT-BASIC, Schreibmaschinentastatur, 10 Funktionstasten, 10er-Block **öS 6.990,—**

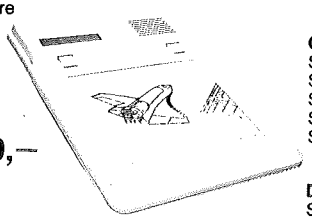
Super-Expander SVI-605, ein eingebautes Diskettenlaufwerk (160 K), Centroniks-Interface, 4 freie Steckplätze, Betriebssystem CP/M 2.2 **öS 12.990,—**

Super-Expander SVI-605 A, zwei eingebaute Diskettenlaufwerke (je 160 K), Centronics-Interface, 4 freie Steckplätze, Betriebssystem CP/M 2.2 **öS 18.990,—**

Super-Expander SVI-605 B, mit Supersoftware-Paket, zwei eingebaute Diskettenlaufwerke (je 320 K), Centronics-Interface, 4 freie Steckplätze, Betriebssystem CP/M 2.2 WordStar, Mailmerge, CalcStar ReportStar, DataStar **öS 25.990,—**

SVI-328 Pro Profisystem bestehend aus: Computer SVI-328, Super-Expander SVI-605 A (inkl. WordStar, Mailmerge, CalcStar, DataStar, ReportStar) Betriebssystem CP/M 2.2, 80-Zeichenkarte SVI-806, **öS 32.290,—**

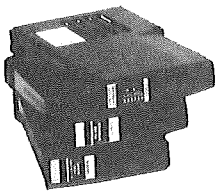
Grafik-Tablett SVI-105, 186 x 158 mm Zeichenfläche, Kassette mit Anwender-Software inkl. **öS 2.090,—**



Erweiterungskarten für SVI-605, A, B

SVI-803 16 K-Speicher-Erweiterung (für SVI-318) **öS 890,—**

SVI-805 RS 232, serielle Schnittstelle **öS 1.790,—**



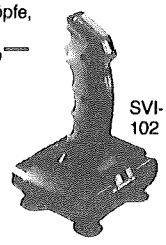
SVI-806 80-Zeichenkarte **öS 2.390,—**

SVI 807 64-K-RAM Speichererweiterung **öS 3.290,—**

Joystick SVI-101, zwei Feuerknöpfe, vier Saugfüße, ergonomischer Handgriff **öS 390,—**

Joystick SVI-102, automatisches Dauerfeuer, zwei Feuerknöpfe, vier Saugfüße **öS 490,—**

(Joystick SVI-101 und SVI-102 auch für Atari und Commodore geeignet)



SVI-102

Die Software

		öS
Kassettensoftware		
SVI-K 110	Einführung in das SVI-Basic inkl. 40seitigem Handbuch	390,—
SVI-K 115	SVI-Dateiverwaltung	290,—
SVI-K 122	SVI-Text	390,—
SVI-K 129	SVI-Termin	290,—
SVI-K 146	Disassembler	590,—
SVI-K 147	Maschinen-Code-Monitor	590,—
SVI-K 148	SVI-Spritegenerator	290,—
SVI-K 149	SVI-Zeichengenerator	290,—
SVI-K 179	Old-Mac-Farmer	390,—
SVI-K 180	Tetra Horror	390,—
SVI-K 181	Tele Bunny	390,—
SVI-K 182	Turboat	390,—
SVI-K 183	SASA	390,—
SVI-K 184	NINJA	390,—
SVI-K 185	Kung-Fu-Master	390,—
SVI-K 188	Armoured Assault	290,—
SVI-K 189	Spectron	290,—
Cartridgesoftware		
SVI-C 220	Sector Alpha	790,—
SVI-C 232	Frantic-Freddy	790,—
SVI-C 236	Music-Mentor	990,—
SVI-C 237	Super-Cross-Force	790,—
SVI-C 291	Flipper-Slipper	790,—
Diskettensoftware		
SVI-D 310	Einf. in das SVI-Basic	590,—
SVI-D 315	SVI-Dateiverwaltung	390,—
SVI-D 322	SVI-Text	590,—
SVI-D 334	SVI-Lager	390,—
SVI-D 348	SVI-Toolkit I (SVI-Spritegenerator u. SVI-Zeichengenerator)	590,—
SVI-D 349	SVI-Toolkit II (Disassembler und Maschinen-Code-Monitor)	1.190,—
SVI-D 359	LISP 80	1.690,—
SVI-D 360	C-Compiler	1.690,—
SVI-D 361	Turbo-PASCAL (Version 2.0)	2.390,—
SVI-D 318	Nevada-FORTRAN (Compiler)	1.390,—
SVI-D 382	Nevada-COBOL (Compiler)	1.390,—
SVI-D 383	Nevada-PILOT (Interpreter)	1.390,—
SVI-D 384	Nevada-EDIT (Editor)	1.390,—
SVI-D 388	Old-Mac-Farmer	390,—
SVI-D 389	Tetra Horror	390,—
SVI-D 390	Tele Bunny	390,—
SVI-D 391	Turboat	390,—
SVI-D 392	SASA	390,—
SVI-D 393	NINJA	390,—
SVI-D 394	Kung-Fu-Master	390,—

Druckeranschlußkabel SVI-205, 1,5 m, für parallele Schnittstelle **öS 590,—**

Diskettenlaufwerk SVI-905, 160 K, zur Erweiterung des Super-Expanders SVI-605 **öS 6.490,—**

Centronics-Interface SVI-802 mit Kabel 205 zum Anschluß an Mini-Expander SVI-602 **öS 3.080,—**

Preise incl. MWSt.

SVI-FACHBERATUNG UND VERKAUF BEI:

Großhandel, Facheinzelhandel **BASTL-COMPUTER-SYSTEME** **2700 Wr. Neustadt,** Hauptplatz 5 Tel. (0 26 22) 22 7 20 - 59 80 Telex 16 522	**DAHMS-PRAKTIKER-ELEKTRONIK** Pilgramgasse 11 **1050 Wien** Tel. (0 22 2) 54 34 21	**ESV ELEKTROTECHNISCHER SERVICE MBH.** Bayerhamerstraße 19—21 **5020 Salzburg** Tel. (0 66 2) 74 7 51	**SCHILLER MICRO-COM-BOUTIQUE** Fasangasse 21 **1030 Wien** Tel. (0 22 2) 78 35 99, 78 56 61	**TARGET ELECTRONIC** Bergstraße 6 **6900 Bregenz** Tel. (0 55 74) 23 7 18
BYTE COMPUTER Favoritenstraße 20 **1040 Wien** Tel. (0 22 2) 65 73 42 Telex 132 827	**EDV-STUDIO PORSCH** Kinderspittalgasse 13 **1090 Wien** Tel. (0 22 2) 42 63 44	**FEDCON ELEKTRONIK** Ing. Franz Krenn Kanalplatz 86 **9400 Wolfsberg** Tel. (0 43 52) 42 73	**TARGET ELECTRONIC** Reichsstraße 123a **6800 Feldkirch** Tel. (0 55 22) 21 5 29 Telex 52 300	**WEHSNER GMBH. COMPUTER-STUDIO** Paniglgasse 18—20 **1040 Wien** Tel. (0 22 2) 65 88 93, 65 78 08

GENERALVERTRETUNG FÜR ÖSTERREICH: MONACOR ELECTRONIC - VERTRIEBS-GESMBH. · 6800 FELDKIRCH · ☎ (0 55 22) 21 9 89 · TELEX 52 300 tamo

1	DAHMS-HANDBUCH 1985, ca. 600 Seiten, ca. 8000 Bauteile; mit Preisliste	98,--
2	EPROM-PROGRAMMER auf VC2o / C64 mit Programmdiscette	1.990,--
3	EPROM- LÖSCHGERÄT - klein, handlich und betriebssicher	1.290,--
4	SPEZIAL-TECHNIKERLEUCHTE, biegsam, für Tisch und Hand, mit Magnet	199,--
5	BLITZSCHUTZSTECKER - ideal für Computer, Kassen etc. -	173,--
6	TRAVELLAR-REISESTECKER für alle Steckerarten im Ausland	42,--
7	DMM 3 1/2 stellig, FLUKE 73, automatische Bereichswahl	2.490,--
8	DMM 3 1/2 stellig, FLUKE 77, wie 73 + Durchgangssummer + Meßwertspeicher	3.990,--
9	DMM 3 1/2 stellig, SOAR 550, Autoranging	990,--
10	DMM 3 1/2 stellig, SOAR 540, Autoranging + Manuell	1.190,--
11	DMM 3 1/2 stellig, SOAR 530, Autoranging + Manuell + Auflösung 0,1 µA	1.490,--
12	DMM 3 1/2 stellig, SOAR 3100, "PEN TYPE", Datahold, Autoranging	1.430,--
13	NORMATEST 3000, Analog-Spiegelskala, 41 Meßbereiche, MADE IN AUSTRIA	1.765,--
14	DMM 3 1/2 stellig, NORMA D 1214, ÖSTERREICHISCHES QUALITÄTSPRODUKT	2.250,--
15	ANSTECKNEBENWIDERSTAND zu NT3000, 2 - 20 A	480,--
16	KAPAZITÄTSMESSVORSATZ zu SOAR 530 bis 550 und alle Geräte, 2 nF-200 µF	1.690,--
17	FC 841 Digital-Frequenzzähler bis 50 Mhz	1.815,--
18	FC 845 Digital-Frequenzzähler bis 150 Mhz	2.950,--
19	ZANGENAMPEREMETER SOAR 2210, 3 1/2 stellig	2.590,--
20	UV-BELICHTUNGSGERÄT I, Fläche 460 x 180	2.180,--
21	PLATINENÄTZGERÄT II - mit Motorpumpe	1.880,--
22	"DALO" ZEICHENSTIFT, ätzfest für Platinen	99,--
23	EISEN-III-CHLORID, Packung in fester Form, löslich in 1/2 l Wasser	25,--
24	COMPUTERPROGRAMM Databecker "Datamat" für C 64	790,--
25	COMPUTERPROGRAMM Databecker "Textomat" für C 64	790,--
26	COMPUTERPROGRAMM Databecker "Faktumat" für C 64	790,--
27	COMPUTERPROGRAMM Databecker "Synthimat" für C 64	790,--

28	SILICON-SCHLAUCHSORTIMENT, FTK 110, 0,5 - 4 Ø, sortiert		99,--
29	KLEINMOTOR für 6 - 12 V DC, mit Triebrad		35,--
30	STECKERKLEINNETZGERÄT 220 V AC/ 9 V DC		29,--
31	MECHANISCHE ZÄHLWERKE, 3 stellig oder 4 stellig		18,--
32	LÖTZINN "Multicore Savbit", 1/2 kg, besonders günstig		195,--
33	-BERNSTEIN-ELEKTRONIK-FLACHZANGE- isoliert, schwarz		195,--
34	LÖTSAUGPUMPE, hohe Saugleistung		98,--
35	KLEINLAUTSPRECHER SP 1 R, 8 Ohm, 0,2 Watt		35,--
36	AUTORADIO-"SNAP IN" - HALTERUNG		15,--
37	AUTORADIO- "ZWILLINGSLEITUNG", ca. 4,5 m		15,--
38	LITZENDRAHT AYL 1 x 0,75, 1 Bund = 25 m		15,--
39	TV-ANTENNEN-STECKER + -KUPPLUNG, gerade		18,--
40	TV-ANTENNEN-STECKER + -KUPPLUNG, gewinkelt		18,--
41	SCART STECKER + EINBAUBUCHSE, 20polig		55,--
42	ZENERDIODE, 0,4 Watt, 10 Volt	50 Stück	59,--
43	DIODE, OA 95	20 Stück	59,--
44	LEUCHTDIODE 5 Ø, rot, matt	100 Stück	139,--
45	LEUCHTDIODE 5 Ø, grün, matt	50 Stück	85,--
46	LEUCHTDIODE 5 Ø, gelb, matt	50 Stück	85,--
47	TRANSISTOR 2SC1051, TO3, NPN, 120 V, 12 A, 80 W		15,--
48	TRANSISTOR 2SB654A, TO3, PNP, 100 V, 12 A, 60 W		15,--
49	TRANSISTOR 2SC2199, TO3, NPN, 150 V, 8 A, 60 W		15,--
50	TRANSISTOR BC548C, TO92, NPN, 30 V, 0,1A	200 Stück	99,--
51	IC- SN 8474N		9,--
52	IC- MM 57105		9,--
53	IC- FZH101A, DTL, 4 NAND-Glieder		3,--
54	1000,--S-GUTSCHEIN für alle "DAHMS-PRAKTIKER"-Waren		980,--

DAHMS praktiker elektronik

FÜR AKTIVE UND PASSIVE BAUELEMENTE
WERKZEUG UND ZUBEHÖR
U.V.A.M.
0222/54 34 21, 54 33 21

ANGEBOTE

FACHBÜCHER, FACHZEITSCHRIFTEN

WENN SIE EINEN VERLÄSSLICHEN PARTNER BRAUCHEN,

DER IHREN SCHNELLEN BEDARF DECKEN KANN.

DAHMS-PRAKTIKER-ELEKTRONIK
1050 Wien, Pilgramgasse 11

DAHMS praktiker elektronik

Wir sind Spezialisten für SVI-Computer

SVI-Computer kauft man nicht irgendwo, sondern im Computer-Studio.
Denn wir können bereits auf fast zwei Jahre SVI-Erfahrung zurückblicken und beraten Sie daher bestens beim Kauf eines SVI-Computers.
Vergleichen Sie die Computer derselben Preisklasse: Sie werden keinen besseren als Spectravideo finden. Auch bei den MSX-Computern ist Spectravideo mit dem SVI-728 wieder führend: Schreibmaschinentastatur mit separatem Zehnerfeld, 80 KByte-RAM-Speicher, Floppylaufwerk 5 1/4" usw.
Wir zeigen Ihnen die Unterschiede zwischen den einzelnen SVI-Computern und machen Ihnen die Auswahl leicht.
Übrigens wir verkaufen nicht nur Peripheriegeräte und Peripheriekarten, wir produzieren auch welche.
Durch unser großes Lager können wir immer prompt liefern.

SVI-328 **SVI-728**

jetzt besonders günstig prompt lieferbar

Neu: EPROM-Programmierkarte (bis 27256)
I/O-Karte
Hardware-Uhr
Experimentierplatine

T U R B O P A S C A L 2 . 0

angepaßt auf SVI-328 prompt lieferbar, mit Texteditor und ausführlichem Handbuch in d e u t s c h e r Sprache nur S 1.890,-
TOOL BOX mit deutschem Handbuch S 1.890,-

Unterlagen zur Fernseh-Computerfamilie g r a t i s im Computer-Studio.

Computer-Studio

PANIGLGASSE 18 · A-1040 WIEN · TEL. (0222) 65 88 93

Was bietet Ihnen der Spectra Video Club Austria?

- regelmäßige Clubabende mit Gelegenheit zum Informationsaustausch
- 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
- kostenloser Bezug der monatlich erscheinenden Clubzeitschrift SVI-JOURNAL
- verbilligte Angebote von Spectravideo-Produkten

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: Rudolf Bolek, Philipp Ott, Rafael Razim, 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