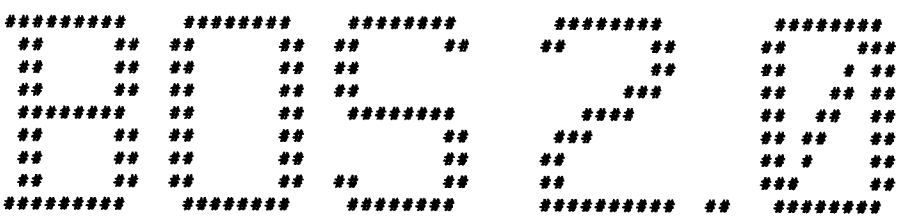
vortex
BOS 2.0

Benutzerhandbuch

vortex®
COMPUTERSYSTEME



B E N U T Z E R H A N D B U C H

Rev. 1 / 10.11.1986

1. Auflage

Falterstraße 51-53
D-7101 Flein bei Heilbronn

Tel.: 07131/52061

DAS INHALTSVERZEICHNIS

BOS 2.0 - DAS NEUE BETRIEBSSYTEM DER VORTEX SPEICHERERWEITERUNG

Jetzt haben Sie es in den Händen, das neue BOS 2.0 . Für viele Besitzer der vortex-Speichererweiterung zum CPC 664 ist es das erste Basic-Betriebssystem für ihre bislang nur unter CP/M nutzbare Speichererweiterung, und für noch mehr Anwender der Speicherkarte am CPC 464 ist es eine wesentliche Erweiterung der Möglichkeiten, von der Sie bislang nur zu hoffen wagten.
Endlich kann man aus der Original-Konfiguration den Speicher der Erweiterungskarte sofort nutzen, ohne eine spezielle Initialisierung aufzurufen, nach der in der Originalbank nur noch 32 KB Speicher übrig sind. Der Drucker-Spooler, Peek- und Poke-Befehle sind sofort nutzbar.
Was aber das Wesentlichste ist: die unter CP/M bekannte Ramdisk ist nun auch unter Basic verfügbar. Und zwar mit den kompletten Vorteilen des VDOS-Betriebssystems.
Trotz allem ist die Möglichkeit der Erweiterung des Programmspeichers nicht unter den Tisch gefallen: Sie können weiterhin zusätzliche Basic-Programmbänke installieren, es gibt aber nicht mehr ein "Alles oder Nichts". Bislang konnte man unter BOS 1.0 nur die maximal mögliche Anzahl an zusätzlichen Basic-Bänken einbinden. Dies können Sie jetzt zwar auch noch, aber wenn Sie beispielsweise bei der SP512 keine 8, sondern nur 3 zusätzliche Basic-Bänke benötigen, so ist das problemlos möglich. Dies ist sehr vorteilhaft, denn der Rest der Speichererweiterung verbleibt dann weiterhin für Ramdisk und Drucker-Spooler.
Apropos Verbleiben: der Speicher ist frei zwischen Basic, Ramdisk, Ramdatei, Videospeicher und Drucker-Spooler aufteilbar. Wenn Sie also einen 512 KB großen Drucker-Spooler benötigen, so ist dies einfachst durch Sie zu initialisieren.
Und als letzte Neuerung: der Video-Mode wurde stark verbessert. Insgesamt kann man nun bis zu 32 Bilder in der Speichererweiterung ablegen und mit zwei neuen Befehlen wesentlich schneller sichtbar machen. Eine Bildlade-Frequenz von circa 10 Bildern pro Sekunde ist nun möglich, was sogar bei einer entprechenden Bildfolge eine fließende Bewegung ergibt.

Aber genug der Vorrede. Es kribbelt Sie garantiert schon in den Fingern, alle diese neuen Möglichkeiten auszuprobieren. Deshalb möchte ich Ihnen auch sofort beschreiben, wie Sie den neuen Eprom in Ihre Speichererweiterung einsetzen.

DER EINBAU DES EPROMS IN DIE SPEICHERERWEITERUNG

1. Schalten Sie sämtliche Geräte Ihres Computersystems ab und stecken sie sämtliche an Ihre CPC-Konsole angeschlossene Peripherie und Kabel ab, damit Sie die Konsole frei bewegen können.

2. Legen Sie die Konsole mit der Oberseite nach unten auf eine weiche Unterlage und lösen Sie mit einem Kreuzschraubenzieher die sechs (beim CPC 664 sieben) Kreuzschrauben, die das Bodenteil mit dem Oberteil verbinden.

3. Drehen Sie die Konsole jetzt vorsichtig um. Dabei werden die gelösten Schrauben herausfallen. Sammeln Sie die Schrauben ein und legen Sie sie beiseite. Die Konsole sollte jetzt so vor Ihnen liegen, wie sie auch im Betrieb vor Ihnen liegt.

4. Beim CPC 464 klappen Sie jetzt das Oberteil vorsichtig nach vorne weg, beim CPC 664 nach links. Sie sehen jetzt in beiden Fällen die CPC-Grundplatine mit eingebauter Speichererweiterungsplatine.

5. Wenn Sie jetzt auf die Bestückungsseite der Speicherer weiterung schauen, sehen Sie an der linken Seite der Platine zwei IC- oder Sockelreihen mit je acht 16-füßigen ICs oder Sockel. Dies sind die beiden Speicherbänke. Direkt daneben ist eine 28-polige Fassung, die beim CPC 664 leer ist und in der beim CPC 464 das Eprom mit dem BOS 1.0 steckt.

6. Wenn Sie vor einem CPC 664 sitzen, dann lesen Sie doch bei Punkt 7 weiter, denn an dieser Stelle müssen die CPC 464-Besitzer Ihren BOS 1.0-Eprom aus dem Sockel holen. Das geht eigentlich recht einfach: Nehmen Sie einen kleinen Geradschlitzschraubenzieher und hebeln Sie mit diesem den Eprom langsam und vorsichtig aus seiner Fassung. Am besten wechseln Sie zwischen den beiden Seiten sogar ab.

7. Nun nehmen Sie den neuen BOS 2.0-Eprom zur Hand und stecken Ihn in diese Fassung. Sie müssen dabei beachten, daß Sie ihn "richtigherum" einstecken. Der Eprom hat nämlich eine kleine Kerbe, die zu Ihnen, also in Richtung Platinenmitte zeigen muß. Wenn Sie kein Füßchen des Eproms verbogen haben, ist der größte Teil der Arbeit schon getan.

8. Achten Sie darauf, daß die Schutzfolie richtig zwischen der Speichererweiterung und der Grundplatine liegt. Dann können Sie das Gerät jetzt wieder zuschrauben.

9. Nachdem Sie Ihre komplette Peripherie wieder angeschlossen und das System wieder eingeschaltet haben, muß die Einschaltmeldung des BOS 2.0 erscheinen. Diese sieht so aus (bei der SP512):

BOS 2.0 SP512 (C)1986 vortex GmbH

Ihr BOS 2.0 ist damit betriebsbereit und Sie können mit der Erklärung des neuen Betriebssystems weitermachen. Sollte die Einschaltmeldung nicht kommen, so überprüfen Sie bitte, ob Sie den Eprom richtig gesteckt haben und ob der Clip der Speichererweiterung richtig angebracht ist (siehe Einbauanleitung der Speichererweiterung).

Bei der Befehlsbeschreibung sind zwei Dinge zu beachten:

- alle Befehle werden in einem Kästchen vor ihrer eigentlichen Beschreibung kurz in Ihrer Syntax und Funktion definiert. Bei der Syntax-Definition kann es nun sein, daß ein oder mehrere Parameter mit zwei Schrägstrichen ("/Parameter/") eingerahmt sind. Solch ein Parameter muß nicht zwingend übergeben werden, darf aber vom Benutzer optional benutzt werden.
- wird als Parameter eine Zeichenkette verlangt, so kann dies eine direkte Eingabe zwischen zwei Anführungszeichen oder aber eine alphanumerische Variable sein. Wenn Sie in einem solchen Fall Sie die Zeichenkette in einer alphanumerischen Variablen stehen haben, muß die Adresse dieses Strings übergeben werden. Das sieht dann beispielsweise so aus:

```
a$="test" <ENTER>
!era,@a$ <ENTER>
```

Dieselbe Funktion hat dann, wie gerade beschrieben, der Befehl

```
!era,"test" <ENTER>
```

DIE RAMDISK UNTER BASIC

Wenn Sie eine Diskettenstation besitzen, werden Sie als Benutzter der vortex Speichererweiterung die Ramdisk unter CP/M sicherlich kennen. Sie steht Ihnen ab sofort auch unter Basic zur Verfügung. Sämtliche RSX-Befehle von VDOS und AMSDOS wirken nun auch auf dieses Laufwerk.

Sie müssen dabei allerdings beachten, daß diese erst ab der SP128 sofort verfügbar ist. Dies liegt an der Kompatibilität zu der Ramdisk unter CP/M. Daher haben Sie bei einer dieser Erweiterungen beim Einschalten eine Ramdisk mit derselben Größe wie unter CP/M. Wie Sie die Ramdisk vergrößern können wird bei der Erklärung des Befehls !BOS beschrieben.

Wenn Sie eine SP64 besitzen, so geben Sie jetzt bitte folgenden Befehl ein:

```
!bos,0,0,2 <ENTER>
```

Damit steht auch dem SP64-Besitzer eine Ramdisk zur Verfügung. Die Befehlssequenz, die Sie in diesem Falle gerade eingegeben haben, wird später erklärt werden.

Nach dem Einschalten des Systems ist die Ramdisk nicht formatiert, d.h. sie ist nicht gebrauchsfertig. Sie kann unter Basic erst benutzt werden, wenn Sie unter CP/M automatisch beim Booten oder einmal durch das Programm RAMDISK.COM formatiert wurde. Es geht aber auch wesentlich einfacher, nämlich mit dem Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: !FORMAT <ENTER>                                        *
*                                                                    *
*     Funktion: Formatieren der Ramdisk.                             *
*                                                                    *
*     ACHTUNG: SÄMTLICHE DATEN AUF DER RAMDISK GEHEN VERLOREN        *
*                                                                    *
**********************************************************************
```

Geben Sie doch einfach diesen Befehl ein. Es kommt sofort die Rückfrage "Ramdisk formatieren J/N ?", um ein versehentliches Formatieren der Ramdisk (und einen damit verbundenen Datenverlust) zu vermeiden. Wenn Sie N drücken, kehren Sie ohne irgendwelche Folgen ins Basic zurück, drücken Sie allerdings J, so wird die Ramdisk formatiert. Der erfolgreiche Formatiervorgang wird Ihnen mit der Meldung "Ramdisk formatiert" nochmals bestätigt.

Die Ramdisk ist nun also formatiert und gebrauchsfertig. Aber wie benutzt man Sie nun? Ganz einfach: genauso wie Sie bei VDOS oder AMSDOS mit RSX-Befehlen A und B zwischen den beiden Diskettenlaufwerken hin- und herschalten können, gibt es nun einen RSX-Befehl zum Einblenden der Ramdisk als aktives Laufwerk. Dies ist der Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: |MD <ENTER>                                            *
*                                                                    *
*     Funktion: Einblenden der Ramdisk als Default-Laufwerk.         *
*                                                                    *
**********************************************************************
```

Die normalen Basic-Befehle zum Abspeichern, Laden und Katalogisieren, sowie die RSX-Befehle des DOS beziehen sich nun auf die Ramdisk. Aktivieren Sie doch mit |MD <ENTER> Ihre Ramdisk und verlangen Sie mit CAT <ENTER> eine Auflistung aller vorhandener Dateien. Als Ausgabe werden Sie diese Bildschirm-Meldung erhalten:

```
Laufwerk M: User 0

  OK in   0 Files, 444K frei
```

Beachten Sie dabei bitte, daß sich diese und des weiteren als Beispiele aufgeführte Meldungen immer auf eine eingebaute SP512 beziehen. Sollten Sie eine kleinere Erweiterung haben, so ist die im obigen Beispiel aufgeführte freie Kapazität Ihrer Ramdisk natürlich kleiner.

Die Ramdisk ist also das Laufwerk M - der Befehl zum Umschalten heißt allerdings |MD und wieso nicht |M ? Ganz einfach: wir wollten Konflikte mit anderen Erweiterungen vermeiden, denn es gibt schon ROMs, die ebenfalls den Befehl |M haben. Dies sind beispielsweise das VDOS 2.0 der S- und D-Stationen unseres Hauses (Aufruf des Monitosr) und MAXAM von Arnor (Aufruf des Assemblers). MD ist im BOS 2.0 die Abkürzung für "Memory Drive".

Jetzt wissen wir also, daß noch sehr viel Platz auf der Ramdisk frei ist. Daher werden wir als nächstes ein kleines Basic-Programm abspeichern. Geben Sie doch einfach den "Einzeiler"

```
10 print "Dies ist ein Test":goto 10 <ENTER>
```

ein und speichern Sie dieses kleine Programm mit

```
save"test <ENTER>
```

auf der Ramdisk ab. Danach zerstören Sie das Programm mit

```
new <ENTER>
```

im Basic-Speicher. Es dürfte nun nicht mehr vorhanden sein, was Sie mit

list <ENTER>

überprüfen können. Dafür müßte es jetzt doch auf der Ramdisk sein! Also schnell mit

cat <ENTER>

nachschauen und siehe da, es kommt folgende Ausgabe:

```
Laufwerk M: User 0

TEST    .BAS   1K

  2K in   1 Files, 442 K frei
```

Man sollte es nun doch auch einfach wieder in den Basic-Speicher laden können, oder? Richtig - geben Sie einfach nur

load"test <ENTER>
list <ENTER>

ein und das Programm wird Ihnen in seiner vollen Länge von einer Zeile ausgegeben. Selbstverständlich können Sie auch ein Programm von der Ramdisk aus direkt starten. Dazu müsen Sie nur

run"test <ENTER>

eingeben und das kleine Testprogramm wird von der Ramdisk geladen und sofort gestartet.

Wie in Ihrem Benutzerhandbuch zum CPC beschrieben, können Sie genauso einfach und problemlos die BASIC-Befehle OPENIN, OPENOUT, CLOSEIN, CLOSEOUT, PRINT #9 und INPUT #9 verwenden. Das Abspeichern und Laden von Binär- und ASCII-Dateien, sowie das "mergen" von Programmen ist selbstverständlich auch möglich.

Zusätzlich zu diesen Basic-Befehlen gibt es noch eine Vielzahl von RSX-Befehlen, welche die gleiche Syntax der entsprechenden RSX-Befehle aus VDOS oder AMSDOS haben. Diese wollen wir nun der Reihe nach durchgehen.

Der erste Befehl dieser DOS-Befehle ist

```
**********************************************************************
*                                                                    *
*     Aufruf: !DIR /,Befehlsstring/ <ENTER>                          *
*                                                                    *
*     Funktion: Ausgabe des Diskettenkataloges oder Teilen           *
*               davon.                                               *
*                                                                    *
**********************************************************************
```

In seiner einfachsten Form entspricht der DIR-Befehl dem CAT-Befehl. Nach Eingabe von

```
!dir <ENTER>
```

erhalten Sie die von vorher bekannte Auflistung der auf der Ramdisk vorhandenen Dateien. Sie können im Gegensatz zum CAT-Befehl bei diesem Befehl die Katalog-Ausgabe noch spezifizieren. Geben Sie beispielsweise

```
!dir,"*.bas" <ENTER>
```

ein, so werden Ihnen nur die Dateien mit der Extension BAS, d.h. alle Basic-Programme aufgelistet. Das Zeichen * im obigen Beispiel hat einen speziellen Namen: WILDCARD. Außer dem * gibt es noch ein weiteres Wildcard-Zeichen, das Fragezeichen. Würde man statt dem Stern im obigen Beispiel ein Fragezeichen einfügen, das heißt

```
!dir,"?.bas" <ENTER>
```

eingeben, so würden alle Dateien ausgegeben werden, deren Namen nur ein Zeichen lang sind und welche die Extension BAS haben.

In der bislang von uns verwendeten Form greift der DIR-Befehl immer auf das Default-Laufwerk zu, und dies ist ja zur Zeit die Ramdisk. Sie können allerdings auch von einem anderen Laufwerk das Inhaltsverzeichnis ausgeben lassen, ohne zuvor auf dieses umzuschalten. Das geht mit der Befehlssequenz

```
!dir,"a:test.*" <ENTER>
```

die Ihnen alle Dateien mit dem Namen TEST vom Laufwerk A auflisten würde.

Was könnte man nun mit der immer noch vorhandenen Datei TEST.BAS noch machen? Wir könnten uns zum Beispiel sagen, daß uns der Name der Datei nicht mehr gefällt. Also wollen wir die Datei umbenennen. Mit dem nächsten Befehl ist die ganz einfach.

```
**********************************************************************
*                                                                    *
*     Aufruf: !REN,"neuer Name","alter Name"                         *
*                                                                    *
*     Funktion: die Datei "alter Name" wird in "neuer Name"          *
*               umbenannt.                                           *
*                                                                    *
**********************************************************************
```

Wir wollen der Datei jetzt einfach den Namen NEUTEST.BAS zuweisen. Dazu muß der REN-Befehl wie folgt angewendet werden:

```
!ren,"neutest.bas","test.bas" <ENTER>
```

Wenn wir uns jetzt das Inhaltsverzeichnis der Ramdisk mit dem Befehl CAT anschauen, so gibt es immer noch eine Datei, die jetzt aber NEUTEST.BAS heißt. Es ist unsere ehemalige Datei TEST.BAS - oder glauben Sie es nicht? Dann laden Sie die Datei mit

```
load"neutest <ENTER>
```

in den Basic-Speicher und schauen Sie es sich mit LIST an.

So, nun wollen Sie die von uns angelegte Datei NEUTEST.BAS verwerfen, da diese sowieso nur ein Beispiel ist und nichts anderes bringt, als wertvollen Speicherplatz auf der Ramdisk zu verschlingen. Aber wie löscht man diese Datei nur ?!?

Keine Angst, es geht - denn es gibt den Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: !ERA,"Befehlsstring" <ENTER>                           *
*                                                                    *
*     Funktion: Löschen einer oder mehrerer Dateien.                 *
*                                                                    *
**********************************************************************
```

Wollen wir jetzt die Beispiel-Datei löschen, so müssen wir nur

```
!era,"neutest.bas" <ENTER>
```

eingeben und die Datei ist nicht mehr vorhanden, was Sie mit einem Aufruf des CAT-Befehls überprüfen können. Der ERA-Befehl arbeitet genauso wie der DIR-Befehl mit Wildcards, d.h. Sie können beispielsweise

```
!era,"*.*" <ENTER>
```

eingeben, was zur Folge hätte, daß alle Dateien auf dem derzeit aktiven Laufwerk gelöscht werden. Gerade !era,"*.* kann aber auch fatale Folgen haben, wenn Sie ihn versehentlich anwenden. Gehen Sie daher vorsichtig mit diesem Befehl um!

Bei der Ausgabe des Inhaltsverzeichnisses mit dem Befehl CAT oder DIR ist Ihnen sicherlich die Zeile "Laufwerk M: User 0" aufgefallen. Es gibt im DOS also zwei Unterteilungen, "Laufwerk" und "User", die man anwählen kann. Wie weiter oben besprochen, läßt sich das Laufwerk mit den Befehlen |A, |B und |MD umschalten. Es gibt allerdings noch eine weitere Möglichkeit:

```
**********************************************************************
*                                                                    *
*     Aufruf: |DRIVE,"Laufwerkskennung" <ENTER>                      *
*                                                                    *
*     Funktion: programmierbares Wechseln des aktiven Lauf-          *
*               werkes.                                              *
*                                                                    *
**********************************************************************
```

Durch Eingabe von

```
|drive,"a" <ENTER>
```

können wir einfach auf das Laufwerk A umschalten. Da statt des direkt eingegebenen Strings auch eine Stringvariable übergeben werden kann, ist es ein Einfaches das Laufwerk in einem Programm frei zu wechseln. Solch ein kleines Beispielprogramm wäre:

```
10 input "welches Laufwerk ",l$
20 |drive,@l$
30 cat
40 goto 10
```

Es frägt Sie immer nach einer Laufwerksnummer, wählt diese dann mit |DRIVE an und zeigt Ihnen das Inhaltsverzeichnis dieses Laufwerkes.

Jetzt ist noch der Begriff USER zu erklären. Aber das tun wir am besten mit dem dazugehörigen Befehl:

```
**********************************************************************
*                                                                    *
*     Aufruf: |USER,Integer-Wert <ENTER>                             *
*                                                                    *
*     Funktion: Anwählen eines Benutzerbereiches.                    *
*                                                                    *
**********************************************************************
```

Jedes Laufwerk läßt sich unter BOS 2.0, VDOS und AMSDOS in 16 Benutzerbereiche - die sogenannten Userbereiche - unterteilen. Dies macht vor allem dann einen Sinn, wenn mehrere Anwender die gleiche Diskette benutzen, oder wenn Sie als einzelner Anwender in verschiedenen Bereichen verschiedene Anwendungsprogramme haben wollen. Wenn Sie beispielsweise im Userbereich 4 alle CP/M-Programme abgespeichert haben, so werden diese nur dann beim Auflisten des Inhaltsverzeichnises angezeigt, wenn Sie zuvor den Userbereich 4 angewählt haben. Dies geschieht einfach mit

```
|user,4 <ENTER>
```

Ein nachfolgender CAT-Befehl würde so aussehen:

```
Laufwerk M: User 4

  OK in   0 Files, 442K frei
```

Als Zusammenfassung der beiden Befehle DRIVE und USER, die im AMSDOS verwendet werden, gibt es noch den Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: |SELECT,Befehlsstring                                  *
*                                                                    *
*     Funktion: Laufwerk und/oder Userbereich anwählen               *
*                                                                    *
**********************************************************************
```

Mit ihm können Sie auf einfache Art Benutzerbereiche und Laufwerke anwählen. Dazu zwei Beispiele:

```
|select,"6" <ENTER>
```

schaltet auf den Userbereich 6 des derzeitigen Laufwerkes um. Sollten Sie gleichzeitig noch ein anderes Laufwerk anwählen wollen, so sähe die Befehlssequenz so aus:

```
|select,"B6" <ENTER>
```

Damit hätten Sie gleichzeitig auf das Laufwerk B umgeschaltet und den Userbereich 6 angewählt.

Jetzt gibt es noch zwei speziellere Befehle, die folgende Bedeutung haben:

```
**********************************************************************
*                                                                    *
*     Aufruf: |RESET <ENTER>                                         *
*                                                                    *
*     Funktion: Schließen aller derzeit offenen Dateien.             *
*                                                                    *
**********************************************************************
```

Wollen Sie sicher sein, daß alle Dateien geschlossen sind, so brauchen Sie nur den Befehl

```
|reset <ENTER>
```

eingeben. Er schließt Ihnen automatisch alle eventuell noch offenen Dateien. Dies ist vor allem sinnvoll, wenn man bei einem Diskettenlaufwerk Disketten wechseln möchte, ohne versehentlich Dateien offen zu lassen.

```
**********************************************************************
*                                                                    *
*     Aufruf: !ATTRIBUT,Dateiname,Dateiattribut <ENTER>              *
*                                                                    *
*     Funktion: Einer Datei bestimmte Zustände zuordnen.             *
*                                                                    *
**********************************************************************
```

Mit dem ATTRIBUT-Befehl haben Sie die Möglichkeit, einer Datei bestimmte Merkmale zuzuordnen. Dabei gibt es vier verschiedene Merkmale. Diese sind:

S - Die Datei ist eine sogenannte "SYS"- oder "System"-Datei. Sie wird beim Auflisten des Inhaltsverzeichnisses mittels der Befehle CAT und !DIR nicht mehr angezeigt, sie ist also "unsichtbar".

D - Die Datei ist eine sogenannte "DIR"- oder "Directory"-Datei. D.h. sie wird bei der Auflistung des Inhaltsverzeichnisses angezeigt. Dies ist der Normalzustand nach dem Erstellen einer Datei.

W - Die Datei ist R/W (read/write). D.h. schreib/lesbar. Sie kann auch gelöscht werden. Dies ist ebenfalls der Default-Zustand nach dem erstmaligen Erstellen einer Datei.

R - Die Datei ist R/O (read only). D.h. nur lesbar und damit schreibgeschützt. Sie kann also auch nicht gelöscht oder umbenannt werden.

Als Beispiel wollen wir eine Datei schreibschützen. Dazu müssen wir allerdings zuerst wieder eine Datei auf der Ramdisk haben. Also geben Sie folgende Befehlssequenz ein:

```
10 rem test <ENTER>
!select,"m0" <ENTER>
save"test <ENTER>
cat <ENTER>

Lauwerk M: User 0

TEST    .BAS   1K

  2K in   1 Files, 442K frei
```

Auf der Ramdisk befindet sich nun wieder ein kleines Basicprogramm, das sich wie weiter oben beschrieben wieder laden, umbennen und löschen läßt. Das Schreibschützen der Datei geht nun so:

```
!attribut,"test.bas","r" <ENTER>
```

Listen Sie das Inhaltsverzeichnis mit

```
cat <ENTER>
```

auf, so erhalten Sie diese Ausgabe:

```
Laufwerk M: User 0

TEST     .BAS     1K#

  2K in    1 Files, 442K frei
```

Wie Ihnen sicherlich sofort auffällt, wurde hinter der Ausgabe der Dateilänge ein "#" ausgegeben. Dies bedeutet, daß diese Datei schreibgeschützt (R/O) ist. Ein einfaches (oder versehentliches) Löschen dieser Datei ist nun nicht mehr möglich. Wenn Sie es mit

```
!era,"test.bas <ENTER>
```

versuchen, so meldet Ihnen das Betriebssystem

```
TEST     .BAS nur lesbar
```

und löscht die Datei n i c h t, was Sie mit einem erneuten Auflisten des Inhaltsverzeichnisses überprüfen können. Erst nach Eingabe von

```
!attribut,"test.bas","w" <ENTER>
```

können Sie die Datei wieder löschen.

RELATIVE UND SEQUENTIELLE DATEIVERWALTUNG

Bislang hatten wir nur immer über Dateien gesprochen, deren Inhalt ein Programm darstellte (Programm-Dateien).
Stellen Sie sich vor, Sie möchten ein kleines Adressverwaltungsprogramm in BASIC schreiben mit dem Sie die Adressen (Name, Anschrift usw.) Ihrer Bekannten bearbeiten können, d.h. eingeben, ändern und suchen können. Das setzt aber voraus, daß Sie diese "Daten" permanent speichern können (auf Diskette oder Cassette). Sie müssen Ihr Programm also derart konstruieren, daß es eine Adress-Datei anlegt, in der alle Adressen stehen.
Diese Datei wäre also eine reine Daten-Datei, ihr Inhalt wäre kein ausführbares Programm.
Sie haben jetzt zwei sogenannte "Kanäle" (Ein- und Ausgabekanal) über die Sie Daten zum Rekorder, oder auch zum eventuell vorhandenen Laufwerk schicken und von dort auch empfangen können (die Englischen Fachbegriffe sind hierfür INPUT-Stream und OUTPUT-Stream).
D.h. Sie können gleichzeitig immer nur eine Datei zum Schreiben und eine Datei zum Lesen "geöffnet" haben.
Bevor Sie auf eine Datei "zugreifen" können, muß diese erst "geöffnet" werden. Wenn Sie die Datei im Moment nicht mehr brauchen, muß sie wieder geschlossen werden.
Es geht also nicht, daß Sie gleichzeitig zwei Dateien zum Schreiben öffnen, oder gleichzeitig zwei Dateien zum Lesen öffnen. Dies ist natürlich eine gewisse Einschränkung, mit der man aber leben kann.
Ein weitaus größerer Nachteil ist es, daß der Inhalt dieser Dateien nur nacheinander gelesen und auch nur nacheinander geschrieben werden kann. Dieses "Nacheinander" kennzeichnet die Art, wie diese Dateien vom Betriebssystem her "verwaltet" werden. Man nennt diese Dateiverwaltung deshalb auch "Sequentielle Dateiverwaltung".
Wenn Sie z.B. 1000 Adressen in Ihrer Adresskartei haben und die letzte lesen wollen, dann müssen Sie sich erst durch die 999 Adressen davor durchlesen. Das dauert und bringt wenig Effizienz in die ganze Sache.
Das Vorhandensein der Sequentiellen Dateiverwaltung ist historisch bedingt. Der erste Schneider (der CPC464) war nur mit einem Cassettenrecorder ausgerüstet. Es ist eigentlich klar, daß eine Dateiverwaltung hier nur sequentiell sein kann, denn das Speichermedium ist hier ein Band. Will man von X nach Y, dann muß das ganze Band dazwischen durchlaufen werden.
Dann kamen die 3" Laufwerke. Man hat hier praktisch am Betriebssystem wenig geändert, es wurde "einfach" der Rekorder durch das Laufwerk ersetzt. Es blieb bei der Sequentiellen Dateiverwaltung, obwohl mit einem Disketten-Laufwerk wesentlich mehr gemacht werden kann.
Nachfolgend finden Sie ein kleines Beispiel für die Sequentielle Dateiverwaltung.

Typische BASIC Befehle, die im Zusammenhang mit dem Lesen einer sequentiellen Datei auftauchen, sind:

OPENIN EOF INPUT#9 LINE INPUT#9 CLOSEIN

Typische BASIC Befehle die im Zusammenhang mit dem Schreiben einer sequentiellen Datei auftauchen sind:

OPENOUT PRINT#9 WRITE#9 CLOSEOUT

```
10 OPENOUT "ZAHLEN.DAT"
20 FOR A=1 TO 100
30 A$=STR$(A)
40 PRINT#9,A$
50 NEXT
60 CLOSEOUT
70 OPENIN "ZAHLEN.DAT"
80 INPUT "Welche Zahl suchen Sie ",ZAHL$
90 INPUT#9,VAR$
100 IF ZAHL$=VAR$ THEN GOTO 130
110 Z=Z+1
120 GOTO 90
130 PRINT "Die Zahl steht an der ";Z+1;" ten Position in der Datei"
140 CLOSEIN
150 END
```

Mit Zeile 10 wird die Datei ZAHLEN.DAT zum Schreiben geöffnet, war Sie bislang noch nicht vorhanden, dann wird Sie jetzt angelegt. Innerhalb der FOR/NEXT Schleife werden die Zahlen 1 bis 100 - umgewandelt in Strings - mit PRINT#9,A$ in diese Datei geschrieben. Zeile 60 schließt die Datei ordnungsgemäß.
In 70 wird die Datei wieder zum Lesen geöffnet. Wir suchen eine bestimmte Zahl, bzw. deren Position in der Datei ZAHLEN.DAT. Zeile 100 zeigt einen für sequentielle Dateien typischen Vergleich: man muß sich durch die Datei "durchvergleichen", bis man den Datensatz gefunden hat, den man eigentlich gesucht hat.

So, nun haben Sie allerdings das BOS 2.0 und verfügen damit auf der Ramdisk über VDOS-Fähigkeiten !
Es beherrscht die gerade beschriebene Sequentielle Dateiverwaltung natürlich auch. Es kann aber noch wesentlich mehr.
Es ermöglicht Ihnen eine "Relative Dateiverwaltung".
Eine Relative Dateiverwaltung zeichnet aus, daß ein Datensatz an jeder beliebigen Stelle der Datei herausgeholt, oder hineingeschrieben werden kann, ohne jedoch vorher alle Datensätze davor gelesen zu haben.
Es kann hier auf jeden Datensatz per Datensatznummer wahlfrei zugegriffen werden, das spart natürlich Zeit und Programmieraufwand.
Außerdem erlaubt Ihnen BOS 2.0 neben den 2 sequentiellen Kanälen noch bis zu 16 (!) relative Kanäle gleichzeitig zu öffnen. Damit dürfte sich jedes gestellte Problem optimal in ein Programm verwandeln lassen.
Relative Dateien bestehen wie die sequentiellen Dateien aus Datensätzen (auch Rekords genannt). Im Unterschied zu einer sequentiellen Datei, sind aber bei der relativen Datei diese Rekords vollkommen unabhängig voneinander.

Jeder Rekord hat eine Nummer über die er selektiert werden kann. Rekords können in beliebiger Reihenfolge gelesen bzw. geschrieben werden. Die Länge eines Datensatzes kann innerhalb gewisser Grenzen frei gewählt werden.
Es wird für jede geöffnete relative Datei im RAM-Speicher ein bestimmter Bereich reserviert, der sogenannte "Rekordpuffer". Wird ein Datensatz gelesen bzw. geschrieben, dann gelangt der Datensatz immer über diesen Puffer aus der Datei bzw. in die Datei. Es ist klar, daß die Größe dieses Puffers von der einmal gewählten Rekordlänge abhängt.
Es ist zu beachten, daß bei einer geöffneten relativen Datei die Datensatzlänge nicht änderbar ist.
Ein weiterer Unterschied zwischen beiden "Verwaltungsarten" besteht darin, daß bei einer relativen Datei die Datensätze (Rekords) noch strukturiert werden können. Man sagt auch der Rekord kann noch in verschiedene "Felder" unterteilt werden.
Man kann also sagen die Relative Datei besteht aus einer bestimmten Anzahl von Datensätzen, wobei alle Datensätze noch in eine bestimmte Anzahl von Feldern unterteilt sind.
Die Feldaufteilung in einer relativen Datei muß bei allen Datensätzen dieselbe sein.
Jede geöffnete relative Datei benötigt für ihren Rekordpuffer, für Ihren FCB (File Control Block, enthält auch die Dateibezeichnung der relativen Datei) und etliche interne Zeiger und Zähler Speicher im RAM (alles in allem 90 Bytes + Rekordpuffer pro Kanal).
Dieser Speicher wird bei der Definition des Kanals (siehe |FILES Befehl) im oberen Speicherbereich des CPC's durch dynamisches Herabsetzen des HIMEM's reserviert. Wurde ein Kanal definiert, so kann eine relative Datei auf ihm eröffnet werden (siehe |OPEN Befehl). Es ist zu beachten, daß sowohl bei der Kanaldefinition (|FILES), als auch beim öffnen einer Datei auf diesem Kanal (|OPEN), Rekordlängen angegeben werden müssen.
Dies hat folgenden leicht einsehbaren Grund:
Es ist durchaus möglich auf einem Kanal hintereinander mehrere Relative Dateien zu öffnen und zu schließen, ohne daß jedesmal eine Kanaldefinition (|FILES) erfolgen muß. Man gibt nur bei der einmaligen Kanaldefinition die größte zu erwartende Rekordlänge an und öffnet dann später die einzelnen Dateien (|OPEN) mit ihren jeweils individuellen Rekordlängen.

Nachdem eine relative Datei geöffnet wurde (|OPEN), muß der Rekordpuffer vor dem ersten Zugriff auf die Datei (|PUT, |GET) strukturiert werden, also in Felder unterteilt werden (|FIELD), d.h. man muß angeben, welche Teile des Rekords beim Zugriff in Variablen eingelesen werden sollen.
Jetzt ist die relative Datei für Datenübertragung bereit und man kann bis zum Schließen der Datei (|CLOSE) wahlfrei (man sagt oft auch zufällig, daher kommt auch die manchmal gefundene Englische Bezeichnung "Random File") Rekords lesen und schreiben.
Es ist zu beachten, daß ausschließlich Stringvariablen in Rekords geschrieben und auch wieder ausgelesen werden können.
Will man numerische Variablen oder Felder (auch String-Felder) in Relativen Dateien speichern, so muß man diese zuerst in Stringvariablen umwandeln, bzw. einer Stringvariablen zuweisen

und diese dann abspeichern (siehe BASIC Befehl STR$ in Ihrem CPC BASIC Handbuch).
Will man z.B. wieder einen numerischen Wert "rückgewinnen", so muß er nach dem Auslesen aus der Datei, mittels des BASIC Befehls VAL wieder umgewandelt werden.

Ein Rekord kann, nachdem die Datei geöffnet wurde, durchaus mehrmals neu segmentiert (!FIELD) werden, ohne daß die Datei jedesmal geschlossen und wieder neu geöffnet werden müßte.

Doch nun zu den Befehlen, die den Umgang mit den Relativen Dateien ermöglichen.

```
**********************************************************************
*                                                                    *
*     Aufruf: !FILES,Kanalnummer,Pufferlänge /,.../ <ENTER>          *
*                                                                    *
*     Funktion: Reservieren von Arbeitsspeicher für eine oder        *
*               mehrere relative Dateien.                            *
*                                                                    *
**********************************************************************
```

Bevor mit einer Relativen Datei gearbeitet werden kann, muß hierzu ein Kanal und ein Rekordpuffer definiert werden.
Das öffnen einer Relativen Datei ohne vorhergehendes Reservieren eines Puffers für diese Kanalnummer mit dem !FILES Befehl ergibt die Fehlermeldung:

Feld nicht definiert

Aus diesem Grunde sollte (muß aber nicht !) der !FILES Befehl am Anfang eines BASIC Programmes stehen und für alle später verwendeten Kanalnummern ein ausreichender Rekordpuffer reserviert werden.
Der !FILES Befehl arbeitet mit Zahlen-Doubletten, d.h. zu jeder angegebenen Kanalnummer muß eine maximale Rekordlänge mitangegeben werden. Die Kanalnummer darf Werte von 0 bis 127 annehmen (es dürfen insgesamt 16 Kanäle gleichzeitig geöffnet sein), die maximale Rekordlänge kann beliebig gewählt werden und ist nur durch den zur Verfügung stehenden Arbeitsspeicher begrenzt.
Jeder weitere !FILES macht den zuvor erfolgten hinfällig. Wird nur FILES<ENTER> eingegeben, so werden alle Arbeitsbereiche von Relativen Dateien im RAM gelöscht.

Durch die Eingabe von:

```
!FILES,1,128,2,64,12,20 <ENTER>
```

werden für Kanal 1 ein 128 Bytes großer Rekordpuffer, für Kanal 2 ein 64 Bytes großer Rekordpuffer und für Kanal 12 ein 20 Bytes großer Rekordpuffer reserviert. Es können jetzt auf diesen 3 Kanälen Relative Dateien eröffnet werden.

```
**********************************************************************
*                                                                    *
*     Aufruf: !OPEN,"Dateiname",Rekordlänge,Kanalnummer <ENTER>      *
*                                                                    *
*     Funktion: öffnen einer Relativen Datei.                        *
*                                                                    *
**********************************************************************
```

Mit dem !FILES Befehl haben wir die Voraussetzung dafür geschaffen, daß eine Relative Datei überhaupt benutzt werden kann.
Der nächste Schritt ist, daß die relative Datei "geöffnet" wird, damit wir Daten in sie schreiben, oder Daten aus ihr lesen können.

Dazu dient der !OPEN Befehl.

```
!FILES,3,300<ENTER>

!OPEN,"DATEI.REL",145,3<ENTER>
```

Schauen wir uns den Befehl !OPEN einmal genauer an. Offensichtlich wird hier eine Relative Datei namens DATEI.REL geöffnet. Ist sie bereits auf der Diskette vorhanden, dann wird sie einfach geöffnet, existiert sie noch gar nicht, dann wird sie ersteinmal angelegt und dann geöffnet.
"145" ist die aktuelle Rekordlänge, die kleiner gleich der maximalen Rekordlänge (im !FILES auf 300, eingestellt) sein muß.
Geöffnet wird die Datei auf Kanal 3, auf dem zu diesem Zeitpunkt keine weitere Datei geöffnet sein darf.
Noch einmal: auf ein und demselben Kanal darf gleichzeitig nur eine Datei geöffnet sein. Brauchen Sie aber weitere Dateien, dann wählen Sie einfach eine andere Kanalnummer.
Wenn Sie eine Datei öffnen, die bereits offen ist, dann erhalten wir folgende Meldung:

```
Kanal bereits geoeffnet
```

Vergessen Sie einmal den !OPEN Befehl und benutzen Sie dann trotzdem den !PUT oder den !GET Befehl, dann erhalten Sie folgende Meldung:

```
Kanal nicht definiert
```

```
**********************************************************************
*                                                                    *
*     Aufruf: !FIELD,Kanalnummer,Länge1 /,Länge2 /, ... //           *
*             <ENTER>                                                *
*                                                                    *
*     Funktion: Strukturieren des Rekordpuffers.                     *
*                                                                    *
**********************************************************************
```

Nachdem eine Relative Datei geöffnet wurde, muß deren Rekordpuffer segmentiert werden, d.h. man muß angeben, wieviele Variablen mit welcher Länge in einem Rekord der Datei stehen sollen.
Man will in einem Datensatz folgende drei Variablen haben: NAME$ mit einer maximalen Länge von 12 Bytes, ORT$ mit einer maximalen Länge von 20 Bytes und TEL$ mit einer maximalen Länge von 15 Bytes. Der !FIELD Befehl für diese Kombination sieht z.B. wie folgt aus:

```
!FIELD,4,12,20,15<ENTER>
```

Die aktuelle Länge des Puffers im !OPEN Befehl müßte also 12+20+15=47 oder größer gewesen sein.
Diese Segmentierung bezieht sich auf den Rekordpuffer des Kanal 4. Im Zusammenhang mit dem !FIELD Befehl können wir folgende Meldungen erhalten:

Feld zu lang
Wir haben eine Feldlänge definiert, die die Länge des Rekordpuffers - festgelegt mit !FILES - überschreitet.

Feld nicht definiert
Wir haben in einem !PUT oder !GET Befehl eine Variable angegeben, für die im !FIELD Befehl kein Platz vorgesehen ist.

Feld nicht segmentiert
Wir haben mit !PUT oder !GET einen Datensatz schreiben oder lesen wollen, obwohl wir ihn noch gar nicht mit dem !FIELD Befehl in einzelne Felder unterteilt (segmentiert) haben.

```
**********************************************************************
*                                                                    *
*     Aufruf: !GET,Kanalnummer, Satznummer, @String1 /,String2       *
*              /, ... // <ENTER>                                     *
*                                                                    *
*     Funktion: Lesen eines Datensatzes.                             *
*                                                                    *
**********************************************************************
```

Mit dem !GET Befehl liest man den Datensatz mit der angegebenen Nummer aus der Datei auf diesem Kanal in die angegebenen Variablen, die vorher definiert sein müssen. Wurden die Variablen nicht definiert, dann erhalten wir die Fehlermeldung:
improper argument

```
10 NAME$="":ORT$="":TEL$=""
20 !FILES,1,100
30 !OPEN,"ADRESS.REL",47,1
40 !FIELD,1,12,20,15
60 !GET,1,174,@NAME$,@ORT$,@TEL$
70 PRINT NAME$,ORT$,TEL$
80 !CLOSE,1
```

Hier werden zuerst die Variablen definiert (Zeile 10) und dann wird der Datensatz mit der Nummer 174 aus der Datei auf Kanal A in die Variablen NAME$, ORT$ und TEL$ eingelesen.
Bitte beachten Sie, daß hier nur die Darstellung mit dem "@" (Klammeraffen) zugelassen ist !. Zuvor müssen ordnungsgemäße !FILES, !OPEN und !FIELD Befehle gestartet worden sein.
Enthält der gelesene Rekord keine Daten, so erhalten wir die Meldung:

```
Rekord enthaelt keine Daten
```

```
**********************************************************************
*                                                                    *
*    Aufruf: !PUT,Kanalnummer,Satznummer,@String1 /,@String2         *
*            /, ... // <ENTER>                                       *
*                                                                    *
*    Funktion: Schreiben eines Datensatzes.                          *
*                                                                    *
**********************************************************************
```

!PUT arbeitet genauso wie der !GET Befehl, nur schreibt er den Inhalt der angegebenen Variabeln in die Datei auf dem angewählten Kanal.

```
10 NAME$="":ORT$="":TEL$=""
20 !FILES,1,100
30 !OPEN,"ADRESS.REL",47,1
40 !FIELD,1,12,20,15
50 INPUT NAME$,ORT$,TEL$
60 !PUT,1,174,@NAME$,@ORT$,@TEL$
70 !CLOSE,1
```

Der !PUT Befehl kann auch ohne den "@" (Klammeraffen) verwendet werden:

```
!PUT,1,15,"Mueller","Koeln","123456"
```

Hier werden direkt, die drei Worte in den Datensatz mit der Nummer 15 in die Datei auf Kanal 1 weggeschrieben.
Es ist natürlich klar - und das gilt auch für den !GET Befehl -, daß die Satz-Nummer auch durch eine Variable ersetzt werden kann. So z.B.

```
10 NAME$="":ORT$="":TEL$=""
20 !FILES,1,100
30 !OPEN,"ADRESS.REL",47,1
40 !FIELD,1,12,20,15
45 INPUT "Geben Sie die Datensatznummer ein",REC
```

```
50 INPUT NAME$,ORT$,TEL$
60 !PUT,1,REC,@NAME$,@ORT$,@TEL$
70 !CLOSE,1
```

Durch die Wahl von REC kann die Datensatznummer bestimmt werden, in die der Inhalt der drei Variablen geschrieben werden kann.
Wenn Sie einmal eine Rekordnummer angeben, die den Rahmen der Datei sprengt, dann erhalten Sie die Meldung:
Rekordnummer zu gross

```
**********************************************************************
*                                                                    *
*     Aufruf: !CLOSE,Kanalnummer <ENTER>                             *
*                                                                    *
*     Funktion: Schließen einer relativen Datei.                     *
*                                                                    *
**********************************************************************
```

Soll die Benutzung einer Relativen Datei ordnungsgemäß ablaufen, dann muß zum Schluß die relative Datei wieder geschlossen werden.
!CLOSE,1<ENTER>
Schließt die Datei, die auf Kanal 1 geöffnet war.
Ist auf diesem Kanal keine Datei vorhanden, dan bekommen Sie die Meldung:
Kanal nicht geoeffnet

Zwei kleine Anwendungen

Hier folgen nun die Listings zweier kleiner Programme, die Ihnen das Arbeiten mit relativen Dateien nochmals verdeutlichen sollen. Beim ersten Beispiel wird die Datei TEST.REL geöffnet und es wird ein selektierbarer Rekord gelesen und angezeigt oder es wird ein Rekord mit eingebbarem Inhalt beschrieben.

```
10 !FILES,1,600
20 !OPEN,"TEST.REL",520,1
30 !FIELD,1,250,250
40 X$="":Y$=""
50 PRINT "(L)esen oder (S)chreiben eines Rekords oder (E)nde"
60 K$=UPPER$(INKEY$):IF K$<>"L" and K$<>"S" THEN GOTO 60
70 INPUT "Rekordnummer",REC
80 IF K$<>"L" THEN GOTO 110
90 !GET,1,REC,@X$,@Y$
100 PRINT X$,Y$:GOTO 50
110 IF K$<>"S" THEN GOTO 160
120 INPUT "1. Variable",X$
130 INPUT "2. Variable",Y$
140 !PUT,1,REC,@X$,@Y$
150 GOTO 50
160 !CLOSE,1:END
```

10: Einrichten eines Puffers von 600 Bytes Länge für den Kanal mit der Nummer 1
20: öffnen der Datei TEST.REL mit der aktuellen Satzlänge von 520 Bytes auf Kanal 1
30: Es wird nur der Inhalt zweier Variablen in den Puffer geschrieben. Die Inhalte beider Variablen dürfen eine maximale Länge von 250 Bytes haben.
40: Definieren der beiden Variablen für den !GET Befehl
50: Abfrage, was getan werden soll
60: Eingabeschleife
70: Eingeben der Datensatz Nummer
90: Hier kommen Sie an, wenn Sie oben "L" eingegeben haben. Der Inhalt des durch REC selektierten Satzes wird in die Variablen X$ und Y$ eingelesen.
100: Ausdrucken des Inhaltes
120: Hier kommen Sie an, wenn Sie oben "S" eingegeben haben. Sie geben den Variablen X$ und Y$ (Zeile 130) einen Wert.
140: Der Inhalt der beiden Variablen wird in den durch REC selektierten Datensatz geschrieben.
150: Rücksprung auf Zeile 50 für erneute Eingabe
160: Hier kommen Sie an, wenn Sie "E" eingegeben haben. Die Datei wird ordnungsgemäß geschlossen.

Sie sollten zumindest ein paar Mal nur die Wahl "S" treffen, damit die Datei TEST.REL überhaupt einmal einen Inhalt bekommt, der dann mit "L" gelesen werden kann.

Im zweiten Beispiel werden nacheinander 1000 Rekords in die Relative Datei ZUFALL.REL geschrieben. Danach wird über die Zufallsfunktion (RND=random) im wahrsten Sinne des Wortes "rein zufällig" auf die Datensätze der Datei zugegriffen.

```
10 !FILES,10,120
20 !OPEN,"ZUFALL.REL",25,10
30 !FIELD,10,25
40 X$=""
50 FOR REC=1 to 1000
60 X$="Rekord Nr. "+STR$(REC)
70 !PUT,10,REC,@X$
80 NEXT REC
90 REC1=INT(1000*RND(1))+1
100 !GET,10,REC1,@X$
110 PRINT X$
120 GOTO 90
```

10: Reservieren eines 120 Bytes großen Puffers für Kanal 10.
20: öffnen der Relativen Datei ZUFALL.REL auf Kanal 10 mit der aktuellen Rekordlänge von 25
30: Segmentierung des Puffers. Es wird nur eine Variable angelegt.
40: Definition der Variablen X$ für den !GET Befehl.
60: Es werden 1000 Rekords: "Rekord Nr. 1", "Rekord Nr. 2", ...,"Rekord Nr. 1000" in die Datei ZUFALL.TXT geschrieben.
90: Erzeugen einer Zufallszahl zwischen 1 und 1000
100: Auslesen des Datensatzes mit der Nummer der Zufallszahl.
110: Anzeigen des Datensatzes
120: erneut eine Zufallszahl erzeugen und Datensatz auslesen.

Mit diesem Beispiel wird sehr gut die hohe Zugriffsgeschwindigkeit auf die Daten der relativen Datei demonstriert.

Das nachfolgende Programm erfüllt dieselben Aufgaben wie das gerade gezeigte Beispiel, nur daß hier Sequentielle Dateien verwendet werden. Geben Sie es einmal ein. Sie werden staunen um wieviel langsamer hier alles läuft. Das zeigt einmal mehr, daß für professionelles Arbeiten das "Werkzeug" einer Relativen Dateiverwaltung unerläßlich ist.

```
10 OPENOUT "ZUFALL.SEQ"
20 FOR REC=1 TO 1000
30 X$="Rekord Nr."+STR$(REC)
40 PRINT #9,X$
50 NEXT
60 CLOSEOUT
70 OPENIN "ZUFALL.SEQ"
80 REC1=INT(1000*RND(1))+1
90 VGL$="Rekord Nr."+STR$(REC1)
95 FOR A=1 TO 1000
100 INPUT#9,X$
110 IF X$=VGL$ THEN GOTO 120
115 NEXT
120 PRINT X$
```

```
125 CLOSEIN
130 GOTO 70
```

In Zeile 10 bis Zeile 60 wird die Datei ZUFALL.SEQ angelegt. Sie sehen, daß die Daten ohne jegliche Nummernkennzeichnung weggeschrieben werden.
In Zeile 110 wird der vom Zufallsgenerator in Zeile 90 erzeugte String mit dem gerade ausgelesenen verglichen. Sind beide identisch, dann wird der String angezeigt (Zeile 120). Sind sie nicht gleich, dann muß weiter verglichen werden, bis vielleicht dann der 1000 ste Satz übereinstimmt (ungünstigster Fall). Stimmt gleich der erste Vergleich (Rekord Nr. 1) überein, dann haben wir den günstigsten Fall.
Bei der relativen Dateiverwaltung gibt es keine günstigen und ungünstigen Fälle, alle sind gleich.
Wie Sie leicht feststellen werden, wird ein und dieselbe Aufgabe von der Relativen Dateiverwaltung des VDOS ca. 10 mal schneller als von der Sequentiellen Dateiverwaltung gelöst.

DAS ERWEITERTE BASIC DES BOS 2.0

Viele von Ihnen kennen wahrscheinlich den Vorgänger von BOS 2.0, das Basic-Betriebssystem BOS 1.0 . Das wesentliche Merkmal dieses erweiterten Basics ist das Umschalten mit !BOS, um die Speichererweiterung unter Basic nutzen zu können. Damit wurde aber eine maximal mögliche Anzahl an 32 KB großen Basic-Programmbänken initialisiert und viele bislang erstellte Basic-Programme liefen nicht mehr, da Sie schlicht und einfach zu lang waren. Dieses Manko wurde im BOS 2.0 behoben. Nach dem Einschalten des Systems befinden Sie sich nämlich schon im erweiterten Basic-Betriebssystem auf der Bank 0. Diese Bank 0 ist allerdings fast noch so groß wie Ihr Original-Basicspeicher, nur der vom BOS-Rom benötigte Speicher (wenige 100 Bytes am unteren und oberen Ende des freien RAM) wurde abgezogen.

Für die weitere Beschreibung der Befehle wollen wir noch die im weiteren häufig gebrauchten Begriffe "Bank" und "Block" erklären. Mit Bank ist eine der physikalischen, 64 KByte großen Speicherbänke gemeint. Von diesen Bänken kann es, je nach Größe der Speichererweiterung, bis zu neun geben. Dabei wird von 0 bis acht nummeriert und die Speicherbank 0 ist der Original-RAM des CPCs, die Bänke 1 bis 8 liegen in der Speichererweiterung.
Jede dieser physikalischen Bänke ist nun noch in zwei Blöcke a 32 KByte unterteilt, wobei im BOS 2.0 mit zwei Einschränkungen jeder Block eine von fünf Funktionen (Video, Basic, Ramdisk, Spooler, Ramdatei) innehaben kann. Die zwei Einschränkungen wären: 1. Die beiden Blöcke des Originalspeichers (Bank 0) können nur Basic-Speicher sein und 2. nur der untere Block der Bänke 1 bis 8 kann als Basic-Programmspeicher benutzt werden. Das heißt, daß es nur 9 Basic-Programmbänke gibt, wobei immer nur die untere Hälfte des Speichers gemapped wird und die obere Hälfte immer der obere 32 KByte-Block des Original-RAM (Bank 0) ist.

Nach dem Einschalten hat das Betriebssystem einen 64 KB großen Druckerspooler installiert (der aber noch ausgeschaltet ist) und hat den Rest des vorhandenen Zusatzspeichers für die Ramdisk reserviert. Mit dem Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: !CONFIG <ENTER>                                        *
*                                                                    *
*     Funktion: Ausgabe der derzeitigen physikalischen               *
*               RAM-Konfiguration.                                   *
*                                                                    *
**********************************************************************
```

wird Ihnen die derzeitige RAM-Konfiguration bildlich dargestellt. Dies würde bei einer SP512 so aussehen:

!config <ENTER>

```
B S R R R R R R
B S R R R R R R
```

Dabei ist die erste Spalte mit den beiden B Ihr Originalspeicher im CPC und die nächsten 8 Spalten sind die physikalischen Bänke der Speichererweiterung. Jede physikalische Speichererweiterungsbank ist wiederum zweigeteilt und jede dieser beiden Hälften kann eine andere Funktion besitzen.

Die Bedeutungen der möglichen Funktionen sind:

B - Basic
D - Ramdatei
R - Ramdisk
S - Druckerspooler
V - Videospeicher

Weitere Statusinformationen erhalten Sie noch mit dem Befehl

```
**********************************************************************
*                                                                    *
*    Aufruf: !ID <ENTER>                                             *
*                                                                    *
*    Funktion: Ausgabe von BOS-Statusmeldungen.                      *
*                                                                    *
**********************************************************************
```

der Ihnen nach dem Aufruf folgende Meldung ausgibt:

```
BOS 2.0 SP512   (C)1986 vortex GmbH

 0     0     64k     448K       64K         OK
Bks   Scr   Basic   Ramdisk   Spooler   Ramdatei
```

Bks gibt Ihnen dabei die maximale Basic-Banknummer an, Scr gibt die Anzahl der abspeicherbaren Bildschirminhalte (siehe VIDEO-Befehlsgruppe), die restlichen Angaben sind die Größen der für Basic, Ramdisk, Spooler und Ramdatei reservierten Rambereiche.

Es gibt allerdings noch eine weitere Möglichkeit Statusinformationen zur Speichererweiterung zu erhalten und dies zudem permanent, d.h. auch beispielsweise beim Ablauf eines Programmes. Dazu müssen Sie nur die Tasten "CTRL" und "?" gleichzeitig drücken. Daraufhin wird Ihnen links unten auf dem Bildschirm ein kleines Menü in inverser Schrift ausgegeben. Die erste sichtbare Meldung bezieht sich auf die derzeit aktive Basic-Bank, in unserem Fall ist das augenblicklich die Bank 0. Es gibt aber noch weitere Statusinformationen, die Sie mit den Cursortasten "Pfeil unten" und "Pfeil oben" erreichen können. D.h., Sie können mit diesen beiden Tasten "blättern". Die nächsten beiden Informationen beziehen sich auf den freien Basic-Speicher in der derzeit

aktiven Basic-Bank und die Programmlänge des Basic-Programmes in dieser Bank. Des weiteren gibt es noch eine Ausgabe zum Symbol After-Befehl. D.h., es wird ausgegeben, ab welchem ASCII-Wert die Zeichen benutzerdefiniert sind.
Dieses Statusmenü können Sie durch Drücken der ESC-Taste verlassen und Sie werden bemerken, daß der Bildschirm an der Stelle der Menüausgabe wieder richtig hergestellt wurde, und ein möglicherweise laufendes Programm weiter abgearbeitet wird.

Wenn Sie einen Drucker an die parallele Schnittstelle Ihres CPCs angeschlossen haben, können Sie sofort und einfach den Gebrauch der zusätzlichen Basic-Befehle anhand des Spoolers austesten.

Dazu dient der Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: |SPOOL.ON <ENTER>                                      *
*                                                                    *
*     Funktion: Einschalten des Drucker-Spoolers.                    *
*                                                                    *
**********************************************************************
```

Aber was ist das, ein "Drucker-Spooler"? Ganz einfach: Normalerweise hat ein Drucker einen Puffer zum Zwischenspeichern von auszugebenden Zeichen. Dieser ist aber meistens nicht sonderlich groß (ca. 2-4 KB) und ist daher sehr schnell voll. Wenn Sie dann ein längeres Basic-Programm ausgeben, ist Ihr Rechner lange Zeit blockiert, da er sehr schnell Daten absenden kann, aber immer auf den langsamen Drucker warten muß. Ein Drucker-Spooler vergrößert nun Ihren Puffer und Sie können bei der derzeit gewählten Größe bis zu 65536 Zeichen absenden, ohne daß Ihr Drucker selektiert oder gar angeschlossen ist. Das Betriebssystem versucht nun laufend ein Zeichen abzusenden und tut dies, wenn Ihr Drucker bereit ist. In dieser Zeit können Sie aber problemlos mit dem Rechner weiterarbeiten - was der Riesen-Vorteil des Drucker-Spoolers ist.
Diesen beschriebenen Vorgang können wir ganz einfach testen. Schalten Sie Ihren Drucker einmal aus, und geben Sie einfach folgendes ein:

```
|spool.on <ENTER>
print #8,"Dies ist ein Test des Spoolers!" <ENTER>
```

Normalerweise würde Ihr Rechner jetzt blockiert sein, da er laufend versucht, den Text an den nicht selektierten Drucker auszugeben. Jetzt kam er aber sofort mit dem READY zurück und Sie können weiterarbeiten - der Text steht im Spooler und im Hintergrund wird laufend versucht diesen Text auszugeben. Wenn Sie jetzt Ihren Drucker einschalten wird er sofort ausgegeben.

Den Drucker-Spooler können Sie mit

```
**********************************************************************
*                                                                    *
*     Aufruf: !SPOOL.OFF <ENTER>                                     *
*                                                                    *
*     Funktion: Ausschalten des Drucker-Spoolers.                    *
*                                                                    *
**********************************************************************
```

jederzeit desaktivieren. Die Ausgabe über den Kanal 8 erfolgt jetzt wieder im Originalzustand. D.h. nur dann, wenn wirklich ein Drucker angeschlossen und empfangsbereit ist.
Sollte im Spooler noch etwas gestanden haben, so wird dies nicht gelöscht. Nur der Befehl !SPOOL.ON initialisiert den Spooler komplett und löscht ihn dabei auch. Wenn Sie einen mit !SPOOL.OFF unterbrochenen Spool-Vorgang weiterlaufen lassen wollen, geht dies mit

```
**********************************************************************
*                                                                    *
*     Aufruf: !SPOOL.CONT <ENTER>                                    *
*                                                                    *
*     Funktion: Drucker-Spooler weiter entleeren.                    *
*                                                                    *
**********************************************************************
```

Ein Aufruf dieses Kommandos leitet die Druckerausgabe wieder über den Spooler, ohne einen möglicherweise noch vorhandenen alten Inhalt zu löschen.

Eine weitere Möglichkeit, sofort etwas mit der Speichererweiterung zu tun, bietet sich mit:

```
**********************************************************************
*                                                                    *
*     Aufruf: !PEEK,Speicherbank,Adresse,@Integervariable            *
*             <ENTER>                                                *
*                                                                    *
*     Funktion: den Inhalt einer Speicherzelle einer physi-          *
*               kalischen Speicherbank in eine Integervari-          *
*               able einlesen.                                       *
*                                                                    *
**********************************************************************
```

Sicherlich kennen Sie den Basic-Befehl PEEK, der Ihnen den Inhalt einer Speicherzelle rückmeldet. Der RSX-Befehl !PEEK gibt Ihnen die zusätzliche Möglichkeit aus allen vorhandenen physikalischen Bänken Speicherzellen auszulesen. Dabei wird der ausgelesene Wert in eine Integervariable eingeschrieben, die auch wirklich existieren muß. Der komplette Aufruf des RSX-Befehls !PEEK sieht dann so aus:

```
defint w <ENTER>                    <-- Definition als Integervariable.
w=0 <ENTER>                         <-- reeles Anlegen der Variablen.
!peek,1,0,@w <ENTER>
print chr$(w) <ENTER>
D
Ready
```

Wir haben in diesem Beispiel den Inhalt der Adresse 0 der Speicherbank 1 ausgelesen. Erinnern Sie sich an die Ausgabe nach dem RSX-Befehl !CONFIG. Die Bank 0 war die Original-Bank im CPC, die Bank 1 war der Drucker-Spooler und die restlichen Bänke waren die Ramdisk. In der von uns angelegten Integervariablen W steht nun also das erste Byte der Spooler-Bank. Da wir zuvor den Satz "Dies ist ein T..." über den Drucker-Spooler ausgegeben haben, erhalten wir als Inhalt der Speicherzelle den ASCII-Wert des Zeichens "D".

Ein Äquivalent zu diesem physikalischen PEEK-Befehl ist der Befehl

```
**********************************************************************
*                                                                    *
*    Aufruf: !LPEEK,Basic-Banknummer,Adresse,@Integer-               *
*            variable <ENTER>                                        *
*                                                                    *
*    Funktion: den Inhalt einer Speicherzelle einer logischen        *
*              Basic-Bank in eine Integervariable einlesen.          *
*                                                                    *
**********************************************************************
```

Er unterscheidet sich von !PEEK insofern, daß nur aus einer zulässigen Basic-Bank "gepeekt" werden kann. In unserem Fall kann also derzeit nur aus der Bank 0 ein Wert ausgelesen werden. Als Beispiel:

```
defint w <ENTER>
w=0 <ENTER>
!LPEEK,0,0,@w <ENTER>
print hex$(w)
1
Ready
```

liest den Inhalt der Speicherzelle 1 der Basic-Bank 0 in die Integervariable W. Wird eine Banknummer größer der maximalen Basic-Bank übergeben, so erfolgt die Meldung "unzulaessige Banknummer".

Das Gegenstück zum PEEK-Befehl des Basic dürfte Ihnen auch bekannt sein - das Einschreiben eines Wertes in eine Speicherzelle mit dem POKE-Befehl. Wie Sie sich denken können, gibt es diesen Befehl nun auch als RSX-Befehl für das "Poken" in die Speicherbänke.

```
**********************************************************************
*                                                                    *
*     Aufruf: !POKE,Speicherbank,Adresse,Wert <ENTER>                *
*                                                                    *
*     Funktion: Einschreiben eines Wertes in die Speicherzelle       *
*               einer physikalischen Speichererweiterungsbank.       *
*                                                                    *
**********************************************************************
```

Als Gegenstück zum !PEEK-Befehl kann man mit diesem Befehl in eine physikalische Speichererweiterungsbank "poken". Anhand eines Beispiels läßt er sich leicht erklären:

```
!poke,1,0,&41 <ENTER>
```

Mit dieser Sequenz haben wir in die Adresse null der Spooler-Bank den Wert 41 hexadezimal eingetragen. Hätten wir weiter oben den Spooler nicht geleert und abgeschaltet, so wäre bei Selektierung des Druckers statt des Wortes "Dies" "Aies" ausgedruckt worden.

Entsprechend den Peek-Befehlen gibt es auch noch den logischen Poke-Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: !LPOKE,Basic-Banknummer,Adresse,Wert <ENTER>           *
*                                                                    *
*     Funktion: einen Wert in eine Speicherzelle einer logi-         *
*               schen Basic-Bank eintragen.                          *
*                                                                    *
**********************************************************************
```

Die Syntax entspricht vollgültig dem !POKE-Befehl, nur wird bei Übergabe einer Banknummer größer der maximalen Basic-Bank die Fehlermeldung "unzulaessige Banknummer" ausgegeben.

Ein ebenfalls bekannter Basic-Befehl ist im BOS als RSX-Befehl für die Speichererweiterung vorhanden. Es handelt sich um den Befehl CALL zum Aufruf eines Maschinenprogrammes.

```
**********************************************************************
*                                                                    *
*     Aufruf: !CALL,Speicherbank,Adresse /,Parameter1                *
*             /,Parameter2 ... /,Parameter30/// <ENTER>              *
*                                                                    *
*     Funktion: Aufruf eines Maschinenprogramms in einer             *
*               beliebigen Speichererweiterungsbank.                 *
*                                                                    *
**********************************************************************
```

Entsprechend dem normalen CALL-Befehl des Basic kann mit dem RSX-Befehl !CALL ein Maschinenprogramm in einer beliebigen physikalischen Speichererweiterungsbank aufgerufen werden. Der einzige Unterschied besteht in der Anzahl der übergebbaren Parameter. Beim normalen CALL sind bis zu 32 optionale Parameter möglich,

beim RSX-Befehl |CALL allerdings nur bis zu 30. Als Beispiel:

|call,7,&5000,45 <ENTER>

ruft das Maschinenprogramm ab Adresse &5000 in der physikalischen Bank 7 auf und übergibt einen Parameter, der den Wert 45 hat. Entsprechend den Peek- und Poke-Befehlen gibt es auch hier das Äquivalent für den Aufruf von Maschinenprogrammen in einer logischen Basic-Bank. Es ist der Befehl

```
*****************************************************************
*                                                               *
*    Aufruf: |LCALL,Basic-Banknummer,Adresse /,Parameter1       *
*            /,Parameter2 ... /,Parameter30/// <ENTER>          *
*                                                               *
*    Funktion: einen Wert in eine Speicherzelle einer logi-     *
*              schen Basic-Bank eintragen.                      *
*                                                               *
*****************************************************************
```

Der logische Call-Befehl |LCALL entspricht in seiner Syntax vollgültig dem |CALL-Befehl. Einzige Ausnahme ist, daß statt der physikalischen Banknummer eine logische Basic-Banknummer übergeben werden muß.

Das waren die Befehle, mit denen Sie die Speichererweiterung direkt ansprechen können, ohne einen bestimmten Modus extra aufgerufen zu haben. Nun folgen die Befehle zur Erweiterung des Basic-Programmspeichers, an deren erster Stelle dieser Befehl steht:

```
*****************************************************************
*                                                               *
*    Aufruf: |BOS,Video-Blöcke /,Basic-Blöcke /,Ramdisk-Blöcke  *
*            /,Spooler-Blöcke /,Ramdatei-Blöcke//// <ENTER>     *
*                                                               *
*    Funktion: die Aufteilung des Ramspeichers neu konfi-       *
*              gurieren.                                        *
*                                                               *
*****************************************************************
```

Wie Sie es vom BOS 1.0 her vielleicht kennen, müssen Sie auch im BOS 2.0 mit einem bestimmten Befehl den erweiterten Programmspeicher initialisieren. Im Unterschied zum BOS 1.0 können Sie jetzt allerdings sagen, wie viele zusätzliche Programm-Bänke Sie noch haben wollen.

!!!
Außerdem geht ein in der Originalbank befindliches Basic-Programm bei der Initialisierung der zusätzlichen Basic-Bänke nicht verloren.
!!!

Die maximale Anzahl der zusätzlichen Basic-Bänke kann sich zwischen 1 (bei der SP64) und 8 (bei der SP512) bewegen. Die

Anzahl bei den anderen Funktionen darf sich zwischen 0 und 16 bewegen (wobei hier 32KB-Blöcke gemeint sind). Geben Sie Zahlen als Parameter ein, die größer als die maximal möglichen sind, so wird diese Maximalzahl initialisiert. Dabei werden in der entsprechenden Reihenfolge auch die Prioritäten gesetzt. D.h. wenn Sie mit Video-, Basic- und Ramdisk-Bänken die maximal vorhandene Bankanzahl schon belegt haben und trotzdem noch Spooler- oder Ramdatei-Bänke angeben, werden diese ignoriert.

Ein paar kleine Beispiele:

|bos,0,4 <ENTER>

initialisiert Ihnen 4 zusätzliche Basic-Bänke und reserviert den verbleibenden Speicher für die Ramdisk.
|bos,0,2,0,100 <ENTER>

initialisiert Ihnen 2 zusätzliche Basic-Bänke und reserviert den Rest der Speichererweiterung als Drucker-Spooler.

|bos,0,0,8,8 <ENTER>

unterteilt bei einer SP512 die Speichererweiterung hälftig in Ramdisk und Druckerspooler.

|bos,0,0,0,0,16 <ENTER>

reserviert die komplette Erweiterung als Ramdatei.

Die entsprechenden Änderungen der logischen RAM-Konfiguration können Sie sich am besten veranschaulichen, wenn Sie sich nach dem |BOS-Befehl mit den RSX-Befehlen |CONFIG und |ID die entsprechenden Meldungen ausgeben lassen.

Wir wollen jetzt aber die zusätzlichen Programmier-RSX-Befehle an einem Beispiel durchgehen, das mit jeder Speichererweiterung funktioniert, sei es nun die SP64 oder die SP512. Setzen Sie deshalb Ihren Rechner mit CTRL-SHIFT-ESC zurück und geben Sie

|bos,0,1 <ENTER>

ein. Wir haben jetzt zwei Basic-Bänke zur Verfügung. Einmal die Bank 0 (Original-Speicher des CPC) und dann noch die Bank 1, die in der Speichererweiterung liegt. Für Programmierer, die es interessiert: Beachten Sie bitte, daß die Basic-Bank 1 nicht zwingend in der physikalischen Speicherbank 1 liegen muß!

Also - wir wissen, daß wir eine weitere Basic-Bank haben, der RSX-Befehl |ID meldet es ja auch:

```
!id <ENTER>

BOS 2.0 SP512   (C)1986 vortex GmbH

 1     0     96K     448K       32K        OK
Bks   Scr   Basic   Ramdisk   Spooler   Ramdatei
```

Aber wie nutzt man diese zweite Bank ?!?

Zuerst muß man die Möglichkeit haben, auf diese Bank zu gelangen. Dazu gibt es den Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: !BANK,Basic-Banknummer <ENTER>                         *
*                                                                    *
*     Funktion: eine andere logische Basic-Bank einblenden.          *
*                                                                    *
**********************************************************************
```

Wenn zusätzliche Basic-Bänke initialisiert wurden, kann man mit diesem Befehl zwischen den einzelnen Basic-Bänken wechseln. Da wir uns auf Bank 0 befinden und eine zusätzliche Basic-Bank initialisiert haben, könnten wir auf die Bank 1 wechseln, was wir jetzt auch tun:

```
!bank,1 <ENTER>
```

Mit dem CTRL-?-Hilfsmenü können Sie sofort nachschauen, ob es auch geklappt hat. Sie werden als erste Meldung "Bank-Nr. 1" erhalten und vielleicht schon wirklich glauben, daß wir wirklich auf einer anderen Basic-Bank sind. Um Ihnen dies ganz klar zu verdeutlichen, wollen wir ein kleines Programm eingeben:

```
10 print"Dies ist die Basic-Bank eins" <ENTER>
20 goto 10 <ENTER>
```

Nun wechseln wir mit

```
!bank,0 <ENTER>
```

wieder in die Bank null. Wenn wir jetzt mit

```
list <ENTER>
```

nachschauen, was im Programmspeicher steht, so wird uns nichts ausgegeben - der Programmspeicher der Bank 0 ist erwartungsgemäß leer. Deshalb schreiben wir auch in die Bank 0 ein kleines Programm:

```
100 print"Die Basic-Bank null ist aktiv!" <ENTER>
110 goto 100 <ENTER>
```

Ein LIST überzeugt uns, daß dieses Programm wirklich im Speicher steht. Wir wechseln einfach in Bank 1 und "listen" dort einmal:

```
!bank,1 <ENTER>
list <ENTER>
10 PRINT"Dies ist die Basic-Bank eins"
20 GOTO 10
```

Wir haben also verifiziert, daß zwei verschiedene Programme im Speicher stehen. Diese laufen auch getrennt, was Sie durch Bankwechsel und Start des Programmes (RUN) überprüfen können.

Wie Sie anhand des Beispieles erkannt haben, ist das Auflisten aller im Speicher vorhandenen Programme mit dem normalen Basic-Befehl LIST etwas umständlich. Deshalb gibt es den RSX-Befehl

```
**********************************************************************
*                                                                    *
*    Aufruf: !LIST /,Banknummer1 ... /,Banknummer9// <ENTER>         *
*                                                                    *
*    Funktion: Programme über mehrere Bänke hinweg auflisten.        *
*                                                                    *
**********************************************************************
```

Wenn Sie den RSX-Befehl !LIST ohne Parameter eingeben, so erscheint bei uns derzeit folgende Ausgabe:

```
----- Bank 0 -----

0-100 PRINT"Die Basic-Bank null ist aktiv!"
0-110 GOTO 100

----- Bank 1 -----

1-10 PRINT"Dies ist die Basic-Bank eins"
1-20 GOTO 10
```

Entsprechend können Sie als Parameter auch Basic-Banknummer angeben, die dann nur aufgelistet werden.

```
!list,1 <ENTER>
```

würde Ihnen beispielsweise nur die Bank 1 auflisten.

Wenn Sie die Ausgabe des !LIST-Befehls umleiten wollen, haben Sie diese Möglichkeit mit dem !DEV-Befehl.

```
**********************************************************************
*                                                                    *
*     Aufruf: !DEV /,Kanalnummer/ <ENTER>                            *
*                                                                    *
*     Funktion: Zuweisung eines Ausgabekanals für den erwei-         *
*               terten !LIST-Befehl.                                 *
*                                                                    *
**********************************************************************
```

Es stehen Ihnen folgende Möglichkeiten der Ausgabeumleitung zur Verfügung:

0 - Bildschirm
8 - Drucker
9 - Diskettenlaufwerk/Recorder.

Rufen Sie den !DEV-Befehl mit

```
!dev,8 <ENTER>
```

auf, so wird bis zum nächsten Aufruf des !DEV-Befehles die komplette Ausgabe des !LIST-Befehles auf den Drucker umgeschaltet. Geben Sie beim Aufruf des !DEV-Befehles keinen Parameter an, wird automatisch Kanal 0, d.h. der Bildschirm gewählt.

So, jetzt wissen wir also, wie man zwischen Bänken umschaltet, in die Bänke eigenständige Programme schreibt und diese auch komplett auflisten kann. Aber wie verkettet man nun die Programme in den einzelnen Bänken? Dazu gibt es wiederum eine Gruppe von RSX-Befehlen.

```
**********************************************************************
*                                                                    *
*     Aufruf: !GOTO,Basic-Banknummer,Zeilennummer <ENTER>            *
*                                                                    *
*     Funktion: Es wird die entsprechende Basic-Bank gewechselt      *
*               und die Programmausführung wird in der überge-       *
*               benen Zeilennummer fortgesetzt.                      *
*                                                                    *
**********************************************************************
```

Wie es Ihnen die kurze Funktionsbeschreibung schon klar gemacht hat, dient dieser Befehl dazu, einen Sprung in ein Programm auszulösen, das in einer anderen Speicherbank liegt.
In unser Beispielprogramm kann man solche Sprünge einfach einbauen, und das wollen wir auch tun. Geben Sie einfach die folgende Befehlssequenz ein.

```
!bank,0 <ENTER>
110 !goto,1,10 <ENTER>
!bank,1 <ENTER>
20 !goto,0,100 <ENTER>
!list <ENTER>
```

```
----- Bank 0 -----

0-100 PRINT"Die Basic-Bank null ist aktiv!"
0-110 |GOTO,1,10

----- Bank 1 -----

1-10 PRINT"Dies ist die Basic-Bank eins"
1-20 |GOTO,0,100
```

Wir haben jetzt aus den zwei separaten Programmen ein zusammenhängendes gemacht. Wenn wir es starten, müßte immer eine Meldung ausgedruckt, die Bank gewechselt und die nächste Meldung ausgegeben werden. Starten Sie es einfach mit

run <ENTER>

und Sie sehen - es klappt!
Um Sie noch etwas zu verblüffen: Es muß keine feste Zeilennummer im erweiterten GOTO-Befehl stehen, sondern man kann auch in eine Zeile Springen, deren Nummer in einer Variablen steht. Durch eine einfache Änderung im Programm läßt sich dies zeigen:

```
|bank,0 <ENTER>
105 zeile=10 <ENTER>
110 |goto,1,zeile <ENTER>
run <ENTER>
```

Wir haben nur eine Integervariable definiert und springen nun in die Bank 1 in die Zeile, die in der Variablen ZEILE steht. Da die Variable den festen Wert 10 zugeordnet bekommen hat, läuft das Programm genauso wie vorher.

!!!
Aber eines muß hier gesagt werden: Wenn Sie in einer Bank die Zeilennummern mit dem Basic-Befehl RENUM neu numerieren, so müssen Sie in den anderen Bänken die RSX-GOTO-Zeilennummern manuell ändern. Das Gleiche gilt auch für den RSX-GOSUB-Befehl, der gleich erklärt wird. Also: Ein wichtiger Punkt, den es immer zu beachten gilt!
!!!

```
*****************************************************************
*                                                               *
*    Aufruf: |GOSUB,Basic-Banknummer,Zeilennummer <ENTER>       *
*                                                               *
*    Funktion: ein Unterprogramm in einer anderen Basic-        *
*              Bank aufrufen.                                   *
*                                                               *
*****************************************************************
```

Ähnlich wie der RSX-Befehl GOTO verzweigt auch dieser in eine Zeile in einer anderen Basic-Bank. Unser Beispielprogramm verändern wir mit der Erklärung des nächsten Befehles.

```
**********************************************************************
*                                                                    *
*     Aufruf: !RETURN <ENTER>                                        *
*                                                                    *
*     Funktion: in die Bank und Zeile zurückkehren, von der aus      *
*               ein Unterprogramm mit !GOSUB aufgerufen wurde.       *
*                                                                    *
**********************************************************************
```

Am besten und einsichtigsten lassen sich die Befehle !GOSUB und !RETURN erklären, wenn wir unser Beispielprogramm ändern. Also geben Sie folgende Befehlssequenz ein:

```
!bank,0 <ENTER>
110 !gosub,1,zeile <ENTER>
120 goto 100
!bank,1 <ENTER>
20 !return <ENTER>
!list

----- Bank 0 -----

0-100 PRINT"Die Basic-Bank null ist aktiv!"
0-105 zeile=10
0-110 !GOSUB,1,10
0-120 GOTO 100

----- Bank 1 -----

1-10 PRINT"Dies ist die Basic-Bank eins"
1-20 !RETURN

!bank,0 <ENTER>
run <ENTER>
```

Das Programm gibt nun die erste Meldung aus, springt in das "Unterprogramm" in Bank 1, und gibt dort die zweite Meldung aus. Dann kehrt es mit dem !RETURN in das "Hauptprogramm" in Bank 0 zurück und springt dort wieder an den Anfang des Programms.

Bevor wir zu einem ganz wesentlichem Punkt, nämlich der Variablenverwaltung kommen, klären wir erst noch den Punkt des Laden und Speichern von Programmen im Bank-Basic.

```
**********************************************************************
*                                                                    *
*     Aufruf: !SAVE,"Dateiname" /,Bank1 /,Bank2 /, ... ///           *
*             <ENTER>                                                *
*                                                                    *
*     Funktion: Abspeichern eines Programmes, das auf mehrere        *
*               Basic-Bänke verteilt ist.                            *
*                                                                    *
**********************************************************************
```

Mit Hilfe dieses Befehles können Sie ein Programm, das über mehrere Basic-Bänke verteilt ist, in einem Arbeitsgang auf Diskette/Cassette sichern. Haben Sie zum Beispiel ein Programm über drei Bänke verteilt im Speicher liegen, so können Sie es mit

```
!save,"test" <ENTER>
```

abspeichern. Danach befinden sich auf der Diskette/Cassette folgende Programme:

```
TEST.BBS  -  das Ladeprogramm.
TEST.BK0  -  der Programmteil aus der ersten Bank.
TEST.BK1  -  der Programmteil aus der zweiten Bank.
TEST.BK2  -  der Programmteil aus der dritten Bank.
```

Dieses Programm können Sie sofort mit dem !LOAD-Befehl wieder in den Speicher laden.

```
**********************************************************************
*                                                                    *
*     Aufruf: !LOAD,"Dateiname" <ENTER>                              *
*                                                                    *
*     Funktion: Laden eines mit dem !SAVE-Befehl abge-               *
*               speicherten Programmes.                              *
*                                                                    *
**********************************************************************
```

Mit dem Befehl

```
!load,"test" <ENTER>
```

wird das zuvor mit !SAVE gespeicherte Programm wieder in die verschiedenen Basic-Bänke geladen.

Wenn Sie ein solches Programm mit !LOAD geladen haben, müssen Sie es noch mit dem Basic-Befehl RUN starten. Um beides in einem Zuge zu erledigen, gibt es noch den RSX-Befehl:

```
**********************************************************************
*                                                                    *
*     Aufruf: !RUN,"Dateiname" /,Basic-Banknummer, Zeilen-           *
*             nummer// <ENTER>                                       *
*                                                                    *
*     Funktion: Laden und Starten eines mit dem !SAVE-Befehl         *
*               abgespeicherten Basic-Programmes.                    *
*                                                                    *
**********************************************************************
```

```
!run,"test" <ENTER>
```

würde das zuvor gespeicherte Programm TEST.BAS in die verschiedenen Basic-Bänke laden und den Programmablauf in der ersten Zeile der momentan aktiven Basic-Bank starten. Geben Sie optional eine Basic-Banknummer und eine Zeilennummer ein, so wird das Programm in der entsprechenden Bank und Zeile gestartet.

```
!run,"test",3,100 <ENTER>
```

lädt beispielsweise das Programm TEST.BAS in die verschiedenen Basic-Bänke, wechselt auf die Basic-Bank Nummer 3 und startet dort das Programm mit der Zeilennummer 100.

So, nun kommen wir zu dem etwas schwierigeren Teil, der Variablenverwaltung.
In unserem Beispielprogramm haben wir bislang nur eine Variable benutzt. Die Integervariable ZEILE in der Programmbank 0 diente uns dazu, den erweiterten !GOTO- und !GOSUB-Befehl mit Variablen zu demonstrieren. Sie bekam in Zeile 0-105 ständig den Wert 100 zugewiesen, aber welchen Wert hat diese Variable in der Programmbank 1 ? Das läßt sich doch leicht feststellen. Geben Sie nur folgende Befehle ein:

```
!bank,1 <ENTER>
15 print zeile <ENTER>
!bank,0 <ENTER>
run <ENTER>
```

Es wird uns nun der Wert der Integer-Variablen ZEILE in der Programmbank 1 ausgegeben. In der Bank 0 hat die Variable den Wert 100, aber in Bank 1 wird nun laufend der Wert 0 ausgegeben ? Des Rätsels Lösung: ohne andere Deklaration ist jede Variable im Bank-Basic eine "lokale" Variable. Das heißt ganz einfach, daß Wertzuweisungen an eine Variable nur in der derzeitig aktiven Basic-Bank vollzogen werden oder noch klarer ausgedrückt: Sie können in jeder Basic-Bank Variablen mit gleichem Namen anlegen, die allerdings völlig unabhängig voneinander verschiedene Werte haben können - ein Riesen-Vorteil für Programmierer, die Strukturen bevorzugen: endlich kann man Unterprogramme mit lokalen Variablen entwickeln!

Aber trotzdem muß es noch die Möglichkeit geben, Variablen als "global", d.h. über mehrere Bänke hinweg gültig zu deklarieren. Dies geht mit

```
*********************************************************************
*                                                                   *
*     Aufruf: !COMMON /,Zeichenkette /,Basic-Banknummer1            *
*             /,Basic-Banknummer2 /, ... //// <ENTER>              *
*                                                                   *
*     Funktion: Variablen als "global" definieren.                  *
*                                                                   *
*********************************************************************
```

Wenn wir in unser Beispiel-Programm folgendes einfügen

```
!bank,0 <ENTER>
10 !common,"z" <ENTER>
```

so sind des weiteren alle Variablen, deren Name mit einem "Z" anfängt, als "global" definiert und auf allen Basic-Bänken gültig. Starten wir das Programm mit RUN, so wird nun als Wert der Variablen ZEILE in der Programmbank 1 ebenfalls 100 ausgegeben. Die Variable Zeile ist also über alle Basic-Bänke hinweg gültig.

Mit der dem !COMMON-Befehl übergebenen Zeichenkette können Sie gezielt Variablengruppen anwählen, die in allen Programmbänken zugänglich sind. Diese Zeichenkette hat folgende Struktur:

```
string$="b1,b2,b3-b4,b5,b6-b7..."
```

Hierbei stehen b1,b2,... für die Anfangsbuchstaben der Variablengruppen. Hier noch zwei Beispiele zu dieser Zeichenkette, um deren Aufbau zu verdeutlichen:

```
string$="A,M,Z"
```

würde alle Variablen, deren Namen mit A, M oder Z anfangen, als "global" definieren.

```
string$="A-E,Y"
```

würde alle Variablen, die mit A, B, C, D, E oder Y anfangen, als "global" definieren.

Wie Sie sicherlich bemerkt haben, deklariert der !COMMON-Befehl mit ausschließlich übergebener Zeichenkette die entsprechenden Variablen als "global" über alle vorhandenen Basic-Bänke hinweg. Sie haben allerdings auch die Möglichkeit, selektiv zu sagen, auf welchen Programmbänken diese Variablen "global" sein sollen. Dies geht einfach durch Anhängen der gewünschten Basic-Banknummern an die Parameterliste. Beispielsweise

```
|common,"A-C",2,3 <ENTER>
```

deklariert die Variablen, die mit A, B oder C beginnen, auf den Basic-Bänken 2 und 3 als "global", während die entsprechenden Variablen auf den verbleibenden Bänken weiterhin "lokal" sind.

Sie können aber auch den Ursprungszustand wieder herstellen:

```
|COMMON <ENTER>
```

definiert alle Variablen wieder als "lokal" auf allen Basic-Bänken.

Zwei Dinge gilt es bei der Variablenverwaltung zu beachten:

- Variablenfelder können nicht als "global" definiert werden. Möchte man dennoch Feldelemente übertragen, so muß man diese zuvor einer STRING-, REAL- oder INTEGER-Variablen des Typs COMMON zuweisen.

- Da beim Umschalten zwischen den Basic-Bänken immer alle als COMMON deklarierten Variablen von Bank zu Bank transferiert werden, hängt die Geschwindigkeit des Umschaltvorganges entscheidend von der Anzahl der COMMON-Variablen ab.

Der nächste zu erklärende Befehl ist in seiner Funktion sicher jedem Programmierer eines CPCs bekannt.

```
**********************************************************************
*                                                                    *
*     Aufruf: |NEW /,Basic-Banknummer1 /,Basic-Banknummer2           *
*             /, ... /// <ENTER>                                     *
*                                                                    *
*     Funktion: Löschen von Basic-Programmbänken.                    *
*                                                                    *
**********************************************************************
```

Mit dem erweiterten |NEW-Befehl können die Inhalte der Basic-Programmbänke gelöscht werden. Optional lassen sich selektiv einzelne Programmbänke löschen.

```
|new,1,3 <ENTER>
```

löscht die Programmbänke 1 und 3, alle anderen Programmbänke bleiben unversehrt.

Nun gibt es noch einen speziellen Betriebsmode des BOS 2.0, den VIDEO-MODE. In ihm lassen sich bis zu 32 komplette Bildschirminhalte in der Speichererweiterung zwischenspeichern und wieder laden. Dieser Betriebsmode wird aufgerufen mit

```
**********************************************************************
*                                                                    *
*    Aufruf: |VIDEO.ON <ENTER>                                       *
*                                                                    *
*    Funktion: Einschalten des Video-Mode.                           *
*                                                                    *
**********************************************************************
```

Nach Aufruf dieses Befehles befinden Sie sich in Bank 0. Der HIMEM-Wert ist auf &3F7F heruntergesetzt. Dies liegt daran, daß die Möglichkeit, den Bildschirmspeicher auf &4000 zu legen, von diesem Betriebsmode genutzt wird. Beim schnellen Laden einer Bilderfolge aus der Speichererweiterung wird ein Bild immer in den nicht sichtbaren Bereich geladen und dann der Bildschirm auf diese Adresse gelegt. Dadurch ist der Ladevorgang nicht sichtbar, und der Eindruck einer bewegten Bildfolge wird noch verstärkt.
Ein Verlassen des Video-Modes ist nur durch ein komplettes Zurücksetzen des Rechners (CTRL/SHIFT/ESC) oder mit dem RSX-Befehl |BASIC möglich.

Die Anzahl der in der Speichererweiterung ablegbaren Bildschirminhalte hängt von der Größe der Erweiterungskarte und Ihrer Anfangsinitialisierung ab. Mit dem Befehl |BOS reservieren Sie sich eine von Ihnen benötigte Blockanzahl für die Speicherung von Bildschirmen. Dies tun Sie mit dem ersten Parameter, der aber immer eine gerade Zahl sein muß (spezielle Eigenschaft der Befehle |FORWARD und |BACKWARD !).

Die maximale Anzahl von speicherbaren Bildschirminhalten liegt zwischen 4 und 32. Die genaue Anzahl können Sie mit dem folgenden Befehl ermitteln:

```
**********************************************************************
*                                                                    *
*    Aufruf: |SCREENS <ENTER>                                        *
*                                                                    *
*    Funktion: Berrechnen der höchst möglichen Bildnummer.           *
*                                                                    *
**********************************************************************
```

Um beim Arbeiten im Video-Mode zu wissen, wieviele Bildschirminhalte in der Speichererweiterung abgelegt werden können, gibt es diesen Befehl. Er legt Ihnen nach dem Aufruf die Systemvariable SCR an, in der die maximale Bildnummer steht. Da die Zählweise im Video-Mode mit 0 beginnt, gilt: SCR+1=Anzahl der möglichen Bilder.

Als Beispiel:

```
|screens <ENTER>
print scr <ENTER>
```

gibt Ihnen die maximale Bildnummer im Video-Mode aus.

Zum Speichern eines Bildschirminhaltes in der Speichererweiterung dient der Befehl

```
**********************************************************************
*                                                                    *
*     Aufruf: !SCREEN.OUT,Bildnummer <ENTER>                         *
*                                                                    *
*     Funktion: Abspeichern eines Bildschirminhaltes in der          *
*               Speichererweiterung.                                 *
*                                                                    *
**********************************************************************
```

Sofern Sie den Video-Mode mit !VIDEO.ON aktiviert haben, können Sie mit !SCREEN.OUT den momentanen Bildschirminhalt in der Speichererweiterung ablegen und später mit den weiter unten erklärten Befehlen wieder sichtbar machen. Sie müssen beim Aufruf des Befehls eine Bildnummer angeben, unter der das Bild abgespeichert werden soll. Diese Zahl darf zwischen 0 und SCR liegen. Beispielsweise legt

```
!screen.out,2 <ENTER>
```

den momentanen Bildschirminhalt unter der Bildnummer 2 in der Speichererweiterung ab.

Das Zurückladen eines abgespeicherten Bildes in den Bildschirmspeicher wird mit diesem Befehl möglich:

```
**********************************************************************
*                                                                    *
*     Aufruf: !SCREEN.IN,Bildnummer <ENTER>                          *
*                                                                    *
*     Funktion: Einladen eines kompletten Bildes aus der             *
*               Speichererweiterung in den Bildschirmspeicher.       *
*                                                                    *
**********************************************************************
```

Beispielsweise lädt

```
!screen.in,2 <ENTER>
```

das unter der Bildnummer 2 in der Speichererweiterung liegende Bild in den Bildschirmspeicher und macht es dadurch wieder sichtbar. Wollen Sie allerdings schnellstmöglich alle in der Speichererweiterung liegenden Bilder zeigen, so empfiehlt es sich, einen der beiden nächsten Befehle zu verwenden.

```
**********************************************************************
*                                                                    *
*     Aufruf: !FORWARD /,Bildnummer/ <ENTER>                         *
*                                                                    *
*     Funktion: Schnelles Sichtbarmachen von Bildern in auf-         *
*               steigender Folge.                                    *
*                                                                    *
**********************************************************************
```

Mit !FORWARD können Sie schnellstmöglich alle Bilder ab der Bildnummer 0 sichtbar machen. Geben Sie allerdings als Parameter eine Bildnummer an, so werden nur die Bilder bis zu dieser Nummer gezeigt. Ein Beispielprogramm finden Sie bei der Erklärung des nächsten Befehles.

```
**********************************************************************
*                                                                    *
*     Aufruf: !BACKWARD /,Bildnummer/ <ENTER>                        *
*                                                                    *
*     Funktion: Schnelles Sichtbarmachen von Bilder in ab-           *
*               steigender Folge.                                    *
*                                                                    *
**********************************************************************
```

Mit !BACKWARD können Sie schnellstmöglich alle Bilder ab der Bildnummer SCR rückwärts sichtbar machen. Geben Sie allerdings als Parameter eine Bildnummer ein, so werden nur die Bilder bis zu dieser Nummer gezeigt. Hier nun ein kleines Demonstrations-Programm für den Video-Mode:

```
5 !bos,16
10 !video.on
20 !screens
30 for i=0 to scr
40 mode 2
50 move 0,10*i:draw 639,399-10*i
60 move 0,399-10*i:draw 639,10*i
70 !screen.out,i
80 next i
90 !forward:!backward::goto 90
```

Tippen Sie es einfach einmal ab und schauen Sie sich die Bewegungseffekte nach dem Aufbau der ganzen Bilder an. Ihrer Fantasie sind dann in dieser Richtung wohl keine Grenzen mehr gesetzt. Statt der zwei einfachen Linien können Sie ja auch wesentlich komplexere Figuren errechnen und dann bewegen lassen!

Um den Komplex der Video-Befehle noch abzurunden, gibt es noch zwei Befehle zum Umschalten des Bildschirmspeichers zwischen der oberen 16K-Bank (ab Adresse &C000) und der unteren 16K-Bank (ab Adresse &4000).

```
**********************************************************************
*                                                                    *
*     Aufruf: |SCR.BASE,Integerwert <ENTER>                          *
*                                                                    *
*     Funktion: Adresse des Bildschirmspeicher setzen.               *
*                                                                    *
**********************************************************************
```

Mit diesem Befehl können Sie die Startadresse des Bildschirms auf eine der vier möglichen Adressen 0000H, 4000H, 8000H oder C000H setzen. Der dabei zu übergebende Wert ist eine Integerzahl zwischen 0 und 3. Rufen Sie diesen Befehl beispielsweise mit

|SCR.BASE,1 <ENTER>

auf, so liegt ab sofort der Start des Bildschirmspeichers ab Adresse 4000H. Beachten Sie dabei, daß zwar alle vier Adressen theoretisch möglich sind, aber im CPC wegen der komplexen Speicheraufteilung nur die Adressen 4000H und C000H sinnvoll sind.

Eine weitere Befehlsgruppe umfaßt die "Ramdatei", welche bei den CPC 464-Besitzern mit Sicherheit bekannt ist. Wegen ihnen wurde diese Befehlsgruppe auch in das BOS 2.0 implementiert. Dadurch laufen Programme, die unter BOS 1.0 erstellt wurden und die Ramdatei benutzen, auch unter BOS 2.0 . Sollten Sie allerdings neue Programme mit Relativer Dateiverwaltung schreiben wollen, so empfiehlt es sich, mit der Ramdisk und den 16 relativen Dateien zu arbeiten, da dieses Konzept wesentlich flexibler als die Ramdatei ist.

```
**********************************************************************
*                                                                    *
*     Aufruf: |RAMOPEN, Rekordlänge <ENTER>                          *
*                                                                    *
*     Funktion: öffnen einer relativen Ramdatei.                     *
*                                                                    *
**********************************************************************
```

Bevor Sie mit einer relativen Ramdatei arbeiten können, muß diese geöffnet, d.h. für den Datenaustausch vorbereitet werden. Mit dem Parameter wird die Datensatzlänge angegeben, die dann mit dem Befehl |RAMFIELD weiter strukturiert wird. Sie darf zwischen 8 und 1048 liegen.

WICHTIG: Vor dem öffnen einer Ramdatei muß mit dem Befehl |BOS natürlich Platz für diese reserviert werden!

Beispiel:

```
|bos,0,0,0,0,16 <ENTER>
|ramopen,512 <ENTER>
```

reserviert die komplette Erweiterung für die Ramdatei und öffnet diese mit einer Datensatzlänge von 512 Bytes.

```
**********************************************************************
*                                                                    *
*     Aufruf: !RAMFIELD,Länge1 /,Länge2 /,.../// <ENTER>             *
*                                                                    *
*     Funktion: Einteilung des Datensatzes in einzelne Felder.       *
*                                                                    *
**********************************************************************
```

Dieser Befehl dient zur Strukturierung der Datensätze der relativen Ramdatei. Er darf erst nach dem !RAMOPEN-Befehl im Programm stehen. Ein Datensatz mit einer Länge von beispielsweise 256 Bytes wird z.B. mit Länge1=30, Länge2=40 und Länge3=20 in drei unterschiedlich lange Felder zerlegt. Die Befehle !RAMREAD und !RAMWRITE nutzen in diesem Fall drei Variablen den drei Feldern des Datensatzes entsprechend. Die Summe aller Feldlängen eines Datensatzes darf logischerweise die gesamte Länge des Datensatzes nicht überschreiten.

Die Befehlssequenz bis zu dem oben als Beispiel aufgeführten !RAMFIELD-Befehl würde so aussehen:

```
!bos,0,0,0,0,16 <ENTER>
!ramopen,256 <ENTER>
!ramfield,30,40,20 <ENTER>
```

```
**********************************************************************
*                                                                    *
*     Aufruf: !RAMWRITE,Rekordnummer,@String-Variable1               *
*             /,String-Variable2 /,.../// <ENTER>                    *
*                                                                    *
*     Funktion: Schreiben eines Datensatzes in die relative          *
*               Ramdatei.                                            *
*                                                                    *
**********************************************************************
```

Der Inhalt der übergebenen String-Variablen (nur solche sind zulässig!) wird in die entsprechenden Felder des übergebenen Datensatzes geschrieben. Die Satznummer darf zwischen 0 und REC-1 liegen (siehe !RECORDS).

Beispiel:

```
!ramwrite,71,@a$,@b$,@c$ <ENTER>
```

würde den String-Variablen A$, B$ und C$ in den 71.ten Datensatz der Ramdatei eintragen.

```
**********************************************************************
*                                                                    *
*    Aufruf: !RAMREAD,Rekordnummer,@String-Variable1                 *
*            /,String-Variable2 /,...// <ENTER>                      *
*                                                                    *
*    Funktion: Lesen eines Datensatzes aus der relativen             *
*              Ramdatei.                                             *
*                                                                    *
**********************************************************************
```

Der Inhalt des angewählten Datensatzes wird in die übergebenen String-Variablen eingelesen. Die Nummer des Datensatzes darf zwischen 0 und REC-1 liegen.

Beispiel:

```
!ramread,45,@a$,@b$,@c$ <ENTER>
```

würde den Inhalt des Datensatzes Nummer 45 in die String-Variablen A$, B$ und C$ einlesen.

```
**********************************************************************
*                                                                    *
*    Aufruf: !RAMCLOSE <ENTER>                                       *
*                                                                    *
*    Funktion: Schließen der relativen Ramdatei.                     *
*                                                                    *
**********************************************************************
```

Dieser Befehl schließt die mit !RAMOPEN geöffnete Ramdatei.

```
**********************************************************************
*                                                                    *
*    Aufruf: !RECORDS <ENTER>                                        *
*                                                                    *
*    Funktion: Errechnen der möglichen Anzahl von Daten-             *
*              sätzen in der Relativen Ramdatei.                     *
*                                                                    *
**********************************************************************
```

Wird nach einem !RAMOPEN-Befehl der Befehl !RECORDS eingegeben, so wird die Zahl der möglichen Datensätze errechnet und in der Systemvariablen REC abgelegt.

Beispiel:

```
10 !bos,0,0,0,0,16
20 !ramopen,128
210 !records
220 print rec
```

gibt die Anzahl der möglichen Datensätze bei maximaler Ramdatei-Größe und einer Datensatzlänge von 128 Bytes aus.

DER ROM-RESIDENTE MONITOR

Der eingebaute Monitor, mit dem Maschinensprachprogramme geladen, getestet und wieder abgespeichert werden können, ist ein leistungsfähiges Werkzeug für Programmierer, das sofort und ohne wesentlichen Speicherplatzverlust zur Verfügung steht. Er beinhaltet vielerlei Funktionen, wie zum Beispiel Disassemblieren, Anzeigen und Ändern von Speicher- und Registerinhalten, Setzen von Breakpoints, Einzelschrittausführung, etc. Diese werden desweiteren ausführlich erklärt. Der Monitor wird mit dem RSX-Befehl |XM gestartet (! erreichen Sie mit SHIFT und der Taste mit dem 'Klammeraffen' !) und meldet sich mit dem Zeichen * und einem blinkendem Cursor. Das *-Zeichen wird im folgendem als Prompt be-zeichnet. Sie können den Monitor durch Drücken der ESC-Taste immer dann verlassen, wenn dieser Prompt sichtbar ist und befinden sich danach an der Stelle nach dem Aufruf (d.h. der Monitor kann auch aus einem Programm heraus aufgerufen werden, und das Programm läuft nach der Rückkehr weiter).

Die Befehle des Monitors

Nachfolgend werden in alphabetischer Reihenfolge die Befehle des Monitors erklärt. Zu beachten ist dabei, daß verlangte oder angezeigte Werte grundsätzlich in hexadezimaler Form dargestellt werden, wobei 8-Bit-Werte zweistellig und 16-Bit-Werte vierstellig sind, bzw. sein müßen.

A adresse - ASCII-Text eintragen

Nach Eingabe einer Adresse können Sie eine Zeichenkette eingeben, die dann nach der Bestätigung durch ENTER ab der eingegebenen Adresse eingetragen wird.

B - Breakpoints ansehen/ändern

Der Monitor bietet die Möglichkeit bis zu acht Breakpoints (=Unterbrechungspunkte) beliebig zu setzen. Trifft der Monitor beim Ablauf eines Programmes (siehe G-Befehl) auf einen dieser Breakpoints, so wird der Programmablauf unterbrochen, und der Monitor-Prompt * ausgegeben. Die Ausgabe der Breakpoints erfolgt derart:

-B1-	-B2-	-B3-	-B4-	-B5-	-B6-	-B7-	-B8-
0000	0000	0000	0000	0000	0000	0000	0000

Sie können dann den ersten Breakpoint ändern oder mittels <ENTER> zum nächsten Breakpoint gelangen, usw. bis nach dem achten Breakpoint oder durch Drücken von <ESC> der Monitor-Promt * wieder erscheint.

C adresse,länge - Zeichenkette suchen

Sie geben eine Adresse und eine Länge ein. Der Monitor sucht nun alle Übereinstimmungen im Speicher mit den Inhalten ab der eingegebenen Adresse und gibt alle Adressen mit Übereinstimmungen auf dem Bildschirm aus.

Beispiel:

Wenn ab Adresse 4000 von Ihnen die Wert 2 und 3 eingegeben wurden und Sie den C-Befehl so starten

C4000,2 <ENTER>

so werden Ihnen alle Adressen ausgegeben, ab denen die Werte 2 und 3 im Speicher stehen.

D adresse1,adresse2 - Auflisten in hexadezimalem und ASCII-Format

Der D-Befehl gibt Ihnen die Möglichkeit Speicherinhalte in folgendem Format anzuzeigen:

0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Am Anfang der Zeile wird Ihnen die Adresse der momentanen Speicherzelle angezeigt, und darauf die Inhalte dieser und der fünfzehn folgenden Speicherzellen in hexadezimaler Form. Danach erfolgt eine Interpretation der Werte als ASCII-Zeichen. Ist der Wert kleiner als 32 oder grösser als 127, so wird ein Punkt, ansonsten aber das entsprechende ASCII-Zeichen ausgegeben. Geben Sie nach Drücken der Taste D nur eine Adresse ein, so werden die nach-folgenden 100H Speicherinhalte ausgegeben, drücken Sie allerdings nach der ersten Adresse nicht <ENTER>, sondern die Komma-Taste, so können Sie eine zweite Adresse eingeben. Es werden dann von der ersten Adresse bis zur zweiten Adresse die Speicherinhalte aufgelistet. Eine solche Auflistung können Sie durch Drücken einer beliebigen Taste kurz-fristig anhalten. Drücken Sie nun <ESC>, so erscheint wieder der Monitor-Prompt *, drücken Sie aber eine beliebige andere Taste, so läuft das Listing weiter.

F adresse1,adresse2,wert - Auffüllen eines Speicherbereichs mit einem Wert

Oft kommt es vor, daß bestimmte Speicherbereiche z.B. gelöscht werden sollen. Dies ist mit dem F-Befehl möglich, wobei von adresse1 bis adresse2 die Speicherinhalte mit der 8-Bit-Konstanten wert belegt werden.

Beispiel:

```
FC000,FFFF,FF <ENTER>   füllt den Bildschirspeicher (Adressen C000
*                       mit dem Wert FF.
```

G adresse

Dieser Befehl führt ein ab der eingegebenen Adresse befindliches Maschinensprachprogramm aus. Wird hierbei ein Breakpoint (s.o.) erreicht, so geht die Kontrolle an den Monitor zurück.

Beispiel:

```
GBD19  <ENTER>      führt das Maschinensprachprogramm ab der
                    Adresse BD19 aus.
```

I filename - Laden einer Binärdatei von Diskette/Cassette

Wurde der Monitor aus Basic heraus aufgerufen, so kann mit diesem Befehl vom selektierten Eingabegerät (Diskette oder Cassette) eine Binärdatei geladen werden. Sie wird dabei in den originalen Speicher, d.h. in den Speicherteil aus dem sie erstellt wurde, geladen. Es werden die Ladeadresse, die Länge, sowie eine eventuell vorhandene Startadresse angezeigt.

Beispiel:

```
ITEST.BIN <ENTER>       lädt die Datei TEST.BIN in den Speicher.
<2000,0380,2140>        Die ermittelte Anfangsadresse ist 2000,
*                       die Länge 380 und die Startadresse 2140.
```

L adresse1,adresse2 - Disassemblieren eines Speicherbereiches

Die Eingabe und Bedeutung der Adressen entspricht denen des D-Befehls (s.o.), wobei allerdings der Speicherinhalt in Form von Mnemonics dargestellt wird.

Beispiel:

```
L0100,0106 <ENTER>
0100 3E64       LD    A,64H
0102 ED5B3204   LD    BC,(0432H)
0106 E3         EX    (SP),HL
*
```

Listet den Bereich von 100 bis 106. Am Anfang steht die Adresse, dann deren Inhalt in hexadezimaler Form und dann der lesbare Z80-Mnemonic.

M adresse1,adresse2,adresse3

Der Inhalt des Speicherbereiches zwischen adresse1 und adresse2 wird in den Speicherbereich beginnend mit adresse3 kopiert.

Beispiel:

```
M0000,3FFF,C000 <ENTER>   kopiert den Bereich von 0 - 3FFF in den
*                         Speicherbereich ab Adresse C000 (=Bild-
                          schirm).
```

Ofilename,adresse1,länge,adresse2 - Abspeichern eines Speicherbereiches

Wenn Sie nach dem O einen zulässigen Filenamen eingegeben haben, wird endsprechend dem Basic-Befehl SAVE "name",B nach Anfangsadresse des Speicherbereiches, dessen Länge und einer Startadresse gefragt. Danach wird dieser Speicherbereich auf dem selektierten Ausgabegerät (Diskette oder Cassette) abgespeichert.

Beispiel:

```
OTEST.BIN,C000,4000,0000     Es wird in die Binärdatei TEST.BIN
*                            der Speicherbereich von C000 bis
                             FFFF ohne Startadresse abgespeichert
```

P - Ein-/Ausschalten des Druckers

Wenn ein Drucker angeschlossen ist, wird die Bildschirmausgabe durch einmaliges Drücken der Taste P auf diesem mitdokumentiert. Nochmaliges Drücken der Taste P schaltet den Drucker wieder ab.

R - Anzeigen/Ändern der Registerinhalte

Die Registerinhalte für die Abarbeitung eines Maschinensprachprogrammes (mit G oder T) sind mit diesem Befehl änderbar. Sie werden ähnlich den Breakpoints dargestellt und genauso editiert. Es ist nur der Erst-Registersatz änderbar.

S adresse - Speicherinhalt anzeigen/ändern

Nach Eingabe von S und einer Adresse sehen Sie deren Inhalt und können diesen in einer dritten Spalte entsprechend ändern. Durch Drücken von <ENTER> gehen Sie weiter zur nächsten Speicherzelle, der eventuell geänderte Wert der alten Adresse wird in diese eingetragen. Durch drücken von <ESC> kommen Sie wieder zum Monitor-Prompt *.

Beispiel:

```
S4000 <ENTER>          ändern ab Speicherzelle 4000.
4000 $ 24 34 <ENTER>   Die alten Werte der Adresse 4000, 4001
4001 C 43 32 <ENTER>   und 4002 (24, 43, und 58) werden in die
4002 : 58 44 <ENTER>   neuen Werte 34, 32 und 44 abgeändert.
4003 <ESC>
*
```

T adresse,anzahl - Maschinenprogramm schrittweise ausführen.

Ein Maschinenprogramm kann mit diesem Befehl schrittweise ausgeführt werden, um eventuell vorhandene Fehler zu entdecken. Es wird dabei von dem mit dem R-Befehl eingegebenen Registerwerten ausgegangen und die Kontrolle wird nach jedem Schritt an den Monitor zurückgegeben, d.h. Sie sehen die eventuell geänderten Register und können die Abarbeitung auch unterbrechen, wenn Sie eine optionelle Schrittanzahl angegeben haben. Diese Unterbrechung erreichen Sie wie gewohnt mit ESC. Beachten Sie bitte, daß Programme im ROM oder Befehle die mit dem Zweitregistersatz des Z80 arbeiten nicht schrittweise abgearbeitet werden können. Desweiteren sollte der Stackpointer SP nicht unterhalb eines ROMs liegen, was zum Absturz des Systems führen kann.

Beispiel:

```
T4000,0007
```

führt sieben Schritte ab Adresse 4000 aus. Die Registerinhalte werden immer angezeigt und der Ablauf kann wie beim D-Befehl angehalten oder abgebrochen werden.

X - den unteren ROM ein/ausschalten

Der X-Befehl dient als Schalter um den unteren ROM (das Betriebssystem) entweder ein- oder auszuschalten. Die Befehle D, L und M wirken dann entweder auf den ROM oder das darunterliegende RAM.

Y wert - einen oberen ROM auswählen.

Ab Adresse C000 können Background-ROMs liegen, welche Sie mit diesem Befehl selektieren können. Nach Eingabe von Y wird Ihnen eine Liste aller verfügbaren Background-ROMs und deren Romnummer ausgegeben. Sie können dann durch Eingabe der Romnummer den entsprechenden Background-ROM anwählen. Geben Sie 00 ein, so wird der Basic-ROM eingeblendet, geben Sie eine Zahl größer FC ein, so wird kein ROM eingeblendet.

Beispiel:

```
*Y
04  000100 M WLK
05  010001 PROTEXT
07  000300 V ROM
Y07 <ENTER>
```

Jetzt beziehen sich alle D-, L- und M-Befehle mit Adressen über BFFF auf den ROM mit dem Namen "V ROM" (das Disketten-Betriebssystem VDOS).

Z wert - Einblenden einer anderen Speicherbank

Mit diesem Befehl können Sie eine andere Speichererweiterungs-Bank einblenden und sämtliche Schreib/Lesezugriffe beziehen sich in der Folge auf die entsprechende Ram-Bank. Geben Sie als Wert eine Null ein, so ist der Original-Speicher des CPCs selektiert. Mit Werten zwischen 1 bis 8 wählen Sie eine der zusätzlichen Speichererweiterungsbänke.
Es ist dabei zu beachten, daß zwar bei Schreib/Lesezugriffen die entsprechend angewählte Speicherbank benutzt wird. Ansonsten ist allerdings immer die Original-Bank aktiv, d.h Laden und Abspeichern mit dem Befehlen I und O, sowie Tracen und Ausführen von Maschinenprogrammen mit T und G, sind nur in dieser Original-Bank möglich.

WEITER NÜTZLICHE RSX-BEFEHLE

```
**********************************************************************
*                                                                    *
*     Aufruf: !FAST <ENTER>                                          *
*                                                                    *
*     Funktion: Verschnellerte Bildschirmausgabe im Mode 2.          *
*                                                                    *
**********************************************************************
```

Wenn Sie sich in der 80-Zeichen-Bildschirmausgabe (Mode 2) befinden, haben Sie die Möglichkeit, die Ausgabe von Text um den Faktor 2 zu beschleunigen. Der einzige Nachteil, den Sie in Kauf nehmen müssen, ist eine nicht mehr einwandfrei arbeitende Window-Technik. Bei normaler Ausgabe von Text ohne definierte Windows arbeitet die verschnellerte Bildschirmausgabe vollgültig wie die standardmäßige Ausgabe. Die normale Ausgabe kann mit dem Befehl !SLOW wieder aktiviert werden.
Als Demonstration für die Verschnellerung können Sie folgenden Einzeiler ein-geben: 10 Z=Z+1:?Z:GOTO 10 . Starten Sie dieses Programm mit RUN, so sehen Sie die normale Ausgabe. Unterbrechen Sie das Programm, tippen !FAST <ENTER> und starten das Programm wieder mit RUN <ENTER> , so werden Sie die drastische Geschwindigkeitssteigerung sofort feststellen können.

```
**********************************************************************
*                                                                    *
*     Aufruf: !FRAME <ENTER>                                         *
*                                                                    *
*     Funktion: Erzeugen von flackerfreier bewegter Graphik.         *
*                                                                    *
**********************************************************************
```

Dadurch daß die Ausgabe auf dem Bildschirm normalerweise nicht mit dem reelen Bildschirmaufbau des Monitors synchronisiert wird, kann es in bestimmten Fällen zu einem flackernden Bildschirmaufbau kommen. Als Beispiel können Sie folgendes kurzes Programm eingeben:

```
10 MODE 2
20 FOR I=1 TO 80
30 LOCATE I,12:?"o":LOCATE I,12:?" "
40 NEXT
50 GOTO 20
```

Wenn Sie das Programm starten, wird das Zeichen o relativ ruckhaft von links nach rechts bewegt. Fügen Sie aber die folgende Zeile ein, so werden Sie nach dem Neustart des Programms eine vollkommen fließende Bewegung sehen. Die Zeile lauted:

```
35 !FRAME
```

```
**********************************************************************
*                                                                    *
*     Aufruf: !GCHAR,Spalte,Reihe,@Integervariable <ENTER>           *
*                                                                    *
*     Funktion: Lesen eines Zeichens vom Bildschirm.                 *
*                                                                    *
**********************************************************************
```

Wenn Sie beispielsweise feststellen wollen, welches Zeichen sich an der Position 12,14 auf dem Bildschirm befindet, so können Sie dies mit diesem Befehl ermitteln. Sie müssen dazu zunächst eine Variable definieren, in die der ASCII-Wert des ermittelten Zeichens dann eingetragen wird. Diese Variable muß eine Integer-Varible sein, d.h. bevor Sie den GCHAR-Befehl benutzen müssen Sie beispielsweise die Variable A wie folgt definieren:

```
10 DEFINT A:A=0
```

Benutzen Sie danach den Befehl !GCHAR,X,Y, A , dann wird der ASCII-Wert des Zeichens an den Koordinaten X/Y in die Variable A eingetragen. Diesen Wert können Sie dann mit PRINT A ansehen. Bitte beachten Sie, daß unzulässige Koordinaten (Y zwischen 1 und 25 , X zwischen 1 und dem Maximalwert im jeweiligen Mode) oder durch das Anwenden von Graphikbefehlen nicht mehr identifizierbare Zeichen den Wert 0 zurückmelden.

```
**********************************************************************
*                                                                    *
*     Aufruf: !GPAPER,wert <ENTER>                                   *
*                                                                    *
*     Funktion: Einstellen der Graphik-Hintergrundfarbe.             *
*                                                                    *
**********************************************************************
```

Mit dem Befehl !GPAPER,wert weisen Sie der Graphik-Hintergrundfarbe einen neuen Farbstift zu. Eine ausführliche Erklärung finden Sie beim !MASK-Befehl.

```
Bsp.:   !GPAPER,3   - der neue Farbstift für die Graphik-Hinter-
                      grundfarbe ist der Farbstift Nummer 3.
```

```
**********************************************************************
*                                                                    *
*     Aufruf: !GPEN,wert <ENTER>                                     *
*                                                                    *
*     Funktion: Einstellen der Graphik-Vordergrundfarbe.             *
*                                                                    *
**********************************************************************
```

Mit dem Befehl !GPEN,wert weisen Sie der Graphik-Vordergrundfarbe einen neuen Farbstft zu. Eine ausführliche Erklärung finden Sie beim Mask-Befehl.

```
Bsp.:   !GPEN,2   - der neue Farbstift für die Graphik-Vorder-
                    grundfarbe ist der Farbstift Nummer 2.
```

```
**********************************************************************
*                                                                    *
*    Aufruf: |MASK,wert <ENTER>                                      *
*                                                                    *
*    Funktion: Definieren einer Maske für die Graphikausgabe.        *
*                                                                    *
**********************************************************************
```

Punkte und Linien werden am CPC 464 normalerweise nur in einer Farbe ausgegeben. Mit Hilfe des MASK-Befehls ist es nun möglich, auch zweifarbig gemusterte Graphikausgaben leicht zu erreichen. Wird mit |MASK eine Maske übergeben, so ist diese Maske des weiteren das verwendete Muster. Das Muster entspricht dem binären Wert der übergebenen Zahl. Nullen in dieser Maske ergeben einen Punkt in der Graphik-Hintergrundfarbe, Einsen einen Punkt in der Graphik-Vordergrundfarbe. Benutzen Sie beispielsweise den Befehl |MASK,&55 , so wird der Wert &55 übergeben, der binär dargestellt so aussieht: 01010101 . Das heißt, daß z.B. das Zeichnen einer Linie per DRAW X,Y eine abwechselnd zwischen Graphik-Vordergrund- und Hintergrundfarbe wechselnde Linie ergibt.

```
**********************************************************************
*                                                                    *
*    Aufruf: |UNMASK <ENTER>                                         *
*                                                                    *
*    Funktion: Aufheben der Graphik-Maskierung.                      *
*                                                                    *
**********************************************************************
```

Nach Eingabe des |UNMASK-Befehls ist die Graphikausgabe des CPC 464 wieder in ihrer ursprünglichen Form, d.h. jegliche Masken-definition mittels des |MASK-Befehls wird aufgehoben.

```
**********************************************************************
*                                                                    *
*    Aufruf: |SLOW <ENTER>                                           *
*                                                                    *
*    Funktion: Aufheben der FAST-Routine.                            *
*                                                                    *
**********************************************************************
```

Mittels des Befehls |SLOW wird die Textausgabe auf die normale Geschwindigkeit zurückgesetzt.

```
**********************************************************************
*                                                                    *
*    Aufruf: |ROMOFF /,ROM-Nummer1 /,.../ <ENTER>                    *
*                                                                    *
*    Funktion: Ausschalten von Background-ROMs.                      *
*                                                                    *
**********************************************************************
```

Mit diesem Befehl können Sie entweder alle Background-ROMs abschalten (wenn kein Parameter mit übergeben wurde) oder aber auch selektiv durch Angabe der Nummern der abzuschaltenden

Background-ROMs. Als Beispiel:

!ROMOFF,6,7 <ENTER>

schaltet die Roms mit den Nummern 6 und 7 aus. Welche ROMs welche Nummern haben, erfahren Sie mit dem Befehl !ROMCAT.

```
**********************************************************************
*                                                                    *
*     Aufruf: !DISBOS <ENTER>                                        *
*                                                                    *
*     Funktion: Abschalten des BOS-ROMs.                             *
*                                                                    *
**********************************************************************
```

Dieser Befehl entspricht in der Funktion dem !ROMOFF-Befehl mit selektiver Angabe der BOS-ROM-Nummer. Er schaltet nur den BOS-ROM ab und gibt damit auch den von diesem ROM reservierten RAM-Speicherplatz wieder frei.

```
**********************************************************************
*                                                                    *
*     Aufruf: !ROMCAT /,ROM-Nummer/ <ENTER>                          *
*                                                                    *
*     Funktion: Auflisten aller vorhandenen Background-ROMs          *
*               oder deren RSX-Befehle.                              *
*                                                                    *
**********************************************************************
```

Ruft man den Befehl !ROMCAT ohne einen Parameter auf, so erhält man eine Liste aller vorhandenen Background-Roms. Diese kann beispielsweise so aussehen:

```
Himem: 9BDD

Nr.  Ty   Ma   Ve   Mo   Base  Name

00   80   01   00   00   ----  BASIC
01   80   01   00   00   ----  BASIC
02   80   01   00   00   ----  BASIC
03   80   01   00   00   ----  BASIC
04   80   01   00   00   ----  BASIC
05   01   00   01   00   9C62  M WLK
06   01   00   00   02   9C66  S I O
07   01   00   04   00   A50C  V ROM
```

In der ersten Spalte wird die Nummer des ROMs ausgegeben, die zweite bis fünfte Spalte sind die ersten vier Bytes des ROMs, in denen der Programmierer Statusinformationen abgelegt hat. In der nächsten Spalte wird die Basis-Adresse des vom jeweiligen ROM reservierten RAM-Speichers angezeigt und in der letzten Spalte steht dann noch der Name des ROMs.
Gibt man beim Aufruf des !ROMCAT-Befehls eine ROM-Nummer als Parameter an, so werden alle RSX-Befehle dieses ROMs und die jeweilige Startadresse im ROM ausgegeben.

FÜR VERSIERTE ASSEMBLER-PROGRAMMIERER

Speziell an die Maschinensprach-Programmierer haben wir beim Einbau eines "versteckten" RSX-Befehles gedacht. Dieser Befehl wird genauso wie die versteckten Befehle des DOS mit der Betriebssystemroutine KL FIND COMMAND gesucht und kann danach aufgerufen werden. Er hat den "Namen" hexadezimal 10 und ist deshalb nicht direkt aufrufbar.

Je nach Inhalt des Akku führt dieser Befehl dann verschiedene Funktionen aus. Diese sind:

Akku = 0
Register C = Banknummer (0 bis 8)
Registerpaar HL = Adresse

Funktion: Byte aus der Adresse (HL) aus der im C-Register stehenden Bank lesen. Steht in C eine Null, so wird aus der Original-Bank gelesen, steht in C ein Wert zwischen eins und acht, so wird aus der entsprechenden physikalischen Bank gelesen. Der gelesene Wert wird im B-Register zurückgeliefert.

Akku = 1
Register B = zu schreibender Wert
Register C = Banknummer (0 bis 8)
Registerpaar HL = Adresse

Funktion: Den in B übergebenen Wert in die Adresse (HL) der in C übergebenen Bank schreiben.

Akku = 2
Register C = Banknummer (0 bis 8)

Funktion: Die obere Hälfte der in C übergebenen Speicherbank einblenden. Die Routine kommt mit unterdrücktem Interrupt zurück.

Akku = 3
Register C = Banknummer (0 bis 8)

Funktion: Die untere Hälfte der in C übergebenen Speicherbank einblenden. Die Routine kommt mit unterdrücktem Interrupt zurück.

Akku = 4

Funktion: Den alten Speicherzustand wiederherstellen. Die Routine kommt mit freigegebenem Interrupt zurück.

Wie man diesen RSX-Befehl unter Maschinensprache aufruft, will ich Ihnen anhand des folgenden Beispieles erklären:

```
            ld    hl,command          ; Adresse des RSX-Namens.
            call  kl_find_command     ; Firmware-Routine RSX suchen.
            ret   nc                  ; nicht gefunden --> zurück.
            ld      (faraddr),hl      ; gefunden, d.h. aus der
            ld    a,c                 ; Routinenadresse und der
            ld    (faraddr+2),hl      ; Romnummer einen Far-Call
                                      ; herstellen.
            ld    a,3                 ; Funktion Nummer 3
            ld    c,5                 ; "Bank 5 unten einmappen"
            rst   18H                 ; aufrufen.
            dw    faraddr             ;
            ret                       ;
command:    db    90H                 ;
faraddr:    ds    3                   ;
```

REFERENZKARTE DER BOS 2.0-RSX-BEFEHLE

l,m,n,o	Nummer einer Basic-Programmbank
addr	Speicheradresse
/./	optionale Eingabe
par1,...	Übergabeparameter bei einem !CALL
chn	Kanalnummer des Ausgabegerätes
line	Zeilennummer eines Basic-Programms
var	Integervariable
var$	Stringvariable
wert	Integerzahl
len1,...	Feldlänge
reclng	Rekordlänge
recnum	Rekordnummer
i	Bildnummer (0-31)

Befehl	Syntax
!A	
!ATTRIBUT	,@var$,@var$
!B	
!BACKWARD	/,i/
!BANK	,n
!BASIC	
!BOS	,wert1/,wert2/,wert3/,wert4/,wert5////
!CALL	,n,addr/,par1/,par2/,...///
!CLOSE	,wert
!COMMON	/,@var$/,l/,m/,n/,.../////
!DEV	/,chn/
!DIR	/,@var$/
!DISBOS	
!DRIVE	,@var$
!ERA	,@var$
!FAST	
!FIELD	wert1/,wert2/,...///
!FILES	wert1,wert2/,wert3,wert4/,...///
!FORMAT	
!FORWARD	/,i/
!FRAME	
!GCHAR	,x,y,@var
!GET	wert,recnum,@var1$/,@var2$/,...///
!GOSUB	,n,line
!GOTO	,n,line
!GPAPER	,wert
!GPEN	,wert
!ID	
!LCALL	,n,addr/,par1/,par2/,...///
!LIST	/,l/,m/,...///
!LOAD	,@var$
!LPEEK	,n,addr,@var
!LPOKE	,n,addr,wert
!MASK	,wert
!MD	

```
!NEW            /,l/,m/,...////
!OPEN           @var$,wert
!PEEK           ,n,addr,@var
!POKE           ,n,addr,wert
!PUT            wert,recnum,@var1$/,@var2$/,.../
!RAMCLOSE
!RAMFIELD       ,len1/,,len2/,.../
!RAMOPEN        ,reclng
!RAMREAD        ,recnum,@var1$/,@var2$/,.../
!RAMWRITE       ,recnum,@var1$/,@var2$/,.../
!RECORDS
!REN            ,@var$,@var$
!RESET
!RETURN
!ROMCAT         /,wert/
!ROMOFF         /,wert1/,wert2/,.../
!RUN            ,@var$/,n,line/
!SAVE           ,@var$/,l/,m/,.../
!SCR.BASE       ,wert
!SCREEN.IN      ,i
!SCREEN.OUT     ,i
!SCREENS
!SELECT         ,@var$
!SLOW
!SPOOL.CONT
!SPOOL.ON
!SPOOL.OFF
!UNMASK
!USER           ,wert
!VIDEO.ON
!XM
```

Vortex Computersysteme GmbH