

Einleitung

Mit dem *ALPHA-ROM* besitzen Sie eine leistungsfähige Erweiterung für die *EPROM-Karte 264* oder *224* der Firma *Dobbertin-Industrie-Elektronik GmbH*.

Bei der Entwicklung wurde grösster Wert auf Kompatibilität zu allen drei CPCs gelegt. Wer sich bereits intensiv mit der maschinennahen Programmierung der Schneider-Rechner beschäftigt hat, wird wissen, wie schwierig dies manchmal sein kann. Aber auch die ROMs der englischen Firma *Arnor Ltd.* (z.B. *MAXAM*, *PROTEXT*, *UTOPIA* ...) wurden - soweit verfügbar - berücksichtigt. Speziell in diesem Zusammenhang gibt es jedoch auch ein paar Kleinigkeiten zu beachten. Sollten Sie deshalb bei der Benutzung des *ALPHA-ROMs* auf Fehler oder Ungereimtheiten stoßen, so wären wir Ihnen sehr dankbar, wenn Sie uns eine möglichst genaue Beschreibung des Fehlers (oder sogar Verbesserungsvorschläge) schicken würden. Vergessen Sie bei solchen Fehlerbeschreibungen bitte nicht, auch den verwendeten Rechner, vorhandene Erweiterungen (z.B. *Floppy Schneider/Vortex*, *RAM-Erweiterung*, *ROMs* ...) oder sonstige 'Ungewöhnlichkeiten' aufzulisten.

Als besonderes Extra für die Assembler Programmierer ist auch ein dokumentiertes Listing des ROMs erhältlich. Damit sollte es Ihnen dann möglich sein, den restlichen freien Bereich des ROMs mit sinnvollen Erweiterungen aufzufüllen.

Installation

Die *EPROM-Karte 264* stellt Ihnen vier Sockel zur Verfügung, von denen jeder mit bis zu 16 KByte ROM bestückt werden kann. Dabei nimmt der 1. Sockel eine Sonderstellung ein, da er die 'ROM-Nummer 0' besitzt. Den 'Spezialisten' unter Ihnen dürfte dies nicht neu sein und sie wissen, was daran so besonders ist. Für den 'normalen' Anwender sei dazu nur soviel gesagt, daß diese ROM-Nummer normalerweise dem BASIC-ROM, das in Ihrem CPC steckt, zugeordnet ist. Der Vorteil liegt klar auf der Hand; es ist somit ein Leichtes, Änderungen oder Erweiterungen des BASIC-Interpreters vorzunehmen, da man hierzu lediglich ein neues BASIC-ROM (oder genauer gesagt ein EPROM) in den ersten Sockel der EPROM-Karte stecken muß. Dieses EPROM übernimmt dann nach dem Einschalten des Rechners die Kontrolle und schaltet das interne BASIC-ROM einfach ab. Der Nachteil, der sich hierdurch ergibt, sei jedoch nicht verschwiegen; wenn man das BASIC-ROM gar nicht ändern will, sondern z.B. nur ein paar der inzwischen zahlreich erhältlichen ROM-Module nutzen will, dann muß man trotzdem den 1. Sockel mit einer Kopie des BASIC-ROMs bestücken. Man verliert dadurch immerhin 16 KByte, die man sonst mit einem anderen ROM hätte bestücken können. Dazu kommt noch die Schwierigkeit, sich eine Kopie des BASIC-ROMs zu beschaffen, wenn man sich nicht gerade zu den glücklichen Besitzern eines EPROM-Programmiergeräts zählen kann. Aber damit ist jetzt Schluß, denn das *ALPHA-ROM* kann in allen vier Sockeln der *EPROM-Karte 264* betrieben werden. Wenn Sie also bisher keine geänderte Version des BASIC-ROMs besessen haben, dann sollten Sie das *ALPHA-ROM* in den 1. (den am weitesten vom CPC entfernten) Sockel stecken. Sollte dieser Sockel jedoch bereits von einem ROM belegt sein, das auch weiterhin benötigt wird, dann stecken Sie das EPROM in einen anderen noch unbelegten Sockel.

Nachdem Sie das *ALPHA-ROM* nun auf der EPROM-Karte platziert haben, schalten Sie den CPC wie gewohnt ein. Abgesehen davon, daß HIMEM nun 4 Byte tiefer sitzt als bisher (und das auch nur, wenn das *ALPHA-ROM* nicht im 1. Sockel steckt), sollten Ihnen keine Veränderungen des CPCs auffallen (Was jedoch nicht heißt, daß sich nichts geändert hat). Als kleinen Funktionstest geben Sie bitte **:HELP**, gefolgt von **ENTER** ein. Auf dem Bildschirm sollte nun eine Liste der vorhandenen ROMs erscheinen.

RSX-Befehle und 'Strings'

Den Besitzern eines Floppy-Controllers ist es sicher schon aufgefallen, daß es nicht ohne weiteres möglich ist, bei einem RSX-Befehl einen String als Parameter zu übergeben. Ein kleiner Auszug aus dem Floppy-Handbuch der Firma *Schneider* zeigt deutlich, was damit gemeint ist:

```
:DIR [,@<String-Variable>]
```

Beispiel: **f\$="*.BAS"**
:DIR,@f\$

(Dieser Befehl gibt das Inhaltsverzeichnis der Diskette aus, wobei jedoch nur die Files mit dem Extension 'BAS' angezeigt werden. Die eckigen Klammern zeigen an, daß es sich bei der Angabe der Stringvariablen um eine Option handelt, die wahlweise verwendet werden kann.)

Das ist natürlich eine recht umständliche Angelegenheit, wenn man vorher erst eine Stringvariable mit dem gewünschten Text definieren und diese Variable dann mit dem '@'-Zeichen an den RSX-Befehl übergeben muß. Die Entwickler des *CPC*-Betriebssystems haben diesen Mißstand ebenfalls erkannt und beim CPC 664/6128 behoben. Das nützt jedoch den Besitzern eines CPC 464 reichlich wenig. Aus diesem Grund 'patched' das *ALPHA-ROM* den 'ERROR-Vektor' des CPC 464 (nicht beim CPC 664/6128, da ist das ja nicht nötig) und ermöglicht es so, daß Strings direkt an den RSX-Befehl übergeben werden können. Das obige Beispiel sieht dann also folgendermaßen aus:

```
:DIR,"*.BAS"
```

Selbstverständlich kann auch die 'alte' Stringübergabe ohne Einschränkungen weiterhin verwendet werden.

Der 8-Bit Druckerpatch

Im Gegensatz zu vielen anderen Home- und Personal-Computern besitzen die CPCs bereits serienmäßig eine parallele Druckerschnittstelle, über die fast jeder handelsübliche Drucker mit Centronicsport angeschlossen werden kann. Parallel heißt, daß immer 8 Bit gleichzeitig übertragen werden. Doch leider hat die Sache einen kleinen Haken. Beim CPC werden nur 7 der sonst üblichen 8 Bit angesteuert. Das fällt im 'normalen' Betrieb nicht weiter auf, da beim Drucken von Texten nur 7 Bit benötigt werden. Schwierig wird es jedoch, wenn man Graphikzeichen

mit 'gesetztem' 8. Bit ausdrucken will. Einige Drucker stellen dazu spezielle Befehle zur Verfügung, mit denen man das 8. Bit ausdrücklich setzen oder löschen kann. Aber spätestens beim sogenannten Bit-Image-Mode versagen auch diese Befehle. Diese Schwachstelle kann man jedoch beseitigen. In fast allen Computerzeitschriften, die sich mit den Schneider Computern beschäftigen, sind Umbauanleitungen erschienen, mit denen man das 8. Bit der Druckerschnittstelle zum Leben erwecken kann. Doch wie so oft, ist es mit der reinen Hardwareänderung nicht getan, denn das Betriebssystem des CPC weiß ja nichts davon. Deshalb besitzt das *ALPHA-ROM* einen eigenen Druckertreiber, der diese Hardwareänderung unterstützt. Eine Beschreibung der Hardwareänderung finden Sie z.B in folgenden Zeitschriften :

CPC Magazin 8/87 S.98-99, CPC International 5/86 S.112,
c't 10/85 S.72-73, Happy Computer 7/85 S.26-27, ...

Die ROM-Nummern 8 - 14 beim CPC 464

Im Gegensatz zu ihren meisten Konkurrenten besitzen die CPCs die Fähigkeit, Erweiterungs-ROMs zu verwalten. Bis zu 252 ROMs mit einer Kapazität von jeweils 16 KByte sind theoretisch möglich. Jedes ROM besitzt dabei eine sogenannte ROM-Nummer. Diese ROM-Nummer im Bereich von 1 - 252 ist eine Art Kennung, unter der ein einzelnes ROM gezielt angesprochen werden kann. Die ROMs mit der Nummern 1 - 7 werden allerdings von dem Betriebssystem des CPC 464 bevorzugt behandelt (beim CPC 664 & 6128 die ROMs mit der Nummer 1 - 15). Die ROMs mit diesen Nummern erhalten nach dem Einschalten des CPC die Möglichkeit, sich Speicherplatz zu reservieren und eigene RSX-Befehle in das System einzubinden. Das heist im Klartext, daß die Besitzer eines CPC 464 nur maximal 7 (mit einem Floppy-Laufwerk sogar nur 6) ROMs mit RSX-Befehlen nutzen können. Dies ist für die Besitzer der *EPROM-Karte 264* sicherlich von geringer Bedeutung, da diese Erweiterung sowieso nur 4 Sockel mit fest verdrahteten ROM-Nummern besitzt. Sollten Sie jedoch eine *EPROM-Karte 224* besitzen, die die Möglichkeit bietet, bis zu 14 EPROMs aufzunehmen, dürfte Ihnen diese Tatsache sicherlich schon einiges Kopfzerbrechen bereitet haben. Das *ALPHA-ROM* beinhaltet deshalb eine Routine, die bei einem 464, den ROMs mit den Nummern 8 - 14 die Möglichkeit bietet, sich ebenfalls in das System einzuhängen. ROM-Nummer 15 konnte dabei aus technischen Gründen leider nicht unterstützt werden.

In diesem Zusammenhang sind folgende Punkte erwähnenswert :

- Sollte beim CPC 464 das *ALPHA-ROM* mit dem ROMOFF-Befehl deaktiviert werden, werden damit automatisch auch die ROMs mit der Nummer 8 - 14 ausgeschaltet.
- Das Textsystem PROTEXT besitzt aus unbekannten Gründen eine Abfrage, die einen Betrieb mit einer ROM-Nummer größer 7 nicht zuläßt.
- Das von der Firma Vortex vertriebene Disketten-Betriebssystem 'X-VDOS' belegt, bei Verwendung des RSX-Befehls FAST, genau die Speicherstellen, die das CPC-Betriebssystem bei den ROM-Nummern 8 - 14 benötigt. Das ist einer der Gründe, weshalb das *ALPHA-ROM* prinzipiell nicht mit 'X-VDOS' zusammen betrieben werden kann.

Die Befehle des ALPHA-ROMs

Neben den *passiven Funktionen* des ALPHA-ROMs, die nur indirekt genutzt werden können (wie z.B. der RSX-String-Patch), sind auch Befehle vorhanden, die direkt angesprochen werden können. Bei diesen ROM-Befehlen handelt es sich um sogenannte 'RSX'-Kommandos (RSX ist die Abkürzung für Resident System eXtensions). Die RSX-Kommandos haben ein waschechtes Erkennungsmerkmal, das sie deutlich von den normalen BASIC-Befehlen unterscheidet. Damit ist der 'RSX-Strich' (!) gemeint, der vor jedem RSX-Befehl steht (Er wird von der Tastatur aus über SHIFT-@ erreicht). Ein RSX-Kommando kann maximal 16 Zeichen lang sein (ohne den Strich) und kann bis zu 32 durch Kommas getrennte Parameter besitzen.

Der HELP Befehl

Mit dem RSX-Befehl !HELP erhält man, wie Sie ja bereits gesehen haben, eine Liste der vorhandenen ROMs.

Übergibt man jedoch dem HELP-Kommando noch eine ROM-Nr. (0 - 15) als Parameter, so werden die durch dieses ROM zur Verfügung gestellten Befehlserweiterungen aufgelistet. Geben Sie einmal (in Mode 2) folgendes ein:

!HELP,<ALPHA-ROM Nummer>

Dabei muß der Ausdruck '**<ALPHA-ROM Nummer>**' durch die vom HELP-Kommando angezeigte ROM-Nummer des ALPHA-ROMs ersetzt werden (also z.B. '0', wenn das ALPHA-ROM im 1. Sockel der EPROM-Karte steckt). Auf dem Bildschirm sollte nun in etwa folgender Text erscheinen:

ROM 0:ALPHA ROM	1.00 background		
BANKFIND	BANKOPEN	BANKREAD	BANWRITE
CAT	CLEAR.INPUT	DEBUG.OFF	DEBUG.ON
FRAME	GDUMP	GPAPER	GPEN
HEADER	HELP	LINE	RESET.OFF
RESET.ON	ROMOFF	SCREENCOPY	SCREENSWAP
TYPE	VDU	VDU0	VIEW

Die 1. Zeile enthält neben der ROM-Nummer (hier 0) und dem Namen auch noch die Versionsnummer und den ROM-Typ.

Der ROMOFF Befehl

Manchmal ist es nötig, einzelne oder alle ROMs abzuschalten, weil z.B. ein Programm den Speicherplatz benötigt, den die ROMs nach dem Einschalten des Rechners belegt haben. Für diesen Fall ist der Befehl ROMOFF implementiert worden, der es erlaubt, bestimmte ROMs gezielt zu deaktivieren. Da dabei der Speicher komplett umorganisiert wird, werden alle im RAM stehenden Daten gelöscht. Der Befehl hat folgende Syntax:

!ROMOFF [,<ROM-Nr.>, ... ,<ROM-Nr.>]

Werden keine ROM-Nummern angegeben, dann werden alle ROMs abgeschaltet.

Der HEADER Befehl

Um zum Beispiel ein BASIC-Programm auf Diskette abzuspeichern, stehen Ihnen prinzipiell zwei verschiedene Methoden zur Verfügung :

- das 'normale' Abspeichern mit **SAVE "Name"**
- oder das Abspeichern als ASCII-File mit **SAVE "Name",a**

Der für den HEADER-Befehl relevante Unterschied liegt in der Tatsache, daß beim 'normalen' Abspeichern neben den Programm-Daten auch noch ein sogenannter **File-Header** abgespeichert wird. Dieser **File-Header** hat eine Länge von 128 Byte und beinhaltet folgende Informationen :

- Den Filenamen, unter dem das Programm abgespeichert wurde
- Den Filetyp
- Die Adresse, an die das File eingelesen wird
- Die Filelänge
- Die Einsprungsadresse
- Eine Checksumme

Diese Informationen können mit dem HEADER-Befehl angezeigt und geändert werden.

Die vollständige Syntax sieht folgendermaßen aus:

!HEADER [,"Filename"]

Wird der HEADER-Befehl ohne Angabe eines Filenamen eingegeben, dann wird dieser gezielt vom *ALPHA-ROM* abgefragt.

Ein eventuelles **Unknown command** zeigt an, daß das Diskettenlaufwerk nicht aktiv ist. Bei einem File, das keinen **File-Header** besitzt (z.B.: Text- oder CP/M COM-Files), erhält man einen **File type Error**.

Der FRAME Befehl

Der FRAME-Befehl ist im Prinzip nichts weiter, als ein gut getarnter CALL (denn CALL &BD19 erfüllt die gleiche Funktion). Die Wirkung wird am besten durch ein kleines Beispielprogramm demonstriert. Probieren Sie das folgende kleine BASIC-Listing einmal mit und einmal ohne Zeile 40 aus:

```
10 MODE 2
20 FOR a=1 TO 79
30   LOACTE a,10
( 40   !FRAME   )
50   PRINT " X"
60   FOR b=1 to 5:NEXT b
70 NEXT a
80 GOTO 10
```

Mit dem FRAME-Befehl ist die Bewegung des Zeichens wesentlich gleichmäßiger und flackerfreier.

Die Befehle GPEN und GPAPER

Durch die Befehle GPEN und GPAPER wird Ihnen der Zugriff auf die beiden Vektoren GRA SET PEN (&BBDE) und GRA SET PAPER (&BBE4) unter BASIC ermöglicht. Die beiden Befehle sind in ihrer Funktionsweise den auf dem CPC 664/6128 vorhandenen Befehlen GRAPHICS PEN und GRAPHICS PAPER nachempfunden.

```
!GPEN [,<Farbstift>]
!GPAPER [,<Farbstift>]
```

Der LINE Befehl

Der LINE-Befehl kombiniert die beiden BASIC-Kommandos PLOT und DRAW miteinander. Er zeichnet, wie der Name ja vermuten lässt, eine Linie. Die vollständige Syntax sieht folgendermaßen aus:

```
!LINE,<X1>,<Y1>,<X2>,<Y2> [,<Farbstift>]
```

```
Beispiel: 10 MODE 1
          20 FOR a=0 TO 400 STEP 20
          30   !LINE,a,399,639,399-a
          40   !LINE,0,a,639-a,0
          50 NEXT a
```

Die Befehle RESET.ON und RESET.OFF

Jeder, der schon einmal in Maschinensprache programmiert hat, kennt diese Situation. In mühevoller Arbeit hat man eine mehr oder weniger große Routine in den Speicher gebracht. Nach dem eingegebenen CALL hängt sich der CPC auf und mit Entsetzen denkt man an das viel zu lang aufgeschobene Abspeichern des Source-Files. Eine hoffnungslose Situation ?

Nicht mit dem RESET.ON-Befehl. Geben Sie einmal im Direktmodus folgende Befehle ein:

```
!RESET.ON
POKE &8000,&18
POKE &8001,&FE
CALL &8000
```

Rien ne vas plus ! Doch halt, nicht gleich zum Netzschalter greifen, denn wir haben ja vorher !RESET.ON eingegeben. Daß das gleichzeitige Drücken von CTRL-SHIFT-ESC in den meisten Fällen die gleiche Wirkung wie ein Aus- und Einschalten des CPCs zeigt, dürfte Ihnen ja bekannt sein. Also probieren Sie es einmal aus. Mit einem Piepston und mit Ready meldet sich der CPC zurück und ein PRINT PEEK (&8000) bestätigt es; der Speicher wurde nicht gelöscht ! Was ist passiert ?

!RESET.ON bewirkt beim Drücken von CTRL-SHIFT-ESC einen Sprung ins ALPHA-ROM. Dort wird dann eine RESET-Routine durchlaufen, die den CPC wieder in einen halbwegs geordneten Zustand bringt. Der Speicherinhalt bleibt dabei unberührt.

Mit !RESET.OFF wird der Original-Zustand wieder hergestellt. Das Drücken von CTRL-SHIFT-ESC löscht dann wieder wie gewohnt den Speicher.

Der GDUMP Befehl

Mit dem GDUMP-Befehl wird Ihnen eine sehr schnelle 'Mode-unabhängige' Hardcopy-Routine für EPSON kompatible Matrix-Drucker zur Verfügung gestellt. Über den optionalen Parameter kann man zusätzlich die Qualität (und die Geschwindigkeit) des Ausdruckes wählen.

!GDUMP [,<Druckqualität>]

Die Eingabe **!GDUMP,3** z.B. druckt jede Zeile 3 mal, wodurch auch mit altersschwachen Farbbändern noch ein gutes Ergebnis erzielt werden kann. Wird GDUMP ohne Angabe eines Parameters aufgerufen, wird automatisch der *Default*-Wert 1 angenommen.

Die Bankman-Befehle BANKFIND, BANKOPEN, BANKREAD und BANKWRITE

Die Befehle BANKFIND, BANKOPEN, BANKREAD und BANKWRITE gehören neben SCREENCOPY, SCREENSWAP, VIEW, VDU und VDU0 zu den sogenannten *Bankman*-Befehlen.

Das Programm *Bankman* ist ein Bestandteil der CPC 6128 System-Software und ermöglicht es, mit eben diesen Befehlen, die zweiten 64 KByte Speicher des CPC 6128 zu verwalten.

Die drei Befehle VIEW, VDU und VDU0 dürften jedoch selbst dem CPC 6128 Besitzer unbekannt sein, da sie im Handbuch des CPCs nicht beschrieben sind und auch nur durch einen *kleinen Trick* zum Leben erweckt werden können.

Die Befehle sind in ihrer Funktion identisch mit denen des *Bankman*-Programms. Im Gegensatz zum Original benötigen sie jedoch keinen Speicherplatz (für Eingeweihte: Als Zwischenspeicher beim Kopieren oder Tauschen von Speicherbereichen wird der 256 Byte große EDIT-Puffer der CPC's benutzt), und ihre Funktionsfähigkeit ist auch nicht auf den CPC 6128 beschränkt. Wer also z.B. eine 64 KByte Speichererweiterung von *dk'tronics* besitzt, kann die *Bankman*-Befehle ohne Einschränkungen auch auf einem CPC 464 oder 664 benutzen. Selbst ohne Speichererweiterung kann man die Befehle SCREENCOPY, SCREENSWAP, VIEW und VDU, zumindest eingeschränkt, verwenden.

Doch nun zu den Befehlen BANKFIND, BANKOPEN, BANKREAD und BANKWRITE. Mit diesen Befehlen kann man die zweiten 64 KByte als 'Stringvariablenspeicher' mit 'Random access' (= wahlfreier Zugriff) verwalten. Dieser Stringspeicher wird über den Befehl BANKOPEN in gleichlange 'Records' (= Abschnitte) unterteilt. Über die Befehle BANKREAD und BANKWRITE können dann beliebige Abschnitte gelesen oder geschrieben werden. Mit BANKFIND hat man zusätzlich noch die Möglichkeit, bestimmte oder alle Records nach einem bestimmten String zu durchsuchen.

Bevor nun die einzelnen Befehle erklärt werden, sollen hier noch einmal ein paar allgemeingültige Hinweise erwähnt werden.

- Die Namen der Variablen sind lediglich Beispiele; statt 'r%' könnte z.B. auch 'a%' oder 'x%' stehen.
- In eckigen Klammern stehende Parameter deuten an, daß diese Parameter wahlweise verwendet werden können.
- In spitzen Klammern stehende Ausdrücke haben lediglich eine erklärende Funktion. Sie müssen durch den entsprechenden Ausdruck ersetzt werden.

Mit

!BANKOPEN,<Recordlänge>

wird die *Recordlänge* für alle folgenden BANKREAD, BANKWRITE und BANKFIND Befehle auf den angegebenen Wert festgelegt und die *aktuelle Recordnummer* auf den Startwert Null gesetzt. Die angegebene Länge muß in den Grenzen von 1 bis 255 liegen.

Mit

!BANKWRITE,@r%,@a\$ [,<Recordnummer>]

wird der in der Variablen 'a\$' stehende String in den zweiten 64 K abgelegt. Die <Recordnummer> gibt an, welcher Record beschrieben werden soll. Wird dieser Parameter nicht angegeben, dann wird die *aktuelle Recordnummer* angenommen. Die *aktuelle Recordnummer* wird nach dieser Operation automatisch auf den nächsten Record erhöht. Ist der angegebene String kürzer als die bei BANKOPEN festgelegte *Recordlänge*, so werden die Zeichen am Ende des alten Recordinhalts nicht verändert. Ist der String zu lang, so werden die überschüssigen Zeichen ignoriert. In der Variablen 'r%' wird nach erfolgreicher Ausführung des Befehls die Nummer des Records abgelegt, der beschrieben wurde. Hat die Schreib-Operation aus irgendeinem Grund nicht geklappt, wird ein negativer Fehlercode übergeben.

- 1 'End of file' -Fehler. Die Adresse der geforderten Satznummer liegt außerhalb der zweiten 64 K.
- 2 'Banking' -Fehler. Es war zur Zeit der Befehlsausführung eine andere als Bank 0 aktiv (siehe auch VDU0 Befehl).

Die Variable 'r%' muß vor dem Aufruf des Befehls definiert werden (z.B. mit 'r%=0').

Mit

!BANKREAD,@r%,@a\$ [,<Recordnummer>]

wird der Inhalt des angegebenen Records in der Stringvariablen 'a\$' abgelegt. Wird der Parameter <Recordnummer> nicht angegeben, dann wird die *aktuelle Recordnummer* angenommen. Die *aktuelle Recordnummer* wird nach dieser Operation automatisch auf den nächsten Record erhöht. Ist die Stringvariable länger als der zu lesende Record, so werden die Zeichen am Ende des Strings nicht verändert. Enthält der Record mehr Zeichen, als die Stringvariable aufnehmen kann, werden die überschüssigen Zeichen ignoriert. In der Variablen 'r%' wird nach erfolgreicher Ausführung des Befehls die Nummer des Records abgelegt, aus dem gelesen wurde. Hat die Lese-Operation aus irgendeinem Grund nicht geklappt, wird ein negativer Fehlercode übergeben.

- 1 'End of file' -Fehler. Die Adresse der geforderten Satznummer liegt außerhalb der zweiten 64 K.
- 2 'Banking' -Fehler. Es war zur Zeit der Befehlsausführung eine andere als Bank 0 aktiv (siehe auch VDU0 Befehl).

Die Variable 'r%' muß vor dem Aufruf des Befehls definiert werden (z.B. mit 'r%=0').

Mit

!BANKFIND,@r%,@a\$ [, <Start-Record> [, <End-Record>]]

werden alle Records in dem angegebenen Bereich mit dem in 'a\$' stehenden Suchstring verglichen.

Der Suchstring darf 'Wildcards' (= Universalzeichen) in Form von Null-Bytes (= CHR\$(0)) enthalten. Die Suche beginnt bei dem mit <Start-Record> angegebenen Record. Wird dieser Parameter nicht angegeben, dann beginnt die Suche bei der *aktuellen Recordnummer*. Ist zusätzlich der Parameter <End-Record> angegeben, dann wird die Suche spätestens dann abgebrochen, wenn dieser Satz geprüft wurde.

War die Suche erfolgreich, dann wird der Record, in dem der gesuchte String gefunden wurde, zum *aktuellen Record*, und seine Nummer wird in der Variablen 'r%' abgelegt. Hat die Such-Operation aus irgendeinem Grund nicht geklappt, wird in 'r%' ein negativer Fehlercode übergeben.

- 1 'End of file' -Fehler. Die Adresse der geforderten Satznummer liegt außerhalb der zweiten 64 K.
- 2 'Banking' -Fehler. Es war zur Zeit der Befehlsausführung eine andere als Bank 0 aktiv (siehe auch VDU0 Befehl).
- 3 'Not found' -Fehler. Der gesuchte String konnte in dem angegebenen Bereich nicht gefunden werden.

Die Variable 'r%' muß vor dem Aufruf des Befehls definiert werden (z.B. mit 'r%=0').

Die beiden folgenden kleinen Beispielprogramme sollen die Benutzung der *Bankman*-Befehle demonstrieren:

Das erste Programm füllt die gesamten zweiten 64 K mit Leerzeichen. Dazu wird die aus 128 Leerzeichen bestehende Stringvariable 'a\$' angelegt und in die 512 gleichlangen Records des Zusatzspeichers geschrieben. Der Parameter 'n%' in Zeile 60 kann auch weggelassen werden, da nach dem Befehl BANKOPEN die *aktuelle Recordnummer* auf Null gesetzt und nach jedem BANKWRITE automatisch erhöht wird.

```
10 reflen=128:maxrec=511:r%=0
20 a$=STRING$(reflen," ")
30 !BANKOPEN,reflen
40 '
50 FOR n%=0 to maxrec
60 !BANKWRITE,@r%,@a$,n%
70 NEXT n%
```

Das zweite Programm (auf der nächsten Seite) demonstriert die Verwendung des BANKFIND-Befehls. In der ersten Hälfte des Programmes (Zeile 100 - 160) werden die zweiten 64 K mit 4 stelligen HEX-Zahlen gefüllt. Das dauert einige Zeit, aber schließlich müssen 16384 HEX-Zahlen erzeugt und gespeichert werden. Im zweiten Teil des Programmes (Zeile 170 - 250) wird dann der Speicher nach allen Records durchsucht, die auf "12" enden. Den Suchstring können Sie in Zeile 170 bei Bedarf auch ändern. Alle Records, die dem Suchstring genügen werden dann in den Zeilen 210 - 230 gelesen und ausgegeben. Beachten Sie bitte, daß vor dem ersten BANKREAD-Befehl der String 'a\$'

in der richtigen Länge definiert werden muß (Zeile 180).

```
100 reclen=4:maxrec=16383:r%=0
110 :BANKOPEN,reclen
120 '
130 FOR n%=0 to maxrec
140   :BANKWRITE,@r%,HEX$(n%,4)
150 NEXT n%
160 '
170 search$=chr$(0)+chr$(0)+"12"
180 :BANKFIND,@r%,@search$,0:a$="...."
190 '
200 WHILE r%>0
210   PRINT "Record ";r%;" : ";
220   :BANKREAD,@r%,@a$,r%
230   PRINT a$
240   :BANKFIND,@r%,@search$,r%+1
250 WEND
```

Die Bankman-Befehle SCREENCOPY und SCREENSWAP

SCREENCOPY und SCREENSWAP sind zwei weitere Vertreter der *Bankman*-Befehle. Ihre Aufgabe ist das Kopieren (SCREENCOPY) und Tauschen (SCREENSWAP) von Speicherbereichen.

Die Speicherbereiche, die über die beiden Befehle angesprochen werden können, werden als *Screens* (= Bildschirm) bezeichnet. Insgesamt sechs solcher *Screens* (*Screen 0* ... *Screen 5*) mit einer Länge von jeweils 16 K stehen Ihnen zur Verfügung.

Screen 0 und *Screen 1* sind dabei in den ersten 64 K (*Screen 0* von &4000 - &7FFF und *Screen 1* von &C000 - &FFFF). Die anderen vier *Screens* befinden sich in den zweiten 64 K (*Screen 2* von &0000 - &3FFF, *Screen 3* von &4000 - &7FFF, *Screen 4* von &8000 - &BFFF und *Screen 5* von &C000 - &FFFF).

Da zumindest *Screen 0* und *Screen 1* in den ersten 64 K liegen, sind die beiden Befehle auch auf dem CPC 464/664 eingeschränkt anwendbar.

Der Befehl

```
:SCREENCOPY [,<Abschnitt>] ,<Ziel-Screen>,<Quell-Screen>
```

kopiert den Inhalt eines Speicherbereiches in einen anderen, während der Befehl

```
:SCREENSWAP [,<Abschnitt>] ,<Screen-Nr.>,<Screen-Nr.>
```

die Inhalte zweier Speicherbereiche tauscht. Über den optionalen Parameter <Abschnitt> ist es möglich, die beiden Befehle auf einen 256 Byte großen Teilbereich zweier *Screens* zu beschränken. Für Abschnitt können Werte von 0 - 63 eingesetzt werden. Das Tauschen der *Screens* kann auf diese Weise in 64 einzelne Operationen zerlegt werden. Dieser Modus ist z.B. dann sinnvoll, wenn neben dem Tauschen oder Kopieren von Speicherbereichen noch ein anderer Prozeß (z.B. mit **EVERY xx GOSUB**) ablaufen soll. Die Parameter <Ziel-Screen>, <Quell-Screen> und <Screen-Nr.> stehen für eine der 6 möglichen *Screen*-Nummer 0 bis 5.

Das folgende Beispielprogramm lädt sechs Bilder von Diskette und speichert sie in den Screens 0 - 5 (Zeile 150 - 230). Im zweiten Teil des Programms werden die Bilder dann der Reihe nach angezeigt (Zeile 250 - 310). Dieser Vorgang kann durch einen Tastendruck abgebrochen werden. Zuvor müssen allerdings erst einmal sechs (unterschiedliche) Bilder (**BILD0.PIC ... BILD5.PIC**) auf Diskette gespeichert werden. Die Wirkungsweise des - in diesem Programm vorkommenden - VIEW-Befehls wird im nächsten Abschnitt erklärt.

```

100 OPENOUT "dummy"
110 IF HIMEM>&3FFF THEN MEMORY &3FFF
120 CLOSEOUT
130 MODE 2: !SCREENCOPY,0,1
140 '
150 FOR bild%=5 TO 0 STEP -1
160   screen%=bild% MOD 2 XOR 1
170   !VIEW,screen%
180   name$="BILD"+CHR$(bild%+48)+".PIC"
190   IF screen%=0 then adr%=&C000
200   IF screen%=1 then adr%=&4000
210   LOAD name$,adr%
220   !SCREENCOPY,bild%,screen% XOR 1
230 NEXT bild%
240 '
250 WHILE INKEY$=""
260   FOR bank%=5 TO 2 STEP -1
270     !VIEW,bank% MOD 2 XOR 1
280     !SCREENSWAP,bank%,bank% MOD 2
290     FOR pause=1 TO 1000:NEXT
300   NEXT bank%
310 WEND

```

Die Bankman-Befehle VIEW und VDU

Diese beiden *Bankman*-Befehle sind - ebenso wie VDU0 - zwar im Original ebenfalls vorhanden, können dort jedoch nur dann verwendet werden, wenn das (geschützte) Start-Programm **BANKMAN.BAS** geringfügig modifiziert wird; dazu muß lediglich die BASIC-Zeile 190 in **190 CALL mcentry,x** geändert werden.

Die beiden Befehle VIEW und VDU haben eigentlich nichts mit den zweiten 64 K zu tun und sind deshalb auch **ohne Einschränkungen** auf einem CPC 464/664 einsetzbar. Sie ermöglichen die Verwaltung von zwei unabhängigen Bildschirmseiten. Die Syntax der beiden Befehle sieht folgendermaßen aus:

```

!VIEW,<Screen-Nr.>
!VDU,<Screen-Nr.>

```

Mit dem VIEW-Befehl kann der Screen für die Bildschirmanzeige und mit dem VDU-Befehl der Screen für die Bildschirmausgabe gewählt werden. Gültige Werte für den Parameter <Screen-Nr.> sind hier jedoch nur 0 und 1. Der Befehl **!VDU,0** z.B. verlegt den Bildschirmspeicher in den Bereich von &4000 - &7FFF. Wird anschließend noch der Befehl **!VIEW,1** ausgeführt, dann erfolgt die Bildschirmausgabe auf *Screen 0*, während jedoch *Screen 1* (&C000 - &FFFF) angezeigt wird.

Der Bankman-Befehl VDU0

Der Befehl VDU0 ist der letzte der insgesamt neun *Bankman*-Befehle. Über ihn kann man eine der vier 16 K Bänke der zweiten 64 K in den Adressbereich der CPU (&4000 - &7FFF) einblenden. Der Befehl hat folgende Syntax:

:VDU0,<Screen-Nr.>

Für <Screen-Nr.> sind die Werte 0 (=Normalzustand) und 2 - 5 zulässig. Die so ausgewählte Speicherbank (16 KByte) wird immer in den Bereich &4000 - &7FFF der ersten 64 K eingeblendet. Der Wert 1 ist als Parameter unzulässig und erzeugt die Fehlermeldung **Improper Argument**.

Beachten Sie bitte, daß die Befehle BANKFIND, BANKOPEN, BANKREAD und BANKWRITE nur im Normalzustand (nach :VDU0,0) fehlerfrei arbeiten.